A knowledge based system to assist in the selection of appropriate geotechnical field tests

Marina Moula

A KNOWLEDGE BASED SYSTEM TO ASSIST IN THE SELECTION OF APPROPRIATE GEOTECHNICAL FIELD TESTS

A thesis submitted to the

School of Engineering and Computer Science

University of Durham

for the degree of

Doctor of Philosophy

by

Marina Moula

March, 1993

To my parents

DECLARATION

I hereby declare that the work reported in this thesis has not been previously submitted for any degree. All material in this thesis is original except where indicated by reference to other work.

STATEMENT OF COPYRIGHT

ABSTRACT

The variety in geological conditions and range of geotechnical problems has led to the development of a considerable number of different in-situ test methods. The correct selection of the appropriate in-situ tests allows a safer and cost-efficient design to be achieved.

A prototype Knowledge-Based System has been developed to assist in the selection of appropriate geotechnical in-situ tests. The system is model-based and has been implemented using PDC Prolog on a Personal Computer to perform two functions: i) general querying of the knowledge bases which it incorporates and ii) advise on selecting in-situ tests.

The system consists of two knowledge bases (the Ground Knowledge Base and the Tests Knowledge Base), an Extended Inference Mechanism consisting of search rules developed to allow inheritance and transitivity inferences (as well as information retrieval facilities), an advisory rule developed for offering assistance in the selection of appropriate field tests, and a menu driven user interface to achieve ease of use. The Extended Inference Mechanism, and the user interface implemented for it, form a basic expert system shell.

The knowledge required to be included in the system was obtained in two ways: i) from technical literature and ii) from a small knowledge elicitation exercise in the form of a questionnaire. The representation scheme adopted is the same for both knowledge bases and allows modifications (additions or deletions) of the existing knowledge to be easily made.

Towards the completion of this research program, a comparative exercise was performed by re-implementing part of the system using the PROKAPPA software on a Sun Sparkstation 2 (both of which became available at that time). Throughout this exercise, the differences between the two implementation schemes were evaluated and the advantages and disadvantages of each of the schemes were identified.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr D.G.Toll, for his support and guidance throughout the course of this work and for our collaboration during my time spent in Durham as an Erasmus student. It was the prospect of undertaking further studies with Dr. Toll that encouraged me to return to Durham for postgraduate research.

Also, I would like to thank the members of staff of the School of Engineering and Computer Science, Applied Mechanics Division, and in particular Professor P.B Attewell for his youthful and humorous attitude. During my stay in England on the Erasmus scheme the helpfulness and understanding of Professor Attewell were crucial factors in my decision to return to Durham University for further studies. Foremost, I have been very fortunate to work with such a distinguished scientist.

Bernard McEleavey's continual humorous assistance during the laboratory demonstration ensured that they were both instructive and entertaining. In addition, the warmth and generosity of Bernard and his wife Lesley gave me a jolly insight into English family life and reduced my feelings of being away from home.

I would like to thank Trevor Nancarrow for his computing support whenever needed and Wendy Lister for her administrational assistance and for being always caring. My thanks also go to Brian Scurr, Alan and Judith Swann, Steve Richardson and, Lesley and Olive Graham.

It is difficult to express the depth of my feelings and gratitude to Nikitas Vaptismas. In any aspect of my life throughout my time in Durham, his presence and continuous support were of greatest importance to me. Nikita, I know you said it first, but I was very fortunate in having you share my office, my work, my thoughts and most importantly my life.

My sincerest thanks go to Andy Oliver for his friendship and non-stop assistance. I would like to acknowledge Andy for our collaboration during the development of the PROKAPPA application presented in this thesis. Working with Andy was both productive and entertaining. Andy, stay as you are!

Many thanks go to Kalliope Tsarouhi whose companionship was invaluable during the writing up of our theses; we shared a tiring but enjoyable experience. In particular, I found our lengthy late night conversations about the intricates of life both amusing and enlightening.

Thanks also to my friends, Antonis Giolas, for his companionship, sincerity and warmth.; Nicholas Antoniu for his thoughtfulness, generosity and artistic sensibilities; Panagiotis Dounis for his D.J.abilities, support and advices; Elias Papadimitropoulos for his indie style and outlook; Vicky Malandraki for her sweet taste and company and Alex Bedford for her independent and optimistic attitude.

I would also like to thank Sandra Mavroidi, Kostas Panou, Aris Koubarelis and Kim Nissan whose friendship made my residence in Durham more interesting. In addition, my thanks go to my colleagues in the department and all the friends I made in Durham. I also thank Paulette for her loyalty and outstanding performance and Ikaria for being what it is.

Finally, I would like to express my sincere thanks to Professor A. Anagnostopoulos of the National Technical University of Athens for giving me the opportunity to study for some period in Durham under the Erasmus scheme and become familiar with university life in England. Also I would like to thank Professor A. Anagnostopoulos for his support and guidance before and during the course of this work.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 General

Geotechnical Engineering involves the study of earth materials for construction purposes. Natural materials (soil and rock) are highly variable and complex and often have properties which are undesirable from the point of view of a proposed structure. The decision to develop a particular site cannot often be taken on the basis of its complete suitability from engineering point of view; therefore geotechnical problems occur and require geotechnical parameters for their solution.

The objective of any subsurface exploration program is to determine the stratigraphy and the relevant physical properties of the ground that are appropriate to the project. This can be achieved by in-situ testing (in conjuction with laboratory testing) which is a major source of both qualitative and quantitative data relating to ground conditions and forms an essential part of a site investigation programme.

The variety in geological conditions and range of geotechnical problems has led to the development of a considerable number of different in-situ test methods. Correct selection of appropriate in-situ tests allows a more efficient and cost-effective design to be performed.

The fundamental aim of this research project is to apply knowledge-based system technology to an area of geotechnical engineering that involves the selection of suitable field testing techniques. Knowledge-based system technology provides a medium that can accommodate the representation and use of the knowledge required to allow successful engineering decisions to be taken.

A Knowledge-Based System has been developed to provide assistance in the selection of appropriate in-situ tests. The development of this system involved the identification, collection and representation of the domain knowledge (information on both ground and in-situ tests had to be obtained to satisfy the requirements of the system), the design and implementation of the process that makes use of the available knowledge and finally the design and implementation of a user interface to facilitate the use of the system. The derivation and organisation of the domain knowledge is considered to be the major contribution of the system in the area of geotechnical engineering. The system has been implemented using PDC Prolog on a Personal Computer.

Towards the completion of this research project the PROKAPPA software and a Sun Sparkstation 2 became available, thus enabling a comparative exercise to be carried out by implementing part of the system (the ground information) in PROKAPPA as well.

A brief description of the contents of the chapters to follow, is presented in the next section.

1.2 Overview of the Thesis

In Chapter 2, the basic concepts of knowledge-based system technology are outlined and a comprehensive review of the existing applications of this technology in geotechnical engineering is presented. A general discussion on the development of these systems then follows.

Chapter 3 is concerned with the design and implementation of the 'Representing the Ground' application using PDC Prolog, which involved the development of the Ground Knowledge Base as well as the development of the process that manipulates the knowledge included in the knowledge base. The Ground Knowledge Base incorporates a model of the ground which corresponds to the needs of the knowledge-based system for assisting in the selection of appropriate field tests. A brief description of the main characteristics of the Prolog programming language and the Prolog 'dialect' that was selected as the implementation language of the system is also presented.

The design and implementation of the 'Representing Geotechnical Field Tests' application using PDC Prolog is presented in Chapter 4. The in-situ tests hierarchy incorporated in the Tests Knowledge Base is described and its subsequent development discussed. The knowledge concerning individual test methods, required to be included in the knowledge base, is identified and the knowledge elicitation exercise carried out to obtain this knowledge is presented. An integral part of this chapter is concerned with brief descriptions of the tests included in the hierarchy; however, due to its size this is presented separately in an appendix (Appendix D).

An overview of the knowledge-based system developed to aid the selection of in-situ tests is given in Chapter 5. The parts that constitute the system are described, i.e. the Ground and Tests knowledge bases, the process that manipulates the knowledge (consisting of the generic rules, the Extended Inference Mechanism and the advisory rule) and the user interface. At the end of the chapter, example consultations with the system are presented.

In Chapter 6, a comparative exercise is carried out by implementing the 'Representing the Ground' application using the PROKAPPA software as well as PDC Prolog. Initially the main features of the PROKAPPA system are described in order for the reader to become familiar with the capabilities of the system and the terminology used. The actual implementation of the application is then described and example consultations are given. Finally, the two implementation schemes are discussed in a comparative way.

Chapter 7 consists of a general discussion of the work presented in this thesis. The main features of the system are briefly reviewed, identifying possible future improvements.

Finally, the conclusions reached from the development of the knowledge-based system to assist in the selection of appropriate in-situ tests and the comparative exercise between PDC Prolog and the PROKAPPA system are presented in Chapter 8.

CHAPTER 2

APPLICATION OF KNOWLEDGE-BASED SYSTEM TECHNOLOGY IN GEOTECHNICAL ENGINEERING

2.1 Introduction

Civil engineering is not only concerned with calculation and numeric analysis but also with ideas, concepts, judgement and the deployment of experience which cannot be represented numerically. Geotechnical engineering is the area of civil engineering most recognised for the use of expert knowledge. The following quote by Peck (Tomlinson, 1986) expresses the view that knowledge of precedents (experience) plays an important, and often decisive role, in the decision making process in geotechnics:

> " The everyday procedures now used to calculate bearing capacity, settlement, or factor of safety of a slope, are nothing more than the use of the framework of soil mechanics to organise experience. If the techniques of soil testing and the theories had not led to results in accord with experience and field observations, they would not have been adopted for practical, widespread use. Indeed, the procedures are valid and justified only to the extent that they have been verified by experience. In this sense, the ordinary procedures of soil mechanics are merely devices for interpolating among the specific experiences of many engineers in order to solve our own problems, or which we recognise to fall within the limits of previous experience. "

Knowledge-Based Systems (KBS) are computer programs that contain domain-specific knowledge (facts and/or heuristics) and employ a separate inference procedure to manipulate this knowledge in order to solve a real-world problem. If these systems operate at an expert's level they are called Expert

Systems (ES) (Mullarkey, 1987; Adeli *et al*, 1988; Konigsberger and De Bruyn, 1990). Although these terms are often used as synonyms in the literature, the term knowledge-based system is considered to better represent most current systems.

Toll (1990) discusses the role of KBS in geotechnical engineering. Although knowledge-based system technology seems to be the right approach in order to overcome the limitations of traditional computing, it has not, as yet, had any major impact in geotechnical engineering. This is due to a number of reasons (Adams *et al*, 1989), among which is the fact that most of the systems developed have not reached yet a point where they can be distributed for practical use.

This chapter is concerned with the application of knowledge-based system technology in geotechnical engineering problems. A brief account of KBS fundamentals is presented in section 2.2 and a comprehensive review of existing KBSs applied in geotechnical engineering is given in section 2.3. Finally, in section 2.4 a general discussion on the development of these systems is presented.

2.2 Knowledge-Based Systems (KBS)

2.2.1 Introduction

Knowledge-based system technology forms an area of research within Artificial Intelligence (AI), a branch of computer science concerned with simulating human intelligence in a computing machine.

Various definitions of KBSs exist in the AI literature; Adeli (1988) presents some of them. Maher and Allen (1987) note that the definitions which are often given for KBS do not necessarily distinguish them from many conventional computer programs. Some of the distinguishing characteristics between the new technology and the traditional programs are presented by Adeli (1988), Maher and Allen (1987) and Κρικετο και Παστρα (1991).

For the purposes of this analysis, it is worth emphasising two of their differences: a) KBS are orientated towards symbolic processing whereas conventional programs are efficient in numerical processing and b) In KBS the knowledge is separated from the inference procedure (declarative programming) in contrast to the traditional programs where knowledge and control are integrated (procedural programming). The main advantages of the latter characteristic of these systems is the transparency in programming and the ability to alter (add, delete or modify) the content of the knowledge base without significantly affecting the remainder of the program.

2.2.2 Architecture of a KBS

In general a KBS consists of three main components:

- *Knowledge base*: the component of a KBS that contains all the information associated with the domain in which the system is applied. This information may be documented definitions, facts and rules as well as rules of thumb and heuristics.
- *Context* (also known as *working memory, short term memory or fact base*): the component of a KBS that contains all the information about the problem currently being solved. Its content changes dynamically and includes information that defines the parameters of the specific problem and information derived by the system at any stage of the solution process.
- *Inference mechanism* (also known as *inference engine, control mechanism* or *reasoning mechanism*): the component of a KBS that controls the reasoning process of the system. The inference mechanism uses the knowledge base to modify and expand the context in order to solve a specific problem.

Additional components such as a *user interface* and an *explanation facility* are required in order to facilitate the use of a KBS and make the knowledge base more transparent to the user. A *knowledge acquisition* facility is also desirable in order to ease the development of the knowledge base.

A variation of the basic architecture described above is the *blackboard model*, which is based upon the separation of the knowledge base into independent knowledge sources and the use of a blackboard as a dynamic global database (context), through which the knowledge sources communicate. The blackboard monitors the changes made in the problem state until a solution is found.

2.2.3 Implementation of a KBS

Implementation of a KBS involves the choice of formalisms for the representation of the domain knowledge and the inference models. These two topics are still very active areas of research in AI and are discussed by Adeli (1988), Maher and Allen (1987), Mullarkey (1987), Κρικετο και Παστρα (1991) and Benchimol *et al* (1987) among others. The most common forms of knowledge representation and inference mechanisms are briefly discussed below. Although the way in which the domain knowledge is represented can be discussed independently from the problem-solving strategy, these decisions are tightly coupled (Mullarkey, 1987).

Knowledge Representation

The main types of declarative knowledge representation are *logic-based representation, rules* and *network-based representation* (Mullarkey, 1987). In logic-based representation the knowledge is represented as assertions in logic. In rule-based representation the knowledge is represented in modular rules which consists of an IF part (situation or condition) and a THEN part (action); these rules are called production rules. In network-based representation, knowledge is represented as a collection of nodes and links between them, explicitly representing the connectivity and hierarchy between pieces of information. A special case of nodes, in a network-based representation, are *frames* that include not only particular properties (slots) with values, but also pointers to other frames or procedures.

Common Inference Mechanisms

The inference mechanism of a KBS can employ one or more problem-solving strategies to search for solutions. The two main inference mechanisms are *forward chaining* (also known as *data-driven* control strategy or *bottom-up* strategy) and *backward chaining* (also known as *goal-driven* strategy or *top-down* strategy). A forward chaining inference mechanism works from an initial state of known facts to a goal state (conclusion or conclusions). A backward chaining inference mechanism assumes a goal state or hypothesis and reasons back to known data or facts to support or discount the assumed hypothesis. A combination of the two strategies described above, called *mixed chaining* inference mechanism, can also be used (hybrid approach).

The strategies described above identify the rules that are applicable to a specific problem and can be combined with other control strategies such as *breadth first search* and *depth first search* for selecting the order in which the applicable rules should be activated. In a breadth first search all the applicable rules are executed in turn before testing whether the halt condition has been satisfied, while in a depth first search the first of the applicable rules is exhaustively explored before examining the next one. However, both strategies are guaranteed to consider all possibilities.

A closely related concept to those outlined above is uncertainty in data and inference. Adeli (1988) has discussed various methods that have been employed to deal with uncertain or incomplete information in the knowledge base. The manipulation of uncertain and imprecise knowledge requires appropriate models of inference (Mullarkey, 1987; Benchimol *et al*, 1987).

2.2.4 Tools for Developing a KBS

The tools which are available for developing a KBS can be divided into three main categories along a spectrum of software complexity: a) General Purpose Programming Language (GPPR), b)

General Purpose Representational Languages (GPRL) and c) Expert System Shells (Mullarkey, 1987). Expert System Development Environments might be added to the upper range of this spectrum.

The first category includes the conventional procedural languages such as Fortran, C, Pascal etc. A number of KBSs have been developed in procedural languages since they offer easy portability among different types of computers and compatibility with numerous pieces of software available in these languages (Adeli, 1987). However, as these languages are mainly orientated towards numerical algorithmic computation they do not provide the most appropriate environment for the development of KBS.

In the second category, symbol manipulation languages are included that have been developed for use in building KBS. These languages (AI languages) are declarative languages in which information is presented in a descriptive form. The most popular AI programming languages are LISP (LISt Programming) and PROLOG (PROgramming in LOGic).

LISP is the most widely-used language among AI researchers in the United States and was one of the first languages to be directed toward symbolic representation and list processing (Adeli, 1988). PROLOG is a symbolic programming language based on predicate logic. It allows information to be specified in a declarative style and includes a backward-chaining inference mechanism. The Prolog language is discussed in more detail in section 3.3.2.

Another class of programming languages, the object-orientated languages, have recently been the subject of very active research work in AI (Benchimol, 1987; Adeli,1988). An object-orientated language is a language which in principle handles only autonomous entities of a single type called objects. Each object is defined by data specific to it (its characteristics) as well as operations and computations that it is able of executing when a message is sent to it.

Expert System Shells, which form the third category of tools, are software packages recently developed in order to aid in the rapid prototyping of application KBSs. They consist of two of the three main components of an expert system, i.e. an inference engine and a user interface. They usually provide one or more knowledge representation forms and inference mechanisms. Expert system shells are easier to use than AI programming languages but are less flexible. Adeli (1988), Κρικετο και Παστρα (1991) and Benchimol *et al* (1987) describe some of the more popular expert system shells. Allwood *et al* (1987) draw attention to some experiences gained from evaluating a number of commercially available expert system shells.

Expert System Development Environments are usually fully developed system building workbenches providing capabilities (such as integrated editors, maintenance tools, debugging tools for all types of available data representations, user interface development facilities, etc.) which are additional to those provided by shells.

Detailed analysis of the fundamental characteristics of KBSs, the available techniques for their development as well as their capabilities and potential applications are presented in the published literature (Maher, 1987; Adeli, 1988; Κρικετο και Παστρα, 1991; Benchimol *et al*, 1987).

2.3 Knowledge-Based Systems in Geotechnical Engineering

A number of KBSs have been developed that demonstrate the potential application of knowledge-based system technology to problems encountered in geotechnical engineering. These systems are briefly presented in this section, grouped into categories according to the areas of geotechnical engineering to which they are applicable. In each group a chronological order has been followed.

2.3.1 Site Characterisation

The term site characterisation is used here to describe the process by which geological, geotechnical and other information relevant to the construction of a particular facility is determined. Knowledge-based systems have been extensively developed to assist in the task of site characterisation.

Smith and Barker (1983) present an interactive system, the Dipmeter Advisor, that uses dipmeter patterns (sequences of dip estimates from a dipmeter log, obtained by using a dipmeter tool) together with knowledge about local geology to infer subsurface geologic structure. The system is made up of: i) a knowledge base consisting of 90 production rules grouped into several distinct sets according to their function (e.g. structural vs. stratigraphic rules), ii) a forward chaining inference engine that resolves conflicts by rule order, iii) a set of feature detection algorithms for a preliminary interpretation of log data and, iv) a menu-driven graphical user interface. The Dipmeter Advisor is written in INTERLISP and operates on the Xerox 1100 Scientific Information Processor.

SITECHAR (Norkin, 1985; Rehak *et al*, 1985) is a KBS component of a geotechnical site characterisation workbench (that includes other components such as databases, workstations and graphics). The purpose of this expert system is to develop inferences on the depositional patterns of the subsurface materials and their physical properties by interpreting field and laboratory data and taking into account existing experience on geology and geomorphology of a specific site or similar ones. The system uses a complex problem solving technique, the *blackboard model* expert system framework. The initial SITECHAR system incorporates the following ruled-based knowledge modules: knowledge of geometry and trends, matching soils by description, proximity (such as "near", "above", etc.), geomorphology (such as erosional surfaces, channel cutting, etc.), geology (such as faults, folds, etc.) and searching for marker beds. Overall control, between the individual knowledge modules and the blackboard to allow an overall problem solution, is provided through a single co-ordinating knowledge-based supervisor. The inference engine supports both forward and backward chaining problem solving techniques.

CONE (Mullarkey, 1986; Mullarkey and Fenves, 1986) is a KBS that interprets raw data from the cone penetrometer (CPT) in order to perform an input and validity scan on the raw data, classification of the soil types (including the profiling of layers) and inference of design parameters with respect to the shear strength of sands and clays. The soils are classified using two electric-CPT based classification systems, the Dutch classification system and the Douglas and Olsen classification system. Another system was also used which is a fuzzy set representation based on the raw database used to develop the Douglas and Olsen system. The shear strength of sands and clays are estimated using empirically and rationally based methods. Fuzzy sets are employed to treat uncertainty with respect to linguistic data (i.e. soil classification), numeric data (i.e. determination of shear strength) and quality information (i.e. appropriateness of a soil classification system, the accuracy of the system for certain soil types etc.). The system has been implemented using OPS5 rules and LISP functions. A typical run of CONE may take up to 1.5 hours on a lightly loaded DEC-20.

SOILCON (Siller, 1987) is a KBS which has been developed for assisting the user in determining the levels of geotechnical investigation necessary for a specific problem. This is based on the requirements of a proposed structure and the level of information known about the site in order to reduce the risk involved with the subsurface to an acceptable level. The system was implemented using the M.1 rule-based expert system shell which provides a backward chaining control strategy. The knowledge base contains 24 investigation techniques ranging from preliminary (e.g. reviewing topographical maps) to more sophisticated (e.g. pressuremeter) that are used to make the ultimate recommendation. The complexity of the recommended investigation increases when there is a large amount of site data available. One of the limitations of the system is that it does not handle geometric descriptions of the problem and site quantitatively.

Alim and Munro (1987) present a very simple prototype KBS on soil investigation. It offers guidance on soil identification based on visual and physical observation of soil characteristics and provides judgement concerning the most likely foundation type under given soil and loading conditions. Based

on these two conclusions it gives possible foundation problems and finally it combines all this information to suggest the most suitable sampling and drilling techniques for the particular investigation scheme. The system was written in micro-PROLOG and uses the PROLOG expert system shell APES. The system handles uncertainty and imprecise knowledge using fuzzy logic to produce degrees of belief which take numerical values from zero to unity. The paper presented by Alim and Munro was discussed by Davey-Wilson, May and Tizani (1988) and some interesting comments arose such as the limitations of the software used (micro-PROLOG and APES) and the danger which can arise from using a numerical degree of belief (the system's solution will intrinsically suggest a higher degree of certainty than is warranted by some of the data).

SITECLAS (Wong *et al*, 1989) is an expert system used to classify a site according to the Australian Standard AS2870.1. The input required involves information about the natural soil or fill found at the site. This system was developed by using SUCAM, a custom-made expert system shell. SUCAM was built to explore the potential of applying expert system technology to geotechnical engineering by using a custom-made shell. It is written in TURBO PROLOG and runs on an IBM PC or compatible microcomputers under MS-DOS. Its main components are: i) a knowledge base, which stores the knowledge about a subject domain in the form of IF-THEN or IF-THEN-ELSE rules, procedures, tables and comments, ii) a fact base, which stores the consultation specifications, the input goals, the input facts and the conclusions of the consultation, providing the advantage of being able to modify the input facts without starting a new consultation, iii) an inference engine, based on backward chaining reasoning, iv) a user interface, which is screen-driven making the system user-friendly, v) an explanation facility, which allows Rule Explanation (why certain information is required), Rule file Explanation (how a certain conclusion was reached), Help File Explanation (for further explanations, comments, remarks, and notes) and vi) modules for different functions such as selecting the appropriate Rule File (an ASCII file storing the domain knowledge), reading the Rule File, reading and writing the Result File, specifying Consultation Control, goals and facts and showing results. SUCAM does not deal with imprecise, uncertain or conflicting knowledge.

LOGS (Adams *et al*, 1989) is a KBS based on the ideas introduced in SITECHAR (Norkin, 1985; Rehak *et al*, 1985), that treats information from several boring logs and provides the user with two dimensional subsurface profiles. It is a rule-based forward chaining system written in the languages OPS5 and Common LISP and implemented in the Knowledgecraft™ environment which provides a window and graphics interface for graphically displaying subsurface cross sections. Knowledge about geology and geomorphology is embodied in the system and is handled through heuristics that apply to a specific region (Kane County Illinois). The system tries to identify marker beds, lenses (wedge-shaped deposits) and lentils (strata with boundaries within the confines of the site). A soil may be identified as a continuous layer even if it is not present in all borings, based on the knowledge of the area's geology. The current version of LOGS comprises approximately 350 rules and future improvements suggested by the authors are three dimensional interpretation and calibration against the judgement of experts. The current version of the system is mainly site specific.

Smith and Oliphant (1991, 1992) describe a KBS for civil engineering site investigation. The primary requirement of the system was to act as an adviser during any stage of the site investigation process and especially during the planning stages (e.g. desk study, site reconnaissance etc.). The system has been implemented to run on an I.B.M. compatible P.C. supporting MS-DOS. It was developed using the shell Leonardo Development System, Level 3, produced by Creative Logic. The shell contains a text editor used to create rules for the knowledge-base and an inference mechanism which mainly uses the default technique of backward chaining, although forward chaining can be enforced where necessary. Leonardo uses rulesets, objects and object frames to represent the knowledge for an application. The system features a systematic data input facility in the form of multiple choice menus that helps minimise oversights or omissions of relevant data. The information obtained is used by the system to provide suggestions to the user on the following stage of a site investigation, the subsoil exploration (possible locations of boreholes, trial pits, etc. and suitable types of soil testing). The information obtained from the subsoil exploration stage is used to create a 2-D visual representation of the soil layers. The strength characteristics of the various soil strata are used by the system to make

recommendations for the suitable foundation types based on the ground conditions present. The prototype system is user friendly and can be used as a learning tool. It provides the facility for future expansion and, the authors suggest, it has a cost saving capability.

Halim *et al* (1991) describe a KBS developed to assist engineers in performing site exploration decisions and evaluation of geotechnical design concerning shallow foundations or slope stability, using probabilistic analysis within an interactive user-friendly environment. The prototype system was developed using the expert system development environment KEE on an Apollo DN3500 workstation. The system has been implemented to perform three major tasks: i) inference of prior estimates of soil and anomaly characteristics (such as lenses or pockets of soft soils within the regular soil deposit) using production rules, ii) selection of the most appropriate exploration program using probabilistic analysis where anomalies and soil properties are represented by a set of attributes such as probability of anomaly presence, and means and standard deviation of anomaly size and locations and iii) reliability evaluation of the proposed geotechnical system. The inference mechanism of the system is forward chaining and the knowledge incorporated is represented through a combination of frames and rules, that are both features of the expert system shell used. The system's functionality is similar to that of SOILCON (Siller, 1987) with additional capabilities to handle uncertainties of the ground conditions quantitatively.

A KBS framework is described by Carpaneto and Cremonini (1991) for the automation of the geotechnical design site characterisation process. The system is based on an existing KBS (Righetti and Cremonini, 1988) employed for stratigraphic soil characterisation. The system consists of several databases where information is stored about the site under consideration, a knowledge base containing the domain knowledge and an inference engine capable of interpreting the available data. The task of characterising the site is divided into the four phases: i) an Input Phase where information from the databases are used to make some preliminary inferences about the soil profile and its properties, ii) a Comparison Phase where rules are used to filter the data obtained in the previous phase and to improve on the possible soil profile, iii) a Reduction Phase where the construction of a best solution is carried out

and iv) an Output Phase where the best solutions detected for the borehole stratigraphy and the corresponding design parameters are processed for appropriate display of the results. Some possible future improvements of the system are also discussed, mainly for making soil profile inferences at sites where limited data is available but where there is a general knowledge of the area.

2.3.2 Foundation Engineering

Foundation engineering is an area where a number of systems are available. These systems could be further categorised according to the specific task of foundation engineering for which they aim to provide assistance.

General foundation design

FOOTER (Adams *et al*, 1989) is a KBS that performs design synthesis for building foundations and was also developed using the expert system shell EDESYN. The input to the system includes soil conditions, water table location, depth of bedrock and the imposed loading conditions from the structure. FOOTER decomposes the foundation design problem into several subproblems: i) selection of foundation type, ii) selection of material type, iii) selection of casting type (when appropriate), iv) selection of excavation type and iv) parametric design of foundation. The output comprises all feasible foundation alternatives which are then evaluated by the user.

Rowlinson (1989) briefly describes Geotech, a KBS under development to assist in foundation design in Hong Kong. The factors which should be considered during the development of the system and which determine its structure are stressed. These are technical, legal and commercial factors as well as local practice. The first module developed is a soil classification and foundation design module based on the UK CP2004/BS8004 recommendations, amended where needed to take into account Hong Kong Geoguide recommendations. An objective of the system is that all design must be constrained by all relevant regulations. At a final stage Geotech should be able to indicate where local practice is likely to

differ from code of practice procedures. The influence of cost/time trade-offs, plant availability, seasonal influences, safety, environmental effects should also be included.

Rashad *et al* (1991) present FOUNdation Design CONsultant (FOUNDCON), a modular knowledge-based Computer-Aided Design (CAD) system under development to assist in foundation design. Communication between modules is achieved using the "blackboard" architecture. The knowledge base consists of the resource level, where knowledge is in the form of computational methods (for bearing capacity, settlements etc.) and the expert's level, where knowledge is in the form of heuristics. The knowledge is represented through frames and slots that have production rules or procedures attached to them. Some of the problem-solving modules of the system (as these are envisaged), are: i) an Interpretation Module that provides a preliminary validity check of the input data and performs soil data interpretation, ii) a Preliminary Design Module that selects the most appropriate foundation system, iii) a Modelling and Analysis Module that models the structural configuration proposed above, and predicts its response to external conditions and iv) a Detailed Design Module that performs the final design, ensuring that all constraints are satisfied.

Meyer (1992) describes a KBS that addresses the preliminary foundation design of multi-story buildings using the expert system shell EDESYN which is based on hierarchical decomposition and constraint direct search. The system uses preliminary soil data (SPT-N value for cohesionless soils, undrained shear strength and Atterberg limits for fine grained soils) and the building's potential configuration, in order to characterise the underlying soil and to produce a set of feasible solutions to the preliminary foundation design problem. Economical alternatives are also considered. The implementation involves decomposing the problem into three major systems : i) building system, ii) soil system which is further decomposed into stratum systems and iii) foundation design system which is further decomposed into the three major foundation types; shallow, compensated and deep. Lisp functions have also been incorporated for the assignment of numerical values to dimensional or capacity attributes. Only static axial loads are considered.

Shallow foundations

FOOT (Yehia and El-Hajj, 1987) is a KBS to assist in the selection and design of spread footings. The program, implemented in FORTRAN-77, consists of four main modules, briefly described below.

i) MAIN is the program module concerned with the problem-specific data such as number, distribution and loading of the columns. The input is either directly from the user or through pre-prepared data files.

ii) DECIDE is the module corresponding to the inference engine of the program and receives the code matrix and column numbering from MAIN. It must be noted, that the columns distribution must be rectangular so if that is not the case, fictitious columns are incorporated in the site plan.

iii) GRAPH is the module that provides general plans of columns and footings, and also plots the reinforcement details for single and double footings, only for the best choice because of memory requirements.

iv) DESIGN performs the structural design after searching into its databank for similar cases. After every run of the program its database becomes larger and so in future problems the solutions should improve.

GEOTECH (Parikh and Kameswara Rao, 1991) is a KBS that was developed using COMMON LISP and can aid in shallow foundation design by calculating bearing capacity and settlement and producing the corresponding foundation design. It considers several properties of the ground, like soil type, and structural information, like load and column dimensions. The system incorporates the uncertainty involved in foundation design by using fuzzy logic. GEOTECH runs inside a geotechnical knowledge rich environment, SOILTECH, that can be reached at any time and includes soil data and information relating to the domain of shallow foundations. The system can handle missing information by using a special knowledge base created for that reason. It uses a forward-chaining inference mechanism and the output is in the form of a list of the most promising alternatives with corresponding confidence factors.

Pile selection

PILE (Santamarina and Chameau, 1987) is a prototype expert system developed to aid the selection of the appropriate type of pile foundation. The system's output is a list of the most promising alternatives based on technical constraints. It is then up to the user to consider additional factors (e.g. economical), in order to reach a final decision. PILE is a forward-chaining system written in LISP. It contains knowledge in the form of rules on subjects like: soil characteristics (chemical environment, groundwater conditions, interbedded soft layers, loose deposits and erratic stratigraphy), loads (per pile, components, design stress), installation conditions (drilling, driving), context (environmental problems, vibrations), material (wood, concrete, steel, composite), construction (predrilled, driven, cast in-situ), improvement (displacement, non-displacement). The system runs in a knowledge-rich environment that includes SOIL, a geotechnical database which can provide information on various aspects of geotechnical engineering (e.g. soil parameters, soil improvement methods) at any time during an execution or independently. PILE includes explanation capabilities, handles uncertainty, resolves conflicts in data memory and incorporates commands that allow its use in instruction. The performance of PILE has been successfully evaluated in a wide range of cases and its production system has been proven efficient and sufficient for small tasks.

Wong *et al* (1991) developed SUPILE, a rule-based KBS that assists in the evaluation of suitability of different types of piles and in the estimation of the required pile size and length. SUPILE consists of a Knowledge Base that contains pile design knowledge, a Fact Base that contains information about the site under consideration and where the results are stored, an Inference Engine where pile dimensions and suitability are estimated and a User Interface that consists of a Project File Manager, a Project Information Editor, a Default Values Editor and a Report and Diagram Generator. The selection of a pile type is performed by finding how many problems would exist if a specific type was used. These problems are quantified in the form of a problem score and finally a suitability score is produced for each pile type. It has a value between 0 and 99, where the higher the suitability score, the more suitable

the pile is. The system features a data-screen input method so that the user can input large amount of information or modify existing information easily and quickly. SUPILE is written in TURBO PROLOG and has been compiled as a stand alone program.

PILEX (Elton and Brown, 1991) is an expert system for assisting in the selection of reliable pile types by considering timber, concrete and steel piles. Spread footings are also considered, although they do not represent a pile type. PILEX was written using the expert system shell program VP-Expert on an IBM Personal Computer. The knowledge base contains information, in the form of rules, that was obtained from literature, combined with experts' (practitioners' and academicians') knowledge to take into account geotechnical, geological, structural and environmental factors that influence the pile selection. The inference engine that the shell supports is backward chaining. The system queries the user about loading parameters, soil condition and groundwater conditions. Some of the future improvements of PILEX are considered to be the inclusion of the cost parameter, lateral loads, heave of adjacent piles and sheet piles.

Bridge Foundations

BABE (Bridge and Building Evaluation) developed by Zheng *et al* (1989) is a KBS to help the user in preliminary investigations of a bridge substructure design. The main function of the system is to aid the selection of the most appropriate type of foundation for a specific superstructure and a set of site conditions. The system also makes suggestions for the superstructure design from a geotechnical point of view and covers the preliminary design of bridge abutments and piers. BABE was developed using the GEOTOX shell which consists of the inference mechanism and the user interface of the GEOTOX knowledge-based system developed for evaluating waste disposal sites (Wilson *et al*, 1987). Some modifications and additions have been made to the inference engine in order to simulate the expert's reasoning in foundation design. The selection of the foundation type (footings, piles or caissons) to be used is based on the loads, the superstructure conditions, geological and hydrogeological characteristics,

the potential problems in construction and the cost of the foundation. The type of foundation selected as well as the loads and design criteria are considered by the system in order to achieve the optimum design of abutments and piers.

Stuckrath and Grivas (1990) present a KBS to aid the selection of bridge foundations at the planning and preliminary design stages. The system has been developed using the NExpert Object rule-based expert system shell that supports both forward and backward chaining. In addition this tool permits object-orientated programming based on knowledge representation by frames. Based on user input concerning structural (load applied directly to the foundation element, admissible settlement) and geotechnical (ground type defined either by laboratory test results, if available or based on visual examination of the site, stratigraphy, ground water) specifications the system presents preliminary design options such as shallow foundations (isolated or strip footings and rafts), improved ground (through compaction or grouting) and deep foundations (piles or combinations of piles and footings or rafts). Future developments of the system envisage an extension of the knowledge base and development of interfaces with other knowledge systems and databases.

Foundation Failures

A KBS is under development for determining the causes of foundation failures (Hadipriono *et al*, 1991). The system is being developed using the expert system shell Personal Consultant Plus version 4.0, which features window oriented menus, mathematics library, external interfacing capabilities (for graphics and additional computational software). An essential part of the system is its knowledge base that consists of a frame, Foundation Failure, and several subframes, Soil Settlement, Expansive Soil, Soil Erosion, Bearing Capacity Failure, Slope Instability and Foundation Corrosion (identified as the possible causes for foundation failure). A frame or subframe groups production rules and parameters. The system queries the user about the evidence showing a possible foundation failure (like crack pattern, joint openings, wall deflection etc.) and about known soil information in order to identify the

cause of failure. A decision made to repair a damaged foundation usually follows an investigation of the causes of failure. Hadipriono and Wolfe (1991) present the application of the concept of fuzzy logic to assess the repairability of damaged foundations.

2.3.3 Earth Retaining Structures

The knowledge-based system technology has also been applied in the area of earth retaining structures. The systems developed are presented below grouped in the same manner as above.

Design

Hutchinson *et al* (1987) present RETWALL, a rule-based KBS for the selection and preliminary design of earth retaining structures. It was implemented using the rule-based expert system shell BUILD which is written in Quintus Prolog and runs on Sun Microsystems workstations. BUILD supports both forward and backward chaining and provides explanation facilities. The system first evaluates if a retaining wall is required or an embankment or cut would be satisfactory, guided by the user's input about the type of application and topographical and soil conditions. If a wall is found necessary, the system evaluates which of the nine wall types that are included in its knowledge base (brick wall, blockwork wall, crib wall, gabions, gravity wall, railway sleeper wall, reinforced earth, reinforced concrete wall, sheet piling) is applicable in that specific case. If more than one wall type is applicable, the system bases its recommendations on the first satisfactory solution encountered. The rules are ordered (allowing directed search) in a way that reflects the expert's preference of wall types. In addition to recommending a wall type (higher level selection module), the system also has the capability to perform the actual design for blockwork walls (lower level blockwork module) and to produce design drawings. Quintus PROLOG allows RETWALL to use C language files to produce graphical displays. Similar lower level modules could be developed for the remaining wall types including embankments or cuts.

Oliphant and Blockley (1989) developed a KBS that advises the user on decisions concerning the selection of earth retaining structures. The system has been written in a FRIL/PROLOG format and was developed on a Vax 11/750 machine (under the UNIX operating system) using C-Prolog. It consists of i) a knowledge base that contains rules for retaining wall selection. It is separated in three parts, the construction process, the design process and environmental impact, ii) a database that allows input to the knowledge base as subjective estimates (expressed as support pairs) of the truth or dependability of all the facts for a given wall, iii) an inference engine that assigns unique support logic values using either the multiplication or the minimum model and iv) a support logic program "shell" called FRISP, that allows complete interrogation of the knowledge base, supports backward chaining incorporating a depth-first search, provides explanation facilities and can handle uncertain and incomplete data by either of the two existing inference models. The system includes 11 case studies of retaining structures and provides a narrative of the history of each one in terms of why it was selected or considered as an alternative, allowing the user to compare these with a proposed retaining wall.

A KBS for retaining wall selection and design is presented by Arockiasamy *et al* (1991) that was developed using the M.1 shell. The shell is implemented in Prolog for use on a IBM compatible Personal Computer. The knowledge is encoded in the system using facts and rules. The system consists of two modules, the selection and the design modules. In the selection module a wall is selected based on the given set of criteria. The selection is made from a list of ten wall types including concrete gravity, cantilever, counterfort, gabions, reinforced-earth, crib, slurry, sheet-pile, tieback and soil nailed walls. The user is asked to describe the site given a list of site locations. Then he/she is queried about site geometry, wall height, project time, material and labour availability, equipment access, construction familiarity and aesthetic considerations. Based on the information provided, the most appropriate wall types are selected. The design module carries out the detailed design of the structure selected. For the cantilever wall that is presented in the paper, the system can consider sloping backfill, surcharge, three different soil layers and water table effects.

Failure Diagnosis

WADI (Chahine and Janson, 1987) is a KBS developed for the preliminary diagnosis of retaining wall failures using the rule-based expert system shell TOPSI (written in Turbo Pascal). The expert system is integrated into a database management system (DBMS) dBASE III and runs on a PC. WADI is applicable to two types of retaining walls: cantilever reinforced concrete walls and gravity concrete or rubble walls, having a maximum height of 8 metres. At the beginning of execution, input information concerning the wall under examination, the backfill soil, the bearing soil, the angle of the backfill and the failure symptoms of the wall is read from the different databases of the DBMS. After the information has been transferred, WADI classifies the bearing and backfill soils in order to determine their engineering design characteristics. Then, it performs some preliminary investigations of the failure data in order to identify the general areas of retaining wall problems that may be relevant to this failure, such as a footing problem, a drainage problem, weak bearing soil, a construction problem. The expert system proceeds to a stability analysis of the retaining wall using conventional design calculations and checking, through computation, a factor of safety against each type of failure (overturning, sliding or settlement). Final conclusions on the causes of the failure observed and recommendations on the actions that could be taken are given by the system by combining the preliminary problems generated and the different unacceptable factors of safety.

RETAIN (Adams *et al*, 1989) is a KBS that allows categorisation and organisation of knowledge relating to failure and rehabilitation of earth retaining walls. The system consists of a database implemented in DBMS INFORMIX and a series of modules which integrate OPS83 production rules, C language algorithmic functions and INFORMIX ESQL database queries. Each module completes a subtask of the solution which is fired by the user from a menu. The modules treat site identification, failure diagnosis, design synthesis and cost estimation. Upon completion of the failure diagnosis module, a table of wall failure modes with associated certainties is stored in the database. Associated with each failure mode is heuristic knowledge regarding design components that may be used for

rehabilitation. Each rehabilitation strategy is related to a set of soil and construction constraints and a preliminary design is produced for each one of them. By combining these design components a complete design is achieved.

2.3.4 Slope Stability Analysis

Knowledge-based systems developed in the area of slope stability analysis are presented below.

Wislocki and Bentley (1989) describe the development of a KBS for the determination of planning applications with respect to landslide hazard existing in South Wales. The system attempts to assess the landslide hazard that may affect proposed development sites and it produces output in the form of planning response options (which have been formulated to allow almost direct integration into the planning process operated by Local Planning Authorities in UK). The expert system has been developed using the expert system shell ESTA (Expert System Shell for Text Animation). The system contains three knowledge bases which relate to: a) sites distant from documented landslides, b) sites in close proximity to documented landslides, and c) sites on documented landslides. Planning response options (i.e. approval, approval under conditions, refusal, call for additional information, etc.) are formulated for each one of these. After an initial session of questioning, the system selects the applicable knowledge base, performs the hazard assessment by a consultation process and produces the appropriate planning responses.

XPENT (Faure *et al*, 1991) is a KBS which is being developed to assist in slope stability analysis in a high performance object-orientated environment that includes a generator of multi-expert systems (SMECI), a programming language (LeLisp), an image language for the realisation of powerful user interfaces (AIDA) and an interactive development tool for graphic interfaces (MASAI). The project is being carried out on a SONY workstation with RISK architecture. The data concerning the analysis of the problem are stored in a database through a complex but easy-to-use interface aimed at reducing recording errors to a minimum when fully developed. The system also includes a module that permits

the realisation of a two-dimensional geometrical and geotechnical model of the slope (profile) that could also be easily modified. The calculations required for slope stability analysis of this model are being performed by software for slope calculations (called 'Nixes and Trolls'), linked to the system. Simulation operations such as embankment, drainage and the consecutive evaluation of increase in stability can be carried out on the original model.

Expert Slope Design System (ESDS) presented by Denby and Kizil (1991) is a KBS to assist geotechnical engineers in the assessment of proposed slope designs in opencast coal operations in the UK. It was developed using the expert system shell Xi-Plus on a PC. The system utilises a multi level knowledge base structure with a number of sub-knowledge bases relating to the factors influencing stability which are being controlled by a main knowledge base that manages the whole system. ESDS provides explanation facilities. The system works interactively, querying the user about the geology on the site, proposed slope design and proposed working method in order to provide an estimate of the stability at a point. Although it can also work in automatic mode, the system has been re-programmed in Pascal to speed up the site assessment process that requires the assessment of a large number of points (which was slow using Xi-Plus). In automatic mode the system obtains geological information (such as strata dip and dip direction, rock mass quality, groundwater condition, etc.) from a geological model and planning information (such as slope configuration, slope curvature condition, etc.) from a design model. These models were created using two commercially available programming packages: i) AutoCAD that allows existing plans and sections to be copied into the system and ii) the language Pascal. ESDS can also be linked to NUmine, a Computer Aided Mine Design and Planning tool in order to analyse the high risk areas in much more detail by applying different slope stability analysis methods.

Gillette (1991) presents the Computerised Adviser on Soil Strength (CASS), a KBS to assist in the selection of shear strength parameters for use in stability analysis. It was written using the rule-based expert system shell Personal Consultant Plus (PC+) and runs on an AT-class PC with extended memory.

After the preliminary data entered by the user, the system starts trying the goals which are the shear strength parameters ϕ and c, a recommendation about the strength representation in the analysis, advice on soil behaviour and warnings about possible problems. The conclusions are reached using a backward-chaining inference mechanism. Checks on the consistency and validity of the input information are also performed by the system.

2.3.5 Soil Improvement

Soil improvement forms another potential area for applying knowledge-based system techniques. Two systems developed for treating this topic are presented below.

Improve (Chameau and Santamarina, 1989) is a knowledge-based decision support system designed to assist in the selection of soil improvement techniques. The knowledge of the system is represented using a structure based on "windows". Windows are mathematical representations of the restrictions to the possible values a variable of an object can take (fuzzy sets). In this way the knowledge and its uncertainty are combined in a unique entity. Each soil improvement method is represented by a stack of windows, which correspond to those physical characteristics and parameters, called dimensions, that restrict the use of the method. The searching algorithm of the system is based on the best-first search. The system consists of four parts: i) the preprocessor, that helps the user decide if there is need for soil improvement, ii) the classification system, that selects the best soil improvement technique but it continues the search for less satisfactory solutions at the user's request iii) the case-based system selects case histories that best resemble the project; it includes 50 case histories which are represented in the same way as the techniques and iv) the postprocessor, a ruled-based system which provides final information and suggestions. These modules have the same format and communicate with each other through a common storage "blackboard". Similar to the system PILE (Santamarina and Chameau, 1987) it retrieves guidelines on soil improvement techniques from the SOIL database. The system also provides explanation capabilities.

The Expert System for Preliminary Ground Improvement Selection (ESPGIS) developed by Motamed *et al* (1991) is a menu-driven system that advises users in selecting ground improvement methods or to evaluate the suitability of a user's preselected method, given the characteristics of the site. The system was developed using the expert system shell VP-Expert version 2.02 on an IBM Personal System /2 Model 50-Z with 1 Mb of RAM running under MS-DOS version 4.0. The inference of the shell can be forward, backward or mixed chaining. The shell allows for database, worksheet and external program access and has the ability to implement confidence factors, explanation of reasoning, a friendly user interface and an on-line editor. Knowledge was obtained from structured and unstructured interviews and from a literature survey and is stored in a knowledge base in the form of rules. EPSGIS allows the user to define the problem by specifying, with varying degrees of certainty, the nature of the ground improvement need, subsurface conditions and other relevant parameters. It questions the user on stratigraphy and simple index properties of the underlying soil and assigns typical values for design parameters for the soils based on the soil's description and its index properties.

2.3.6 Geosynthetics

Geosynthetics (selection and design) is another geotechnical area where the new technology has been applied. Two relevant systems are presented below.

A hybrid KBS is described by Maher and Williams (1991) that selects geosynthetic materials and performs detailed designs for different geotechnical applications. The programs included in the system were developed on an IBM-AT compatible microcomputer. The system comprises three components: i) a KBS that was developed using the shell Rulemaster2 and contains rules on how to select the most appropriate type of geosynthetic for an application, ii) a DBASE III database of geosynthetic product information, mainly concerning information on the important performance parameters of various geosynthetic products, that can be accessed using Structured Query Language and iii) geosynthetic design programs written in the C programming language. The knowledge incorporated in the system contains information about material selection for five different geosynthetic uses such as stabilisation to

reduce erosion, separation of soil layers, reinforcement to improve soil strength, drainage material to remove water and filtration to reduce cross plain flow of soil particles.

Edge Drain by Expert System (EDxES) has been developed by Dimmick *et al* (1991) to assist in the design and specification of the geotextile component of the (pavement) edge drain. The shell used to develop this system was Personal Consultant Plus from Texas Instruments. It is a backward chaining shell that allows knowledge bases to be organised into frames. All knowledge is represented by rules and facts within the frames. EDxES accepts raw site data as input, in the form of rainfall and native soil characteristics, design requirements (consisting of subbase material characteristics, pavement system and edge drain cross section information) and construction conditions. The system considers commercially available geotextiles that are non-woven and perform the dual functions of drainage and separation. The output consists of the required hydraulic and mechanical properties which are determined using typical algorithmic solutions and a list of the ten thinnest (lightest) candidate products arranged in ascending order. One limitation of the system is the underlying soils. It cannot handle soil conditions that include gap-graded, internally unstable silts.

2.3.7 Dam Seepage

Knowledge-based systems have also been developed to aid the diagnosis of dam seepage problems.

Sieh *et al* (1988) describe a KBS developed to assist in the diagnosis of seepage from embankment dams. The diagnostic expert system is part of the Operations and Maintenance Advanced Decision Support System (OMADSS), a prototype personal computer based system for dam seepage analysis, which also incorporates a database of case histories of facilities and a database of graphic images of facilities. The expert system is written in Fortran 77 and is machine portable. It uses vendor supplied run-time expert system software and was developed with the vendor development package. The knowledge included in the system is in the form of rules. The user's input required is information on the

geographic location of the seepage, the location of the seepage with respect to the reservoir water surface, the type of seepage (point source versus non-point), the time the seepage first appeared, the monitoring frequency of the seepage, the status of the seepage (increasing, decreasing or steady), the rate of seepage, and the sediment content of the water. If the system is able to reach a conclusion, the problem type is stated (point source seepage, non-point seepage, sandboils, sinkholes, drainflow), the seriousness of the problem explained (text explanation from the expert) and a recommended course of action is prescribed (text explanation from the expert).

EXSEL (Asgian *et al*, 1988) is a KBS constructed as a diagnostic tool for seepage problems associated with dams such as earth dams, rockfill dams, concrete dams and roller compacted dams. The system queries the user with multiple choice questions in order to find out the symptoms of a problem (turbid seepage or seepage carrying fines, localised seepage/wet spots/soft or quick spots, high piezometric levels, boils, change in flow rate in drains, presence of holes or depressions, whirlpool in reservoir, mass movement (slides, cracks, etc.)). It then determines the likely causes of the problem and makes recommendations for potential remedial actions. EXSEL is valuable for preliminary assessments of seepage problems because it handles only qualitative information (e.g. high piezometric levels, change in flow rates, etc.) and not quantitative information (piezometric levels, flow rates, etc.) which are necessary for final assessments. EXSEL uses the expert system shell ARITY PROLOG. The inference engine of the shell manipulates the knowledge base using the backward chaining technique. The knowledge is represented in the form of IF-THEN rules and frames. The system also gives the option to the user to consult a database of case histories of dam seepage problems. EXSEL runs on a 512K IBM XT compatible PC. If the case histories database is consulted in conjuction with the expert system then a 640K IBM compatible PC is needed. The database can be accessed through the data management computer program dBASE III.

2.3.8 Other Geotechnical Areas

In the rest of this section a number of KBS are described, each one of which is involved with a separate geotechnical task.

GroundWater eXpert (GWX) is a prolog-based KBS, presented by Davey-Wilson and May (1989) and Davey-Wilson (1991), that has been developed to advise on appropriate methods for **groundwater control** in excavations. The primary source of data for the system is the CIRIA report on groundwater control methods. In its latest version (Davey-Wilson, 1991), the knowledge base contains information on each of 27 possible methods. The choice of a method is based on 14 variable parameters from which project type, ground type, excavation size and excavation depth are the most critical. Each parameter is ranked (in the range -10 to 10) in respect to the methods, as a way of assessing its suitability. A negative value indicates the unsuitability of the method for that parameter. A weighting is also attached to every parameter, reflecting its relative importance. The system can use either preset settings or user defined values. The system is menu driven and incorporates the use of comments so that the user can have a qualitative measure of suitability in addition to the quantitative rankings. GWX has been developed for use on a standard MS-DOS micro-computer using LPA PROLOG together with its interface facilities, which enables menus and windows to be easily constructed.

A KBS was developed by Davey-Wilson (1991) for soil **shear strength** analysis using the object orientated software HyperCard, running on an Apple Macintosh computer. HyperCard enables a highly graphical interface to be easily constructed. HyperCard is a series of cards that can be filled in with pictures or texts. Each card is a separate object and up to 32000 cards can form a stack (in other words, can be part of the same application), while different stacks can easily be combined. When programming is required, an object orientated language is provided, named HyperTalk, which incorporates the use of several English words and phrases. The system developed uses soil descriptions as input in order to infer their shear strength in degrees, to a maximum accuracy of $1^{o} \pm 1^{o}$. The user is queried about the particle size distribution, the grain size, the in-situ density and homogeneity. The

input can be obtained either through a graphical interface or in the form of a menu item selection or free format description. The more detailed the answers, the higher the precision of the result. The knowledge involved is in the form of simple if-then-else rules. The same system is also used for educational purposes to simulate the execution of the laboratory shear box test with step by step interaction with the user, linking geotechnical theory to practice. The author suggests that the educational part of the program could be further developed by adding sound effects or digitised photographs or even by linking it with a video.

SOLES (Shyu and Hryciw, 1991) is a KBS to assist in the evaluation of **liquefaction potential** of soil subjected to earthquake excitations. It is a menu-driven system written in Turbo Prolog for use on a IBM-compatible PC. Forward chaining reasoning has been adopted. SOLES consists of three main components : the Control Mechanism, the Blackboard Data Structure and the Knowledge Sources. The Control Mechanism ensures that the desired control flow is followed. The Blackboard Data Structure organises the domain knowledge and the problem solving strategy. The blackboard of SOLES is a global database consisting of four sections which are the earthquake excitation, the soil properties, the analysis results and the overall evaluation, and keeps the data in an hierarchical structure. The Knowledge Sources provide information that will aid in the problem's solution and are represented by a combination of algorithmic procedures and/or set of rules. When needed they modify the data existing in the blackboard. If insufficient data is available SOLES performs the evaluation based on the limited information available and additional inferred data. At present, no facility is provided to allow uncertain data to be processed.

Juang and Lee (1989) describe Rock Mass Classification (RMC), a KBS developed for use on microcomputers for **rock mass classification**. The system is based mainly on Bieniawski's geomechanics classification scheme and is developed using the expert system shell FLOPS (Fuzzy LOgic Production System). Some of the basic features of FLOPS is approximate reasoning with fuzzy logic, the ability for either deductive or inductive reasoning, the support for both forward and backward

chaining inference mechanisms, and the use of blackboard architecture. The Rock Mass Classification system starts by reading in knowledge stored in external databases. Then a total of 182 production rules is generated from 11 user written rules. Next, the problem-specific data are entered through an external program, GETDATA (written in C), which is compatible with FLOPS and provides user interface facilities. By using the inductive reasoning (parallel processing) facility of FLOPS all rules that are fireable are then fired at once, reaching a set of preliminary conclusions, which are processed by the external program FUZZY (written in C) for establishing the final conclusions about the classification of the rock mass after completing the fuzzy computation.

A KBS (Mi and Jieliang, 1989) has been developed to predict the value of surface settlement and the degree of damage to corresponding buildings (brick structure, filled frame structure and infilled frame structure having either shallow foundation or pile foundation) caused by shield-driven **tunnelling** and to propose prevention and strengthening measures (local grouting of soil beneath the buildings or underpinning, diaphragm wall or underground continuous wall). The expert system consists of a control module, a user interface module, three subsystems (the expert inference method, the empirical formula method and the F.E.M (finite element method)) used for the estimation of surface settlement and a module that provides judgement about the building condition and proposes preventative and strengthening measures. The expert inference subsystem consists of an interface for obtaining expert knowledge, an inference engine, (based on fuzzy logic and used to compute the maximum value of surface settlement), a knowledge base and a unit for explaining expert knowledge. The knowledge base stores information about the factors influencing the prediction of settlement (class of soil, ratio of tunnel depth/diameter, stability ratio of soil, type of shield, condition of underground water, level of working quality, transportation manner on the urban ground surface). Two more maximum values are produced by the two other subsystems. The final maximum value of the surface settlement is determined by applying weighting factors to the values obtained from the three different methods.

A KBS has been developed for providing assistance for the planning of safety precautions for a **shallow trench** (less than 7.3 m deep) according to the soil conditions encountered (Siller, 1987). The system is based on two new soil classification systems developed by the National Bureau of Standards in order to increase the safety of this type of excavation. The system has been implemented using Personal Consultant, an expert system shell developed by Texas Instruments for use on PCs. Personal Consultant supports backward chaining reasoning and provides an explanation facility. The knowledge base contains factual data and production rules that represent the heuristics for manipulating the data. The knowledge base consists of two top level sections that permit repetitive consultations without exiting the system. There are three sublevels that then handle the tasks of soil classification, design parameter inference and trench bracing design.

A KBS, presented by Pearse and Rosenbaum (1986), is being developed for the evaluation of **road corridors** taking into account engineering geological criteria. The evaluation is primarily in terms of finance and safety. The system will give a cost for each potential road corridor and a probability of failure within its design life, as well as a summary of the main advantages and disadvantages of each alignment. The system allows manual interpretation and judgement for the selection of the optimum solution since factors other than the engineering geological assessment (economical, social, environmental) will contribute as well. The system is implemented in PROLOG and uses the PROLOG expert system shell APES to provide interactive, explanatory and inferential facilities. During an evaluation assessment the system will consider relevant aspects of the geology, topography, water conditions and geotechnical properties of the ground along each potential route, as well as the availability of construction materials.

2.4 Discussion

The relatively new technology of Knowledge-Based Systems has already been employed to address a wide range of geotechnical engineering problems (such as site characterisation, foundation

design, design of earth retaining structures, slope stability analysis, ground improvement, dam seepage, groundwater control, etc.) as discussed in section 2.3. To date, most of the existing systems could be described either as demonstrational prototypes, developed mainly for research purposes *(e.g. the system presented by Alim and Munro (1987) on soil investigation, FOOTER (Adams et al, 1989), FOOT (Yehia and El-Hajj, 1987), CASS (Gillette, 1991))* or as operational prototypes (representing the majority of the systems described here), intended to be eventually used in the commercial market but not having been fully developed to that stage *(e.g. CONE (Mullarkey, 1986; Mullarkey and Fenves, 1986), SITECHAR (Norkin, 1985; Rehak et al, 1985), PILE (Santamarina and Chameau, 1987), RETWALL (Hutchinson et al, 1987), WADI (Chahine and Janson, 1987), the system described by Davey-Wilson and May (1989) and Davey-Wilson (1991) concerning groundwater control, SOLES (Shyu and Hryciw, 1991)).* Only a small number of systems could be considered (according to their authors) to be near to commercial exploitation *(e.g. XPENT (Faure et al, 1991), PILEX (Elton and Brown, 1991)).* KBSs have also been developed for educational purposes *(e.g. the system described by Davey-Wilson (1991) concerning geotechnical laboratory test simulation).* Whatever the objective of the development of these systems, they all demonstrate the potential that knowledge-based system techniques have for successfully addressing geotechnical engineering problems.

Several interesting points arose from the development process of these systems which are worthy of further attention. These will be discussed briefly in the remainder of this section.

It is well recognised that the knowledge incorporated in a KBS is the most important part of the system (Feigenbaum, 1983). However, it was identified that knowledge acquisition (in other words obtaining that knowledge) is the most difficult task in the development of such a system. The majority of the systems described in this chapter require further development in order to complete the knowledge included in their knowledge bases *(e.g. the system presented by Arockiasamy et al (1991) on retaining wall selection, WADI (Chahine and Janson, 1987), the system presented by Maher and Williams (1991) on geosynthetics, SOLES (Shyu and Hryciw, 1991))* or make it more general *(e.g. LOGS (Adams et al,*

1989)). This was found to be a lengthy process since in most cases it is personal experience and expertise that is missing, which can not usually be derived from published material.

A variety of methods have been adopted for acquiring the knowledge required for the development of the KBSs described earlier in this chapter; however, it appears that no formal methodologies have yet crystallised. The most common methods employed for the acquisition of expertise include: i) literature review, including any published material such as textbooks, technical papers, codes of practice, etc. *(e.g. Geotech (Rowlinson, 1989), WADI (Chahine and Janson, 1987), the system described by Davey-Wilson and May (1989) and Davey-Wilson (1991))*, ii) structured or unstructured interviews with domain experts *(e.g. SITECHAR (Norkin, 1985; Rehak et al, 1985), the system described by Oliphant and Blockley (1989) on earth retaining structures, the system described by Sieh et al (1988) on dam seepage)*. In some systems both of the above techniques have been adopted *(e.g. SITECLASS (Wong et al, 1989), the system presented by Stuckrath and Grivas (1990) on bridge foundations, ESPGIS (Motamed et al, 1991)* or have been combined with a knowledge elicitation exercise in the form of questionnaires *(e.g. the system described by Hadipriono et al (1991) on foundation failures, RETWALL (Hutchinson et al, (1987))*. The knowledge incorporated in ESDS (Denby and Kizil, 1991) was obtained from the analysis of actual case study data.

A vital factor in the design of any KBS which needs to be considered after the knowledge required becomes available, is the selection of an adequate and appropriate knowledge representation scheme. This requires that the nature of the domain knowledge is well understood. The three methodologies most commonly used for representing the knowledge in the systems presented in section 2.3, are rule-based representation *(e.g. Dipmeter Advisor (Smith and Barker, 1983), SITECHAR (Norkin, 1985; Rehak et al, 1985), WADI (Chahine and Janson, 1987), RETAIN (Adams et al, 1989))*, logic-based representation *(e.g. the system described by Davey-Wilson and May (1989) and Davey-Wilson (1991), the system presented by Oliphant and Blockley (1989) on earth retaining structures)* and frame-based representation *(e.g. the system presented by Hadipriono et al (1991) on foundation failures)*. The

representation in the form of If-Then rules seems to be the favourite. Attempts have also been made to develop hybrid systems using two (or more) knowledge representation schemes such as rules and frames *(e.g. the system described by Smith and Oliphant (1991, 1992) on site investigation, the system presented by Halim et al (1991) on site exploration, FOUNDCON (Rashad et al, (1991), EDxES (Dimmick et al, 1991), EXSEL (Asgian et al, 1989))*. A number of systems have sufficient capacity to accommodate uncertain or incomplete information in the knowledge base *(e.g. CONE (Mullarkey, 1986; Mullarkey and Fenves, 1986), PILE (Santamarina and Chameau, 1987), RMC (Juang and Lee, 1989))*. Chameau and Santamarina (1989) reported the use of another form of knowledge representation, the window form, which combines the knowledge and its uncertainty in a unique entity. According to the authors this formalism has many useful features (such as development of composite solutions, search for lacunae (gaps in knowledge), etc.) not available in other systems. In addition to the knowledge base some systems incorporate a database of case histories allowing the user to have access to prior experiences *(e.g. the system presented by Oliphant and Blockley (1989) on earth retaining structures, Impove (Chameau and Santamarina, 1987), the system discussed by Sieh et al (1988) on dam seepage, EXSEL (Asgian et al, 1989))*.

Another critical issue concerning the development of these systems was found to be the choice of the appropriate tools for building them. It must be noted that the selection of the implementation tool is dependent upon the scheme employed to represent the knowledge. The majority of the systems presented above have been implemented using an expert system shell *(e.g. SITECLASS (Wong et al, 1989), the system described by Smith and Oliphant (1991,1992) on soil investigation, BABE (Zheng et al, 1989), ESPGIS (Motamed et al, 1991), EDxES (Dimmick et al, 1991), EXSEL (Asgian et al, 1989), RMC (Juang and Lee, 1989), the system described by Pearse and Rosenbaum (1986))*; in certain cases expert system development environments have been selected *(e.g. LOGS (Adams et al, 1989), the system discussed by Halim et al (1991) on site exploration, XPENT (Faure et al, 1991))*. For a number of systems the high level symbolic programming languages Lisp and Prolog have been selected as the implementation tool with Prolog being the most popular *(e.g. GEOTECH (Parikh and Kameswara Rao,*

1991), PILE (Santamarina and Chameau, 1987), SUPILE (Wong et al, 1991), the system described by Davey-Wilson and May (1989) and Davey-Wilson (1991) concerning groundwater control, SOLES (Shyu and Hryciw, 1991)). A limited number of systems have been developed using procedural languages *(e.g. FOOT (Yehia and El-Hajj, 1987)), the system presented by Sieh et al (1988) on dam seepage).* Finally, some systems make use of tools that allow object-orientated programming *(e.g. the system described by Stuckrath and Grivas (1990) on bridge foundation, the system presented by Davey-Wilson (1991) on geotechnical laboratory test simulation).*

It appears that expert system shells are the favourite implementation tool for building knowledge-based systems for use in geotechnical engineering. This is mainly due to the fact that by employing a shell, users can concentrate on building the knowledge base. The system can be produced quickly by someone without extensive AI programming experience but with understanding of the domain knowledge (Wong *et al*, 1989; Rosenman *et al*, 1989; Gillette, 1991). However, complex expert system shells may require a large learning curve to fully utilise their potential. Nowadays many shells are commercially available which vary from extremely simplistic to very complicated; Motamed *et al* (1991) established general criteria for the selection of an expert system shell identified during the selection process of the most suitable shell for the development of the ESPGIS. The limited preference shown towards expert system development environments can be explained by the relatively high cost of the software. A considerable number of developers still prefer the Lisp and Prolog languages as they enable fast development of prototype knowledge-based systems. It is worth noting that although to date the most popular language for ES implementation has been Lisp, interest has begun shifting recently towards the use of Prolog. Traditional languages seem not to be widely accepted as the most appropriate environment for the development of knowledge-based systems. However they have been chosen in some cases as they offer easy portability among different types of computers, compatibility with numerous pieces of software available in these languages and fast execution.

Selection of the most suitable implementation tool for the development of a KBS is important for the successful development of a commercial system. Smith and Oliphant (1991) identified that although the software tool used was found to be satisfactory for the development of the prototype system, it could prove restrictive during the development of a commercial system. Also, Denby and Kizil (1991) note that the original ESDS was developed in Prolog and although it was a success, problems of maintaining the program and extending it to link with other packages resulted in the re-implementation of the system (ESDS-X) using a suitable expert system shell. Moreover, ESDS-X has also been re-programmed in a procedural language in order to allow quicker execution of one of its operations. Such problems can also arise if large amounts of additional knowledge are required in order to transform a prototype into a practical tool accepted by industry or if additional knowledge requires different knowledge representation schemes not supported by the software selected for the implementation of the prototype. From the discussion above it is apparent that converting a prototype into a near commercial system is not always a straightforward process.

One of the most important and time consuming tasks in the development of a KBS is the creation of a suitable user interface, which will enable the system to be easily used by individuals of varying degrees of computer experience. The aim should be to develop a user interface that balances the needs of the non-expert user and the familiarity available by the experienced one so that it will not discourage the former or become cumbersome for the latter. Explanation facilities are desirable since they add to the system's credibility and enable the non-expert user to learn from the system. The prototypes described earlier in this chapter are user friendly to different degrees and provide one or more explanation facilities.

In order for a KBS to be accepted by practising engineers, it should be capable of communicating successfully with already established databases, algorithmic programs and graphical packages. Such an approach has been adopted in some of the systems presented in the previous section *(e.g. SITECHAR*

(Norkin, 1985; Rehak et al, 1985), WADI (Chahine and Janson, 1987), XPENT (Faure et al, 1991), ESDS (Denby and Kizil, 1991), the system presented by Maher and Williams, 1991)).

It is worth noting that all the prototype systems described require the inclusion of soil information in their knowledge base in different levels of detail according to the application area of the knowledge-based system. For example, systems that address the problem of site characterisation require an understanding of much more detailed soil descriptions than systems that are applied in other areas of geotechnical engineering (such as foundation design, slope stability, groundwater control, etc.).

The existing knowledge-based systems demonstrate that such systems have a major role to play in geotechnical engineering, firstly as a tool to assist experienced engineers and secondly as a means of training inexperienced engineers. Another very important contribution of this approach is the gain of knowledge by making explicit the heuristic rules that govern the decision making process of an experienced geotechnical engineer, documenting and organising this knowledge (as well as knowledge derived from published material) for a specific domain and identifying gaps in the knowledge obtained or available. As knowledge-based system technology develops and familiarity with such systems increases, the verification of these systems and the proper mechanisms for enhancing and distributing them for use by practitioners should be addressed. The development of *commercially acceptable* KBSs should be the direction to follow in the future.

CHAPTER 3

REPRESENTING THE GROUND

3.1 Introduction

Developing Knowledge-Based Systems in Geotechnical Engineering involves representing the ground. The representation of the ground poses a particular problem for knowledge-based systems since geological materials are highly variable and complex. Different levels of detail can be introduced in a representation scheme according to the system's requirements. These requirements depend on the type of information required by the engineer at different stages of a site investigation. These levels of complexity may range from broad geological classifications to detailed soil descriptions and to the determination of quantitative parameters that will allow more accurate estimates concerning the engineering behaviour of a paricular material.

In this chapter a knowledge representation scheme concerning the ground, that corresponds to the requirements of the knowledge-based system for assisting in the selection of appropriate in-situ tests, is presented. In section 3.2 the knowledge included in the system is described. The implementation of the 'Representing the Ground' application in PDC Prolog is then discussed in section 3.3.

3.2 Information Included in the Ground Knowledge Base

The Ground Knowledge Base developed in the system contains a model of the ground. The level of detail introduced in accordance with the system's requirements, is a broad classification based on the British Standards (BS 5930, 1981). The model of the ground is presented in Figure 3.1. In this hierarchy the Ground is described at the highest level as either Soil or Rock.

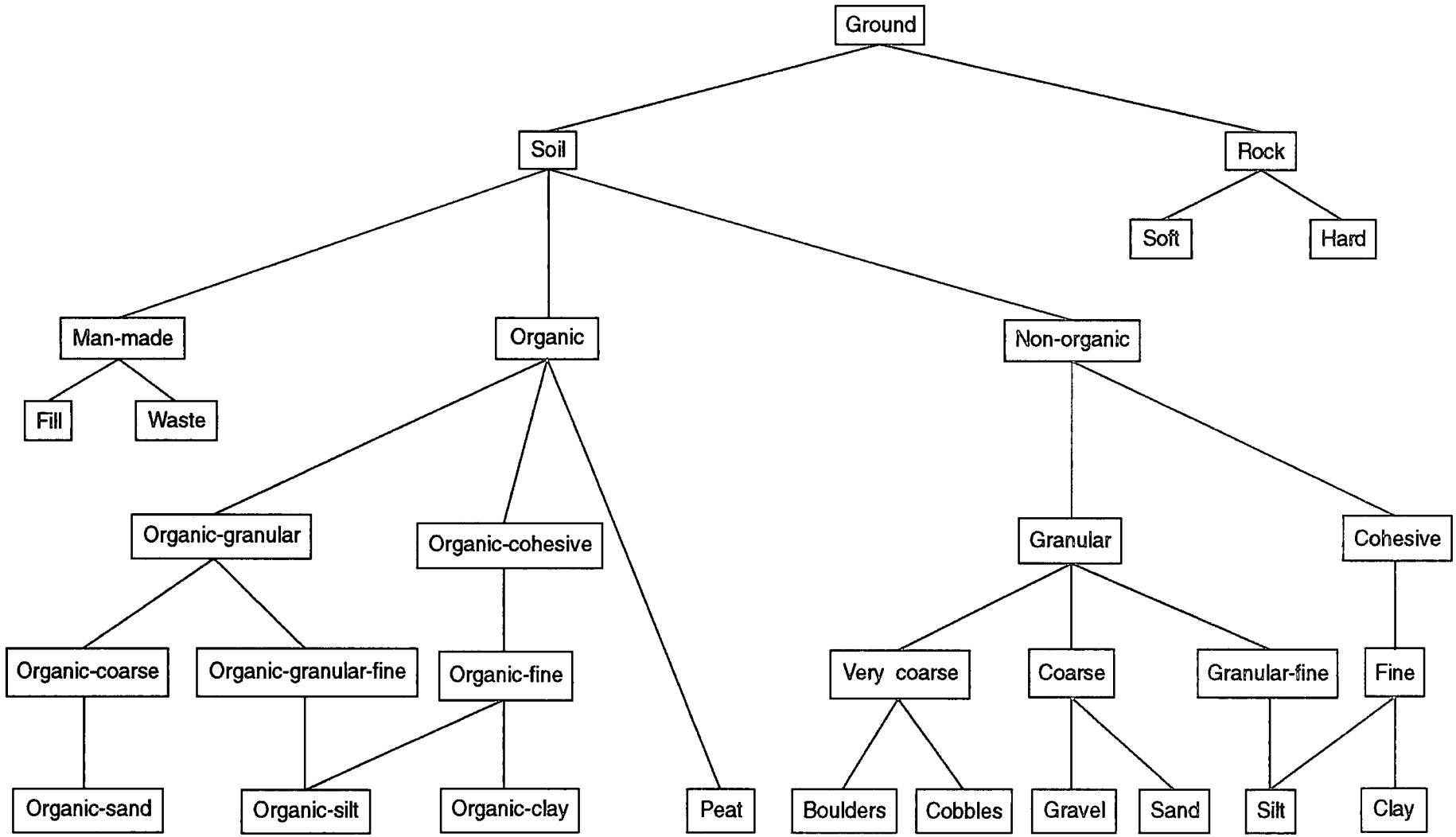


Figure 3.1 Broad classification of the ground

Since the system is mainly concerned with soils, information concerning rock has not been represented in detail. The only subdivisions currently applied to the Rock branch are given by the use of the qualitative terms Soft and Hard Rock which cover the weak (very weak, weak, moderately weak) and strong (moderately strong, strong, very strong, extremely strong) ranges expressing the uniaxial compressive strength of rocks. Consequently, Soft Rock represents rock material having a uniaxial compressive strength range of 0.6 - 12.5 MN/m^2 while the corresponding range for rock material represented by the term Hard Rock is 12.5 - 400 MN/m^2 (BS 5930, 1981). The lower limit (0.6 MN/m^2) is suggested by Spink and Norbury (1991) and the higher limit (400 MN/m^2) by Attewell and Farmer (1976). This subdivision is provided because knowledge concerning Soft and Hard Rock is included in the Tests Knowledge Base.

A Soil can be subdivided into Man-made, Non-organic or Organic. Man-made Soil can consist of Engineering Fill (compacted material) or Waste materials of various origins (non-compacted material). Following the Non-organic branch, a Soil can be identified at the most detailed level by the dominant soil type (Boulders, Cobbles, Gravel, Sand, Silt and Clay). The Organic branch culminates in the organic dominant soil types (Organic Sand, Organic Silt, Organic Clay and Peat).

From Figure 3.1 it can be seen that Silt may be either a Fine Granular Soil or a Fine Cohesive Soil depending on its behaviour. Silt is considered as a granular material if it does not display any plastic properties and as a cohesive material if it does. The same definitions apply to Organic Silt.

This broad classification is based on knowledge about grading and plasticity. For example, Granular soils have a grain size range of 0.002 - 2000 mm and are non plastic. Cohesive soils have a grain size range of 0 - 0.06 mm and a liquid limit range (indicating plasticity) of 0 - 200 %. For the reason mentioned above, the grain size ranges of Granular and Cohesive soils overlap. An upper limit of 200 % has been taken to give an indication of a maximum likely liquid limit. The grain size ranges become more specific upon descending the hierarchy. For instance, the grain size range for Coarse

soils, a subdivision of Granular soils, becomes 60 - 0.06 mm and for Sand, which is a subdivision of Coarse soils, it becomes 2 - 0.06 mm.

The grain size and liquid limit ranges corresponding to the non-organic dominant soil types are given in Table 3.1 (BS 5930, 1981). Organic Sand, Organic Silt and Organic Clay can be classified in the same way as Sand, Silt and Clay if the organic material is removed or ignored. Peat is an organic material with variable grain size and consequently there is no specified grain size range for it in this representation scheme.

Dominant Soil Type	Grain Size (mm)	Liquid Limit (%)
Boulders	2000 - 200	--
Cobbles	200 - 60	--
Gravel	60 - 2	--
Sand	2 - 0.06	--
Silt	0.06 - 0.002	0 - 200
Clay	0.002 - 0	0 - 200

Table 3.1. Grain size and liquid limit ranges corresponding to non-organic dominant soil types

In a more detailed representation the grain size ranges corresponding to Gravel, Sand and Silt can be subdivided into the ranges shown in Table 3.2 (BS 5930, 1981). Silts and Clays can have a more refined classification in relation to the liquid limit, which is presented in Table 3.3 (BS 5930, 1981).

As part of the more detailed representation, additional information relating to the dominant soil types is included such as permeability, consistency and compressibility. The permeability ranges corresponding to dominant soil types given by Fookes and Vaughan (1986) are shown in Table 3.4. The consistency ranges corresponding to Sand and Gravel are expressed in terms of Standard Penetration Test N-values and are presented in Table 3.5 (BS 5930, 1981). The consistency of Clay is usually expressed in terms of undrained shear strength as shown in Table 3.6 (BS 5930, 1981). Silts can be described by the

consistency ranges used for granular soils if the sand proportion dominates, and by consistency ranges used for cohesive soils, if the clay proportion is high (Weltman and Head, 1983). Therefore, Silt is included in both Table 3.5 and Table 3.6. The compressibility ranges of different dominant soil types are given in Table 3.7 (Weltman and Head, 1983).

Dominant Soil Type	Grain Size (mm)	Modifier
Gravel	60 - 20	Coarse
	20 - 6	Medium
	6 - 2	Fine
Sand	2 - 0.6	Coarse
	0.6 - 0.2	Medium
	0.2 - 0.06	Fine
Silt	0.06 - 0.02	Coarse
	0.02 - 0.006	Medium
	0.006 - 0.002	Fine

Table 3.2. Subdivisions of grain size ranges

Dominant Soil Type	Liquid Limit (%)	Modifier
Silt Clay	0 - 35	Low plasticity
	35 - 50	Intermediate plasticity
	50 - 70	High plasticity
	70 - 90	Very high Plasticity
	90 - 200	Extremely high plasticity

Table 3.3. Subdivisions of liquid limit ranges for fine soils

Dominant Soil Type	Coefficient of Permeability (m/s)	Modifier
Boulders Cobbles Gravel	$1 - 10^{-3}$	High Permeability
Sand	$10^{-3} - 10^{-5}$	Medium Permeability
	$10^{-5} - 10^{-7}$	Low Permeability
Silt	$10^{-5} - 10^{-7}$	Low Permeability
	$10^{-7} - 10^{-9}$	Very Low Permeability
Clay	$10^{-9} - 0$	Practically Impervious

Table 3.4 Dominant soil types and permeability ranges

Dominant Soil Type	N-value	Modifier
Gravel Sand Silt	0 - 4 4 - 10 10 - 30 30 - 50 50 - 100	Very Loose Loose Medium Dense Dense Very Dense

Table 3.5 N-value ranges for granular soils

Dominant Soil Type	Undrained Shear Strength Cu (kN/m^2)	Modifier
Silt Clay	0 - 20 20 - 40 40 - 75 75 - 150 150 - 300	Very Soft Soft Firm Stiff Very Stiff

Table 3.6. Undrained shear strength ranges for cohesive soils

Dominant Soil Type	Coefficient of volume compressibility m_v (m^2/MN)	Modifier
Sand Silt	0 - 0.05	Very Low Compressibility
Non-organic Clays	0 - 0.05 0.05 - 0.1 0.1 - 0.3 0.3 - 1.5	Very Low Compressibility Low Compressibility Medium Compressibility High Compressibility
Organic Clays Peat	1.5 - 20	Very High Compressibility

Table 3.7 Coefficient of volume compressibility ranges corresponding to different dominant soil types

It should be noted that most soils in reality are a mixture of different sizes of materials (different soil types). These can be identified by visual examination of the soil sample and/or by means of testing, and are contained in the detailed engineering description of the soil. One (or more than one) of the different components of the soil will be the dominant soil type(s) whose name is usually given in capitals in the soil description. An example of such a description is:

slighty clayey, silty, very sandy GRAVEL

In this example the soil described consists of four different soil types: clay, silt, sand and GRAVEL which is the dominant soil type. In the representation scheme put forward by Toll *et al* (1991a) each soil type participating in the composition of the soil is associated with an Amount. If the soil type is the dominant soil type the Amount is given as *Main* (GRAVEL). For the descriptive term 'very' the Amount is given as *Major* (sand). For the soil's name followed by the ending -y the Amount is given as *Secondary* (silt). For the descriptive term 'slightly' the amount is given as *Minor* (clay). It is worth noting that in a composite soil the Main soil type can indicate either the major constituent of the soil mass or the soil's behavioural type.

A full engineering soil description can also contain additional information concerning the structure, moisture and consistency of the soil mass as well as characteristics of each soil constituent such as colour, shape, grading etc. (Toll *et al*, 1991a).

Although full engineering soil descriptions play an important role in the representation of soils, it is normally the soil's behavioural type (the Main soil type) or broader classifications which are important when decisions are being made about the most appropriate field test to be used. This will particularly be the case at the early stages of a project when over-all feasibility is being considered and a detailed investigation of the soils has not been carried out. For this reason, the model of the ground represented here includes only the Main or dominant soil type, not the lesser constituents.

It is often the case however that the applicability of a certain in-situ technique is influenced by the knowledge of the lesser consituents of a composite soil. The level of detail that such decisions usually require is a composite soil consisting of the Main soil type and one lesser component that may be indicative of the soil's behaviour (e.g *silty Gravel*). In this work, percentage ranges are defined for Secondary soil types in accordance with the BS 5930 (1981) which refers to them for both coarse and fine Main soil types and these are presented in Table 3.8.

Dominant Soil Type	Percentage Ranges of Secondary Soil Types (%)	Modifier
Gravel	5 - 20	Gravelly
Sand	5 - 20	Sandy
	5 - 15	Silty
	5 - 15	Clayey
Silt	35 - 65	Gravelly
Clay	35 - 65	Sandy

Table 3.8 Percentage ranges of secondary soil types

It should be noted however that several inconsistencies are present within BS 5930 (mainly concerning fine soils) which have been identified and discussed by Child (1986) and Vaptismas (1992). Norbury *et al* (1986) have proposed a more consistent scheme for soil descriptions following the basic principles set out in BS 5930.

The information included in the detailed representation of the ground relates to future development of the system. Knowledge about grading, plasticity, compressibility, consistency and permeability of the Main soil type as well as knowledge concerning the Secondary constituent of a composite soil in addition to the Main soil type could be very useful. In some cases the applicability (or the limitations) of a certain in-situ test method do depend on that level of detail. For instance, the Vane Test, although having high applicability in clays, would have only medium applicability in *stiff* clays (Orchant *et al*, 1988) i.e. the consistency modifier does have an effect. In the system developed as part of this research project, facilities have been provided for including and utilising this greater level of detail.

3.3 Implementation of Ground Information in Prolog

3.3.1 Introduction

In this section a brief description of the main characteristics of the Prolog programming language is presented and the Prolog 'dialect' that was chosen for the implementation of the system is discussed. The actual implementation of the 'Representing the Ground' application in PDC Prolog is

then discussed in detail. The development of the Ground knowledge base as a set of facts is presented in section 3.3.3 whereas the development of an extended inference mechanism for accessing the knowledge base is presented in section 3.3.4.

3.3.2 Prolog programming language

PROLOG (PROgramming in LOGic) is a symbolic programming language. The declarative character of Prolog allows the programmer to concentrate on the description of the objects occuring in a problem and the relationships between them rather than on the prescription of the sequence of steps taken by a computer to solve the problem, as happens in procedural languages.

Prolog's syntax is based on *first order predicate logic* formulas written in clause form and further restricted to *Horn clauses*. Deductive inferences in Prolog are based on the *resolution principle* for mechanical theorem proving introduced by J. Alan Robinson.

Prolog's built-in inference mechanism supports *backward chaining reasoning* and brute force *depth first* search. The inference mechanism includes a *pattern matcher*, which corresponds (approximately) to what is called *unification* in the formal definition of resolution (Clocksin and Mellish, 1981). Prolog uses a *backtracking* mechanism to find all possible solutions to a given problem. This mechanism allows *non-deterministic* programming in Prolog.

Prolog is not a purely logic programming language as it provides certain tools such as the *fail* predicate and the *cut* that allow some control on the inference mechanism of a Prolog program but reduce its clarity. It also provides tools to cover practical needs such as input or output. Consequently, Prolog combines both declarative and procedural approaches providing a practical programming system.

The orientation towards the Prolog programming language as the implementation tool for the development of the knowledge-based system presented in this thesis, was mainly due to some features

of the language that seem to be very suitable for the requirements of the system. These features are outlined by Bratko (1990), who stated that "Prolog is especially well suited for problems that involve objects - in particular, structured objects - and relations between them" and Marcellus (1989) who stated that "Prolog shines in two major areas: search and pattern matching. These capabilities are features of the language".

Prolog was first implemented by Alain Colmerauer at Marseilles in 1972 with Robert Kowalski at Edinburgh contributing crucial theoretical work. The first efficient implementation was due to David Warren at Edinburgh in 1977. Many Prolog systems are now commercially available, running on a variety of computers.

The 'dialect' of Prolog chosen for the implementation was Turbo Prolog (1986) via Borland, which is an IBM PC-based Prolog compiler of relatively low cost that provides an easy to use environment similar to that of Turbo PASCAL and Turbo C. It should be noted that a Personal Computer was the only available hardware at that time.

Turbo Prolog differs from DEC-10 Prolog or Edinburgh Prolog (as described by Clocksin and Mellish, 1981) in several ways. Branbury and Woodward (1988) give a detailed list of the similarities, and differences between the two dialects. The basic difference between them is that Turbo Prolog requires type declarations for the arguments to all predicates, making Turbo Prolog a fast language and helping it to detect programming errors. On the other hand, some advanced programming techniques (that exist in Edinburgh Prolog) are not possible because of the limited nature of Turbo Prolog's type declarations.

In 1990 Borland International, distributor of Turbo Prolog, took a business decision to no longer distribute and support the language. Since that date PDC Prolog has been available and supported via the Prolog Development Center in Denmark. For this reason the implementation of the system achieved

up till then, has been transferred to PDC Prolog, which is fully supported and available under many different operating systems.

Initial problems encountered of accessing sufficient memory were overcome using the Phar Lap Dos-Extended version of PDC Prolog 3.30 (User's Guide,1992; Reference Guide, 1992) on a 286 Research Machines Nimbus AX/2 Personal Computer with 3 Mbytes internal memory.

3.3.3 Facts: Ground Knowledge Base

The development of the Ground Knowledge Base was a progressive process. Initially the knowledge base consisted of a set of Prolog clauses, called facts, used to define the classes of the ground model (shown in Figure 3.1) by their members and properties. These facts were described by the predicate *class*. The definition of the predicate class has been altered through the development process in order to satisfy the system's knowledge representation requirements. The progressive development of the predicate *class* is discussed in detail later in this section. In the final stages of the implementation a predicate, called *modifier*, was introduced to accomodate the more detailed representation of the ground information described in section 3.2. A full listing of the final implementation is given in Appendix A.

As was mentioned in section 3.2, the knowledge representation of the ground was based on the broad classification shown in Figure 3.1. The model of the ground could be considered as a tree-like structure. This structure could be described as a general tree that accepts the possibility of a node having more than one parent. The tree is implicit. It does not exist anywhere except in the logical relations between the classes.

The relations between the classes of the hierarchy were described by the predicate *class*. The predicate *class* has three arguments. The first corresponds to the name of the class in the structure, the second specifies its members and the third describes its properties. For example, the class: *Ground*, which is

the root of the tree has the members: *Soil* and *Rock*, and no properties. The member *Soil* forms another class whose members are: *Non-organic*, *Organic*, *Man-made* and has the property: *ground type* that takes the value *Soil*. All the classes of the structure are described in a similar way until the most detailed level, the leaves (or instances) of the tree are reached. This level is formed by the dominant soil types (*Boulders*, *Cobbles*, *Gravel*, *Sand*, *Silt*, *Clay*, *Organic Sand*, *Organic Silt*, *Organic Clay*, *Peat*). Instances are represented in the same way as classes i.e. by the predicate *class*, the only difference being that they have no members.

The initial form of the Prolog clauses for defining objects and in particular the root (top level class), a node (subclass) and a leaf (instance) of the tree-like structure, as described in the previous paragraph, is illustrated below:

a) The root: ground

```
class ( ground, [ soil, rock ], [ ] ).
```

b) A node: coarse

```
class ( coarse, [ gravel, sand ], [ coarseness, coarse, min_grain_size, "0.06",
                                    max_grain_size, "60" ] ).
```

c) A leaf: sand

```
class ( sand, [ ], [ soil name, sand, min_grain_size, "0.06",
                     max_grain_size, "2" ] ).
```

It can be observed from the above examples that the root of the hierarchy is described by the predicate *class* having the third argument that represents its properties as an empty list, []. A leaf of the hierarchy is also described by the predicate *class* having the second argument that represents its members as an empty list, [].

As can be seen from the Prolog code presented above, the properties of each object were initially identified using a straightforward pair of (attribute, value), e.g.:

Attribute: coarseness, Value: coarse

Attribute: min_grain_size, Value: "0.06"

It was found to be necessary to have two types of attribute: (i) Attributes which are identified once and are carried down the structure to the current level, expressing general knowledge. These may also allow identification of the exact position within the structure. (ii) Attributes which appear at several levels within the structure and whose values change, becoming more specific upon descending the structure. These are illustrated in Figure 3.2 which shows a path through the structure, from *Ground* to *Sand.*

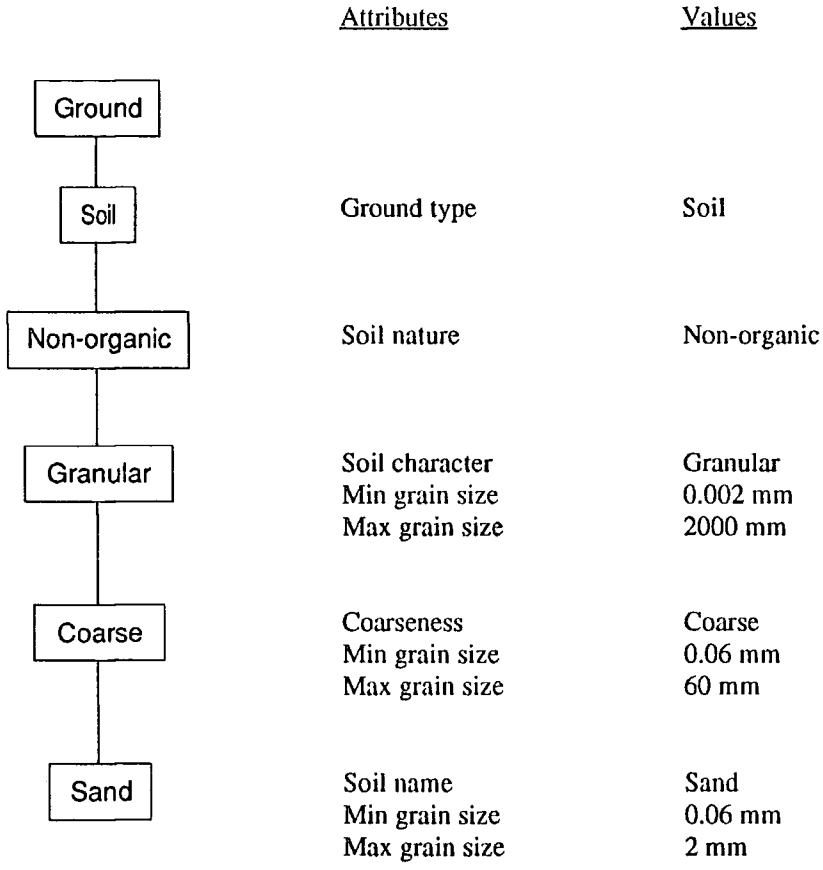


Figure 3.2 - A path through the soil classification hierarchy

The first type of attributes, *ground type*, *soil nature*, *soil character* etc. are identified at one level and are passed down the hierarchy to apply to lower level objects by inheritance. In this example, all attributes that are of the first type also allow identification of the exact position within the structure. Hence, at the level of *Coarse* in the structure, it was known to be *soil*, *non-organic* and *granular* by inheritance from above. The second type, *min grain size* and *max grain size* are redefined at a number of levels. It is worth noting that it is possible to create an attribute at an instance level i.e. the attribute *soil name* for the instance *Sand*.

This simple format of (attribute, value) pairs used initially for the representation of properties hides the restriction that an attribute such as *grain size* had to be expressed as two attributes: *min grain size* and *max grain size*, since it could take a range of values, defined by a minimum and maximum numerical value. It was also found that the straightforward (attribute, value) pair could not easily handle a more detailed representation of the soil. To overcome these problems a more complicated format was introduced:

```
Attribute,    [ (Value1),    (Factor1)
                (Value2),    (Factor2)
                  "              "
                  "              "     ]
```

In this format the attribute does not need to take a unique value but can take a number of different values depending on an external factor (or factors).

Consequently, the definition of the predicate *class* was modified in order to incorporate the new format for representing the properties of an object. The Prolog clauses that define classes (objects) using the complicated format for the representation of their properties are presented below:

a) A node: coarse

```
class ( coarse, [ sand, gravel ],
        [att ( coarseness,
                [ val ( [coarse  ],          fact ( [ ] )        )]),
         att ( grain_size,
                [ val ( [ "0.06", "60" ],    fact ( [ ] )        )])]).
```

b) A leaf: sand

```
class ( sand, [ ],
        [att ( soil_name,
                [ val ( [ sand ],            fact ( [ ] )        )]),
         att ( grain_size,
                [ val ( [ "0.06", "2"  ],    fact ( [ ] )        ),
                  val ( [ "0.06", "0.2"],    fact ( [ fine ] )   ),
                  val ( [ "0.2",  "0.6"],    fact ( [ medium ] ) ),
                  val ( [ "0.6",  "2"  ],    fact ( [ coarse ] ) )])]).
```

The first argument of the predicate *class*, that declares the name of the object, is of type 'symbol'. The second argument that represents its members belongs to the list domain indicating a list of symbols. As was previously stated, an empty list, [], indicates that the object is at the base of the structure since it has no members (see example for *Sand* above). The third argument, that gives the properties of the class, introduces a list of multi-level compound objects. The names *att*, *val* and *fact* (which are usually called functors) indicate the use of compound data objects in PDC Prolog and have been defined to identify attribute, value and factor.

The functors are followed by a number of arguments in parenthesis which represent the objects belonging to them. It can be observed from the examples above that the functor *att* has two arguments. The first argument represents the attribute name and is of type 'symbol'. The second argument is a list of multi-level compound objects identified by the functor *val*. The functor *val* has two arguments as well. The first argument of the functor *val* consists of the attribute values represented as a list of symbols. In the case of an attribute having numerical values (e.g. *grain size*) the numbers are entered as

strings (bound within double quotes) since PDC Prolog performs an automatic type conversion between the string domain and the symbol domain. The second argument of the functor *val* is a compound object represented by the functor *fact*. The functor *fact* has one argument which is defined as a list of symbols and represents the factors (or modifiers) that correspond to specified attribute values. An empty list, [], indicates that the attribute values are not dependent upon a factor, as happens when general ranges of values are given (see examples for *Coarse* and *Sand* above).

The structure of one element in the list of the multi-level compound objects described above is illustrated in Figure 3.3. Although this format may look complex it has proved to be efficient in representing the information structure required. In addition, it clarified the program and facilitated the processing of the data.

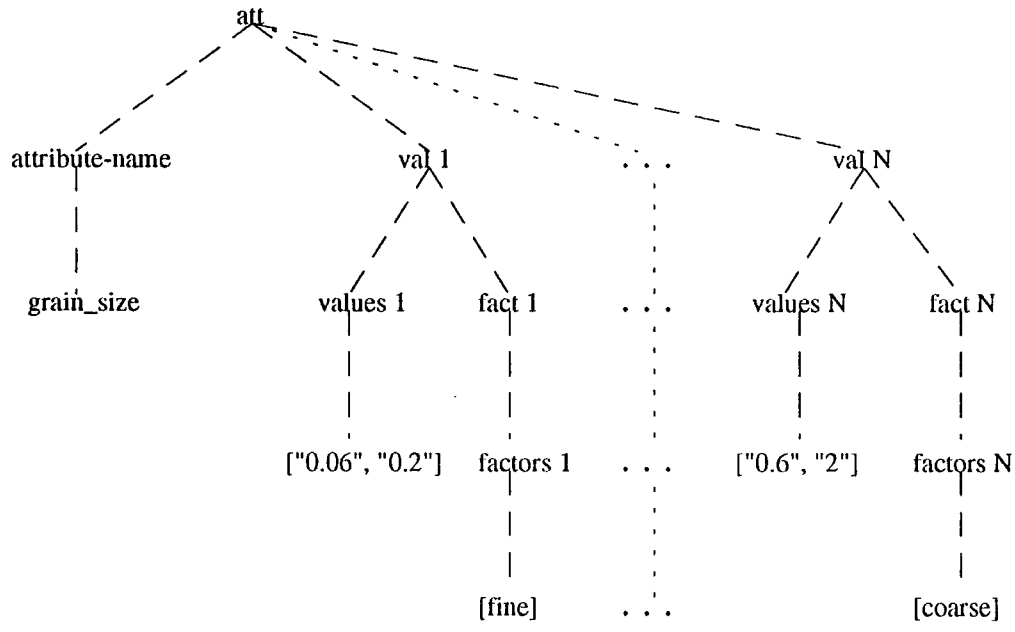


Figure 3.3 Structure of the multi-level compound object

In order to maintain the set of facts that describe the ground model (predicate *class*) at a more general level in relation to the properties attached to each soil type (to the instances of the structure), another level of detail was introduced in this knowledge representation scheme by defining the predicate *modifier*. The predicate *modifier* now deals with information representing more refined classifications such as *coarse* grained, *medium* grained, *fine* grained *Sand* (which used to be included at the predicate *class* level), as well as additional information which is considered to be more specific such as consistency, compressibility, permeability and percentage ranges for Secondary soil types. This predicate has two arguments, the first corresponds to the name of the soil type concerned and is of 'symbol' type. The second corresponds to properties of this soil type and is a list of multi-level compound objects having exactly the same structure as the third argument of the predicate *class*.

An example is given below defining an instance of the hierarchy (e.g. *Sand*) after the introduction of the predicate *modifier*.

The code in PDC Prolog for the predicate *class* defining the instance *Sand* becomes:

```
class ( sand, [ ],
        [att ( soil_type,
                [ val ( [ sand ],          fact ( [ ] )      )]),
         att ( grain_size,
                [ val ( [ "0.06", "2"  ],  fact ( [ ] )      )])]).
```

The more detailed representation of the instance *Sand*, as given by the predicate *modifier*, is presented below:

```
modifier ( sand,
        [att ( grain_size,
                [ val ( [ "0.6", "2"],        fact ( [ coarse ] )          ),
                  val ( [ "0.2",  "0.6"],     fact ( [ medium ] )          ),
                  val ( [ "0.06",  "0.2"  ],  fact ( [ fine ] )            )]),
        att ( "N_value",
                [ val ( [ "0",  "4"   ],      fact ( [ very_loose ] )      ),
                  val ( [ "4",  "10"  ],      fact ( [ loose ] )           ),
                  val ( [ "10", "30"  ],      fact ( [ medium_dense ] )    ),
                  val ( [ "30", "50"  ],      fact ( [ dense ] )           ),
                  val ( [ "50", "100"],       fact ( [ very_dense] )       )]),
        att ( coefficient_of_permeability,
                [ val ( [ "10e-5", "10e-3" ], fact ( [ medium_permeability ] ) ),
                  val( [ "10e-7", "10e-5" ],  fact ( [ low_permeability ] ) )]),
        att ( coefficient_of_volume_compressibility,
                [ val ( [ "0", "0.05" ],      fact ( [ very_low_compressibility ] ) )]),
        att ( secondary_percent,
                [ val ( [ "5", "20" ],        fact ( [ gravelly ] )        ),
                  val ( [ "5", "15" ],        fact ( [ silty ] )           ),
                  val ( [ "5", "15" ],        fact ( [ clayey] )           )]) ]).
```

Looking at the example above, it could be argued that the functor *fact* in the predicate *class* is no longer necessary, thus the nested structure to which it belongs could be redefined without it. Although this is the case for all the facts described by the predicate *class* in the existing system, the compound structure was kept the same for two reasons: (i) it provides a uniform representation scheme with the predicate *modifier*, something that significantly facilitates the programming and (ii) it gives more flexibility to the system for other applications or for modifications (alterations and/or additions) to the existing one.

It has been found possible to add or to delete information from the system without changing the over-all structure, a feature which enhances the functionality of the system.

In the latest version of the system the facts no longer represent a static collection of information but constitute a dynamic database, called knowledge_base; a dynamic or internal database in PDC Prolog, that is composed of facts that can be updated (added, removed or changed) at run time. The predicates *class* and *modifier* that describe the facts are declared as predicates of the database but are accessed in exactly the same way as normal predicates which are declared in the predicates section. This alteration was suggested by PDC Prolog (Appendix B), due to a bug in Prolog that was detected during the development of the system. It was also stated by PDC Prolog that is normally more efficient to declare static facts as internal database predicates (see Appendix B).

Storing the facts in an internal database will allow the provision of facilities to the user for updating the facts representing the knowledge in future development of the system.

3.3.4. Rules: Extended Inference Mechanism

The application described above requires a search-based approach. In order to retrieve information about the classes and instances of the hierarchy a search needs to be carried out at many levels within the two sets of facts (predicate *class*, predicate *modifier*) that describe them. Therefore several rules were introduced to enable or facilitate the searching process. A full listing of these rules is provided in Appendix A. These search rules can be divided into three categories according to the inferences they allow:

- Inheritance rule (*get_all_attributes*)
- Transitivity rules (*discover_members*,

  find_ancestors)
- Information retrieval rules (*find_attribute_and_value*,

  find_modifiers,

  find_objects_and_modifiers)

It should be noted that a number of rules have been written, concerning list processing, which are used by 'higher level' rules such as the ones listed above. These generic rules are discussed in section 5.2.

Inheritance Rule

Inheritance is very useful as it can extrapolate an explicit set of facts to a much larger implicit set of facts, by inferences with rules. Therefore it becomes unnecessary to store information that can be inherited. This is provided by the 'get_all_attributes' rule which acts as follows:

get_all_attributes

Input : Object-name (e.g. sand)
Search-origin point (e.g. ground)

Output: A list of the properties of the object (attributes-values-factors) defined by the predicate *class* and the properties of the ancestors of the object
A list of the properties of the object (attributes-values-factors) defined by the predicate *modifier* (if any)

It is apparent from the output that the rule searches for solutions in both sets of facts (*class*, *modifier*).

The rule makes use of the implied tree-like structure in order to allow the object to inherit its ancestors' properties. The tree-like structure is implied by the logical connections between the facts represented by the predicate *class*. The logical relation between these facts is that all objects (except the top-level class), described by them also appear as members of another object (second argument of the predicate *class*).

Attribute inheritance can be divided into: i) attribute-name inheritance and ii) attribute-value inheritance. If an attribute is defined only once in the hierarchy then both the attribute-name and the attribute-value are inherited by all the subclasses and instances of the object for which it is defined. If an attribute is defined at several levels within the structure then the current level inherits the

attribute-name from the level for which the attribute was initially defined, but the value specified at the original definition is overwritten at every lower level that this attribute is re-defined. Attribute inheritance is performed in this way for both classes and instances.

Instances not only inherit attributes from higher level classes but can have attributes defined at their level within the tree-like structure, as well as in the more detailed level of the predicate *modifier* (which is independent of the structure). Hence, the first list of properties for an instance includes the inherited attributes and the attributes defined at the instance's level. The second list contains the attributes that are incorporated within the predicate *modifier* facts.

Transitivity Rules

Transitivity explains relationships between things further apart from relationships between things closer together. Like inheritance, transitivity is very useful because it saves fact space by storing facts relating only the "closest" things. Transitivity inferences are provided in the system by the rules 'discover_members' and 'find_ancestors' which act as follows:

discover_members

Input: Object-name (e.g. coarse)

Output: A list of all possible members of the object (the members generated are instances, e.g. gravel, sand)

The rule searches for solutions in the first set of facts, defined by the predicate *class*. This rule also makes use of the second argument of the predicate *class* that declares the direct members (children-one level below in the hierarchy) of an object in order to search down the hierarchy and to generate all the instance-members of the input object.

As an example, the code of this rule is given below. Initially a predicate called *discover_member* is defined which identifies all possible members of the object by backtracking.

```
discover_member(Name, Name):-
        class(Name, [], _), ! .
discover_member(Category, Name):-
        class(Category, List, _), ! ,
        members(Member, List),
        discover_member(Member, Name).
```

The second clause for the predicate *discover_member* states that a class *Name* is member of a class *Category* if it is a member of one of the members of the class *Category*. This recursive process is continued until one member of the class *Category* is found to be an instance (this is declared at the first clause). After finding the first solution Prolog backtracks to the last subgoal that offers alternatives (in the above example the predicate *members*) in order to obtain all possible solutions.

A higher level predicate called *discover_members* is then defined in order to collect all the possible solutions generated from backtracking into one list using the built-in predicate of PDC Prolog, *findall*. The user defined generic predicate *remove_duplicates* is used to discard duplicate solutions that may be present. These can arise in cases where a node has more than one parent (e.g. silt). The Prolog code for this predicate is shown below:

```
discover_members(Category, Names):-
        findall(Name, discover_member(Category, Name), Namelist),
        remove_duplicates (Namelist, [], Names).
```

find_ancestors

Input: Object-name (e.g. sand)

Output: A list of all the ancestors of the object (e.g. coarse, non-organic, soil, ground)

This rule also searches for solutions in the first set of facts, defined by the predicate *class* and makes use of the second argument of the predicate *class*. A class is identified as ancestor of a chosen object if the object is one of the direct members of the class. All the ancestors of the object are identified using recursion.

In the system developed in the course of this research the information generated by the 'find_ancestors' rule is provided by the 'get_all_attributes' rule. This is due to the fact that in the scheme for representing the knowledge adopted here each class denotes an attribute that has as its value the name of the class. However, in order to allow for more general representation schemes where this principle may not be appropriate, the rule 'find_ancestors' has also been defined that provides the facility to obtain such information.

Information Retrieval Rules

These rules analyse specific facts at different levels and retrieve information stored in them, but without being dependent upon the relationship between given facts in the model.

find_attribute_and_value

Input: Object-name (e.g. sand)
Factor (e.g.coarse)
Output: Attribute and value(s) that correspond to the factor (e.g. grain_size of 0.6, 2 mm)

The rule searches for solution in the second set of facts (defined by the predicate modifier) because only these facts have factors which are not described by an empty list, [].

find_modifiers

Input values:	Object-name (e.g. sand)
	Attribute-name (e.g. grain_size)
	Attribute-value(s) (e.g. ["0.8", "2"])
Output values:	The corresponding factor(s) (e.g. coarse)

This rule also searches for solutions in the second set of facts (defined by the predicate *modifier*). As the values of an attribute in the model are either symbolic values or a numerical min-max pair (or max-min pair) the rule is able to perform either a simple search, matching up symbolic values, or a comparative information retrieval, checking if the entered values (which can be either one value or a range of values not in a specific order) lie within the predefined ranges specified in the facts. It is also possible to enter values that cover more than one predefined range (e.g. input values: ["0.1", "2"]). In this case the rule will combine in an incremental way the factors that correspond to the predefined ranges that each of the two entered values lie within, producing, for example, the following output value: 'fine _to_coarse'.

find_objects_and_modifiers

Input values:	Attribute-name (e.g. grain_size)
	Attribute-value(s) (e.g. ["1"])
Output values:	The corresponding object(s) and factor(s) (e.g. coarse sand)

This rule searches for solutions in the second set of facts (defined by the predicate *modifier*) in which there are factors specified. However, if no success have been achieved it then acquires solutions in the first set of facts (defined by the predicate *class*). In a similar way to the 'find_modifiers' rule, this is also able to perform a simple or a comparative search in the same way as described above.

This rule is triggered by a higher level rule defined by the predicate *find_all_names_factors*, that collects all possible solutions in lists using the *findall* predicate. The latter rule also allows the identification of solutions in the case where a numeric range of input values does not correspond to one

object, by treating the minimum and maximum values of the input range as distinct values and finding the object(s) and the factor(s) (if any) that correspond to each of these objects. For example, if the selected attribute is 'grain_size' and the input values are ["1","0"], the rule **find_all_names_factors** identifies the fact that no single object corresponds to this range of values and consequently returns for the minimum value (0), 'clay' or 'organic_clay' with 'no modifiers' whereas for the maximum value (1) it returns 'sand' with modifier 'coarse' or 'organic_sand' with modifier 'organic_coarse'.

In certain cases, higher level predicates have been defined in order to direct the output which is produced by the rules. For example, the rule **find_ancestors** is called by a higher level predicate, named **find_all_ancestors**, that collects all solutions generated by **find_ancestors** (e.g. for *Silt* two solutions are found as it is the child of two parents) in a list and controls the way in which these are displayed to the user. It should also be noted that in addition to the rule **get_all_attributes** a rule called **find _vallist** has been written that returns the value(s) (specified at both predicate *class* and predicate *modifier* levels) of a specific attribute of an object. This rule makes use of the **get_all_attributes** rule and is triggered by the higher level predicate **find_vallists** which also directs its output.

The rules described above are structure dependent but not domain dependent. This means that they could be used to search hierarchies that are described by facts having the same structure as those described in section 3.3.3 without making any changes in relation to the knowledge being represented. For this reason it could be considered that they provide an Extended Inference Mechanism on top of the built-in inference engine of PDC Prolog. This is discussed in greater detail below.

Specifically the rules **get_all_attributes, discover_members** and **find_ancestors** are very general. They could be applied in any other hierarchy describing a totally different knowledge domain. The rules **get_all_attributes, discover_members** and **find_ancestors** provide facilities for inheritance and transitivity for the system that could be used if required, as happens in the case of the **get_all_attributes** rule that is utilised by the rule **find vallist** (which is also domain independent).

The rules **find_attribute_and_value, find_modifiers** and **find_objects_and_modifiers** have an implicit domain dependency because they either search for solutions only at the second sets of facts (described by the predicate *modifier*) or as in the case of the **find_objects_and_modifiers** rule, the search is guided initially to the *modifier* facts and if there is no success, solutions are acquired in the *class* facts. The weak domain dependency derives from the fact that in the present application of 'Representing the Ground' there are no factors (modifiers) specified at the predicate *class* level; in this way the searching space is reduced and the system becomes more time-efficient. This is not considered to be total domain dependency as small additions are required to be made to the actual code in order to achieve total domain independency. As an example of the simplicity of the modification which are required in the code, the clause defining the high level predicate *find_attribute_and_value* is given below for both cases (for the sake of simplicity, the definition of this predicate presented here does not include calls to predicates concerned with the output format, as happens in the one presented in the listing of the program, in Appendix A.):

- Clause defining the predicate *find_attribute_and_value* in order to search only the facts described by the predicate *modifier*:

```
find_attribute_and_value(Name, Factor, Old_attlist, All_attlist):-
                modifier(Name, Attlist),
                get_attrib_value(Factor, Attlist, Old_attlist, All_attlist).
```

- Modification of the clause defining the predicate *find_attribute_and_value* in order to search both sets of facts, described by the predicates *class* and *modifier*:

```
find_attribute_and_value(Name, Factor, Old_attlist, All_attlist):-
                class(Name, -, Att_list),
                get_attrib_value(Factor, Att_list, Old_attlist, Temp_attlist).
                modifier(Name, Attlist),
                get_attrib_value(Factor, Attlist, Temp_attlist, All_attlist).
```

In the first case of the example presented above, Prolog searches for a match (with the first subgoal) in the facts described by the predicate *modifier* and when unification is achieved with the appropriate one (guided by the input value for the variable *Name*) PDC Prolog will call the second subgoal *get_attrib_value* and will endeavour to satisfy it. The second subgoal will trigger the *procedure* (a sequence of clauses defining the same predicate is called procedure in PDC Prolog) that defines the predicate *get_attrib_value* in order to find the attribute name and the attribute value(s) that correspond to the input value for the variable *Factor*. The clauses for this predicate are not given as they remain the same in both cases (these can be found in the Appendix A, where the program is listed). The variable *Old_attlist* which is initially given the value of empty list, [], is used to collect every solution generated at each recursive call of *get_attrib_value* and pass it down to next one. When all the solutions generated from the recursive clauses of this predicate have been found the *Old_attlist* gets bound to *All_attlist* (which is unbound until then). The value to which *All_attlist* is bound is then returned to the call.

In the second case it is simply necessary to add two subgoals in the body of the rule requesting PDC Prolog to perform the process described above twice, once for the predicate *class* and once for the predicate *modifier*. It is worth noting that in order to retain all the solutions generated from searching both sets of facts the value that the *Old_attlist* variable obtains after the first search is recorded as *Temp_attlist* which is passed down to the second search. The solutions generated from the second search are appended to the *Temp_list* to give finally the *All_attlist*.

CHAPTER 4

REPRESENTING GEOTECHNICAL FIELD TESTS

4.1 Introduction

The development of a Knowledge-Based System to assist in the selection of appropriate In-Situ Tests requires the representation of knowledge regarding the individual tests that could influence such a decision. This chapter describes a knowledge representation scheme suitable for geotechnical field tests, that corresponds to the needs of such a system.

In section 4.2 the hierarchy of the in-situ tests incorporated in the system is presented and its development is discussed. In section 4.3 the knowledge required to be included in the system is identified and a knowledge elicitation exercise which was carried out is presented. Finally, in section 4.4 the implementation of this representation in PDC Prolog is discussed.

An integral part of this chapter is concerned with brief descriptions of the tests included in the hierarchy; however, due to its size it is presented in Appendix D.

4.2 Hierarchy of In-Situ Tests

Testing in Geotechnical Engineering can be divided into *In-situ* testing, *Large Scale Field* testing, *Back Analysis*, and *Laboratory* testing. The system, at present, is concerned with in-situ tests performed mainly in soil, therefore only information about these tests is included. In further developments of the system, field tests used in rock, along with the other three categories of testing methods, could be analysed in a similar way to that described below.

A thorough review of in-situ testing has been conducted in an attempt to include recent developments. The development of the in-situ tests hierarchy proved to be a lengthy process. The British Standard Code of Practice for Site Investigation (BS 5930, 1981) and the Site Investigation Manual (Weltman and Head, 1983) were adopted as the starting point for the review of in-situ testing. The code describes methods of in-situ testing that were in regular use until the completion of drafting in about 1978 (Manby and Wakeling, 1990). The Site Investigation Manual was written in close association with the code. Since in-situ testing has developed rapidly in the last decade, it was quickly noted that most of the recent developments were not included in the first version of the hierarchy produced. This first attempt had to be revised eight times before the final version of the in-situ test hierarchy was achieved. The most critical stages of this process will be discussed in detail in this section.

The first version of the classification of in-situ tests, based on the British Standards (BS 5930, 1981) and the Site Investigation Manual (Weltman and Head, 1983), is presented in Appendix C. The tests were grouped under four headings, *Borehole* tests, *Probing* tests, *Non-borehole Field* tests and *Geophysical Surveying* (keeping almost the same groupings provided by the two sources of information). Each group was expanded at the most detailed level to individual testing methods. For example, Borehole tests were subdivided into Permeability tests, Standard Penetration test, Vane Test, Pressuremeter tests and Plate tests. Standard Penetration test and Vane Test are individual testing methods whereas Permeability tests, Pressuremeter tests and Plate tests represent groups of tests which could be further divided into lower level groups and finally into individual testing methods. Consequently, Permeability tests were subdivided into Open Borehole tests (subgroup) and Constant Head Test from Piezometers (individual test); Open Borehole Tests were divided into Variable Head tests (subgroup) and Constant Head test (individual test) and finally Variable Head tests were divided to Rising Head test (individual test) and Falling Head test (individual test).

Going through the literature, the first version of the in-situ test hierarchy was expanded by the inclusion of recent test methods (e.g. Flat Plate Dilatometer Test) among which were a number of self-boring tests

(e.g. Self-boring Plate Test, Self-boring Ko meter Test, etc.). Alterations were also made to test names trying to represent an in-situ testing technique by the name of the test method, rather than by the name of a specific device that operates according to the principle of the test method. For example, the Camkometer Test (name used by BS 5930 (1981) and Weltman and Head (1983), refering to a specific self-boring pressuremeter device) was renamed the Self-boring Pressuremeter Test and the Stressprobe Pressuremeter Test was renamed the Push-in Pressuremeter Test.

These changes led to the fourth version of the in-situ test hierarchy, presented in Appendix C, which incorporates the following principal alterations:

i) The Borehole Tests were divided into two separate groups: *Pre-bored* Tests and *Self-boring* Tests. The self-boring technique of insertion, which is a new method of investigating soil, causes minimal disturbance and allows the possibility of "near perfect" testing of undisturbed soil (Wroth, 1984). For this reason it should be treated separately from the pre-drilled borehole tests and the probing tests. This proposed course of action is supported by the comments of Robertson (1985) and Bageulin *et al* (1978) concerning the Self-boring Pressuremeter Test in comparison with the tests performed on the walls of pre-bored boreholes or those which displace the soil during the insertion.

ii) The Probing Tests were significantly expanded in two ways. Firstly, the Static Cone Penetration Test was divided into the Mechanical Cone Penetration Test and Electrical Cone Penetration Test which were themselves further divided into distinct tests such as Mechanical Cone Resistance and Mechanical Cone Resistance Friction tests, Electrical Cone Resistance, Electrical Cone Resistance Friction, Piezocone and Piezocone Friction tests respectively. Secondly, special purpose penetrometer devices which have been recently developed, were added, such as the Cone Pressuremeter Test, the Electrical Density Probe Test, the Acoustic Cone Test, etc.

iii) Nuclear Density Tests were also expanded to individual tests such as the Backscatter Test, Direct Transmission Test and Air Gap Test.

iv) Finally, the Geophysical Surveying Tests were also expanded in order to incorporate other tests in addition to those discussed in British Standards (BS 5930, 1981), such as the Seismic Cross-Hole Test, the Seismic Down-Hole Test and the Surface Wave Test.

A problem that arose in the early stages of the development process was an overlap between Borehole Tests and Probing Tests. For example, the Vane Test is a borehole test because it is conducted at the base of a borehole, but it can also be a probing test, penetrating the soil without the need for a borehole (BS 5930, 1981; Weltman and Head, 1983).

Several other devices such as the Total Stress Cell, the Iowa Stepped Blade, etc. could be utilised in both ways according to the soil conditions. In addition, similar tests in nature such as Pressuremeters were spread out in order to fit these categories.

In the introductory paragraph of the 'Tests in Boreholes' section in British Standards (BS 5930, 1981), it is written: "Paragraphs 21.2 to 21.7 describe the various forms of test that are commonly conducted as supplementary to a ground investigation carried out by borings. Inevitably, there is some overlap with section five" (section five describes the Field Tests). Also, "Clause 21 and section five are in a sense complementary to each other, and where a particular test is not described in one, it should be sought in the other".

Therefore it was considered that the grouping of in-situ tests adopted could not lead to a consistent hierarchy. It was thought that a more suitable way of organising them was to divide them into nine categories according to the nature of the tests. In this way grouping of tests similar in principle, as well as in their scope, is achieved.

Subsequent additions, deletions and rearrangements led to the final (eighth) version, presented in Appendix C. The following points concerning the development of this final version may be observed:

i) The Constant Head Test from Piezometers was included in the Constant Head Test.

ii) The Penetration Tests were expanded according to the International Standards (ISSMFE, 1988). More details are given in Appendix D.

iii) The Special Penetrometer Probes were separated from the Penetration Tests under the category name Special Penetrometer Tests. The Flat Plate Dilatometer was included in this category because it could be considered as a penetration tool and forms, with the Cone Pressuremeter Test, the sub-category Expansion Penetration Tests. There are other special penetrometer devices that could be incorporated in this category (Mitchell, 1988), although since a lot of these tests are not widely used yet (some are still at the research stage), it was thought that the ones included here demonstrate the wide variety of the recently developed penetrometer devices and their potential abilities.

iv) It was decided not to keep the different Plate Loading Tests shown in versions 1 and 4 separate since they are all similar in principle.

The final (eighth) version of the in-situ tests hierarchy is shown graphically in Figures 4.1 and 4.2. The in-situ tests have been organised into nine categories according to their nature, i.e. *Penetration* tests, *Special Penetrometer* tests, *Pressuremeter* tests, *In-situ Stress Measurement* tests, *Shear* tests, *Bearing* tests, *In-situ Density* tests, *Permeability* tests and *Geophysical Surveying* tests. Each category has been expanded at the most detailed level to individual in-situ testing methods. Due to the large volume of information, the test hierachy is presented in two figures, Figure 4.1 and Figure 4.2. As mentioned earlier in this chapter, only the in-situ tests branch has been expanded in detail. In Figure 4.1 four categories of the in-situ tests, Penetration, Pressuremeter, Shear and Permeability are fully developed

whereas in Figure 4.2 the Special Penetrometer, In-situ Stress Measurement, Bearing, In-situ Density and Geophysical Surveying categories are presented in detail.

Each level of detail in the tests hierarchy denotes a property of the levels below it. This is shown in the final version of the hierarchy (Appendix C) where the attribute names that correspond to different levels are given (for the maximum possible number of levels existing in the present hierarchy). For example, at the In-Situ Tests level the attibute *test category* is defined and inherited by all the levels below it, the Penetration Tests level corresponds to the attribute *test nature* and so on. Different paths through the hierarchy incorporate different number of levels.

A difficulty that was identified through the development process of the in-situ tests hierarchy, was that in many cases tests were described in the published literature under different names although the same test method was implied. For example, the Cone Penetration Test is also referred to as Static Penetration Test, Dutch Sounding Test (among other names).

In-situ testing has developed rapidly during the last few years and new developments are being achieved at a quick pace, as the interest of the engineering community in it continues to increase. Hence, the list of in-situ tests presented in this section (Figures 4.1, 4.2) is by no means exhaustive but it is believed that it covers the major in-situ test methods used in subsurface exploration and provides a good indication of the wide variety of in-situ tests that have been developed. Finally, it is thought that the in-situ test hierarchy that has been developed provides the basis for the inclusion of further developments.

As has already been mentioned in section 4.1, a brief description of all the tests included in the in-situ test hierarchy is presented in Appendix D.

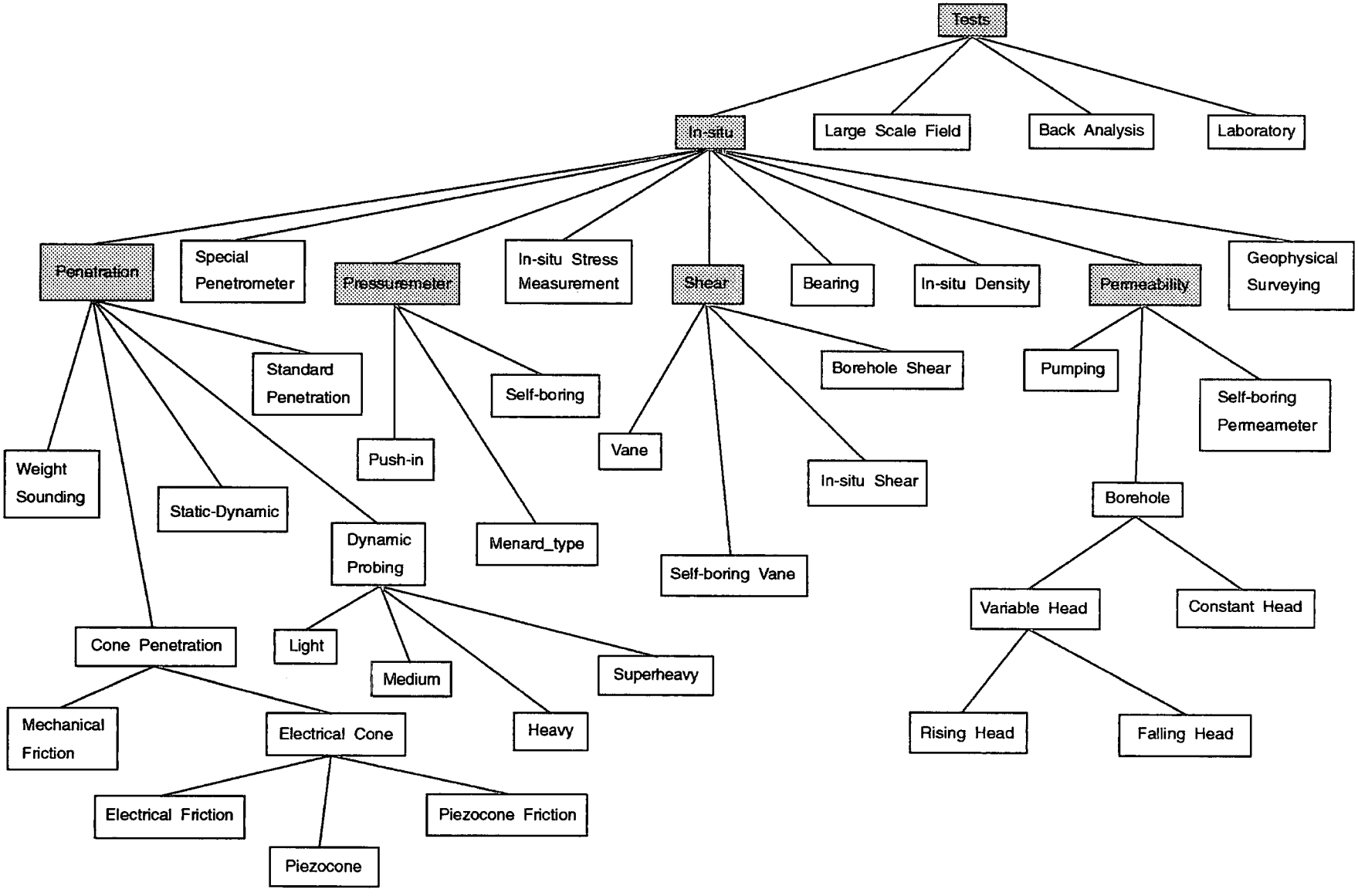


Figure 4.1 In-situ tests hierarchy

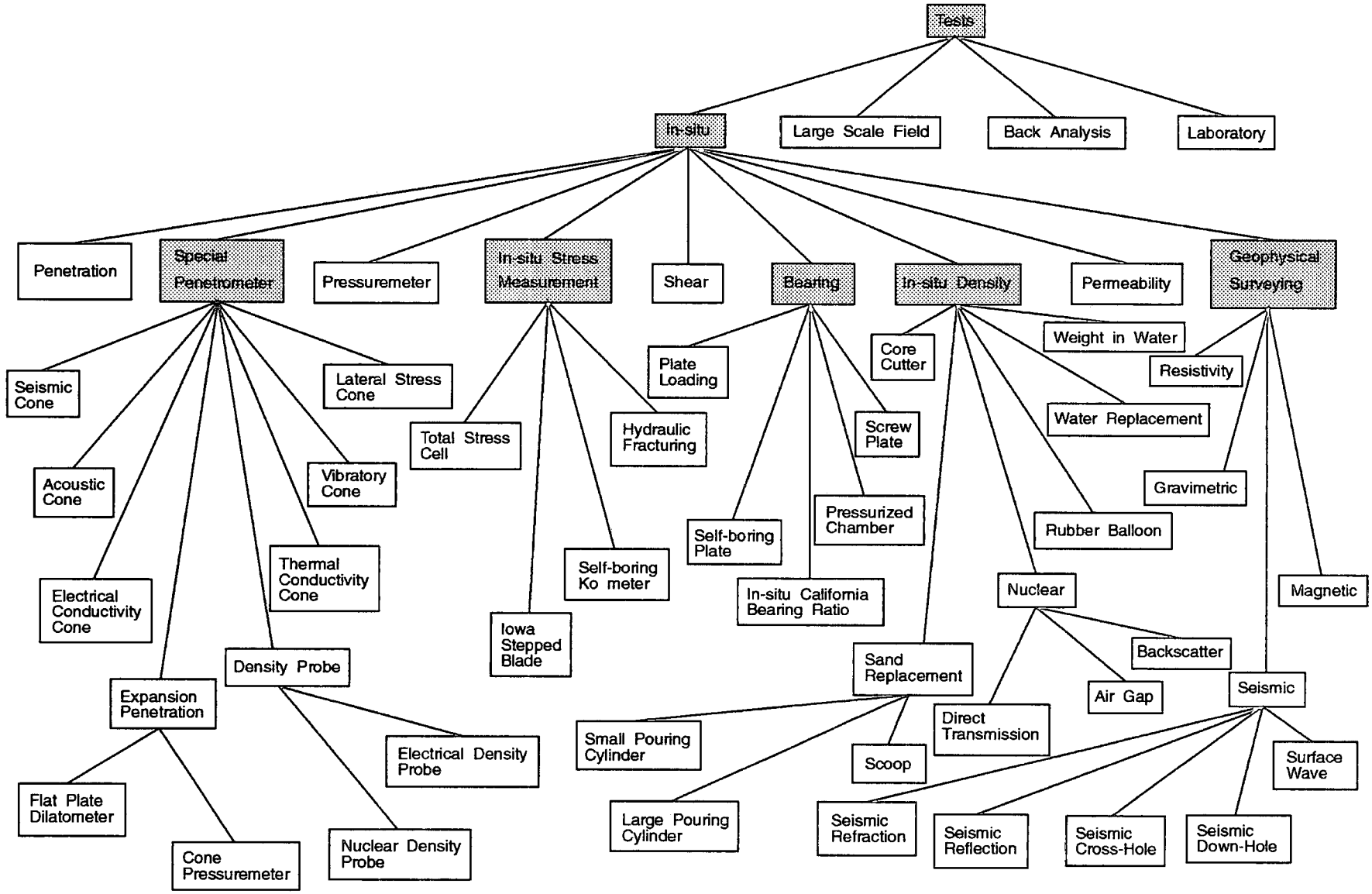


Figure 4.2 In-situ tests hierarchy

4.3 Knowledge Elicitation Exercise

4.3.1 Knowledge Required

The Tests Knowledge Base developed in the system contains the in-situ test hierarchy shown in Figures 4.1 and 4.2. In order for the system to be able to provide assistance in the selection of the appropriate field tests, knowledge about individual tests that could influence such a decision had to be incorporated. It was identified, going through published literature, that the suitability of a given technique is mainly governed by the following factors: geological conditions, project requirements and method of analysis intended for design (Robertson, 1985; Marsland, 1986; Orchant *et al*, 1988). These factors are briefly discussed below.

The geological conditions can vary from hard rock to soft soils and organics. Some of the in-situ tests are applicable in a wide range of soils whilst others are only applicable to specific soil types. For instance, Pressuremeter Tests and Plate Loading Tests, provide a means of obtaining shear strength parameters in a much wider range of soils than the Vane Test (Marsland,1986).

Intuitively, the most important factor affecting the suitability of any in-situ technique is whether the test provides the necessary information for the type of project under consideration, e.g. stratigraphic or profile information or specific soil properties for foundation design (Orchant *et al*, 1988). The project requirements and economics control the level of sophistication that should be adopted in the subsurface exploration and method of design in order to achieve the required accuracy of the prediction.

Design methods can, in general, be divided into those that use direct empirical correlations from in-situ tests measurements to design values, to those that employ soil properties in theoretically-based design equations. Although designs based on succesful past local experience usually provide insurance that unacceptable damage will not occur, they do not provide much guidance on the economy of the construction or the degree of extrapolation which is possible (Marsland, 1986). With the increasing use

of microcomputers for analyses, there will be an increase in the requirement for determination of the physical properties of the ground, such as shear strength and modulus (Robertson, 1985).

Two important factors, that are strictly connected with the performance of the in-situ test techniques, can be identified in the above analysis; their ability to derive geotechnical information and their use in different soil types. Knowledge about these two factors, if combined with the specific requirements of a given project, can provide assistance on the planning of the subsurface exploration using in-situ test techniques.

Hence, the knowledge included in the system about in-situ tests consists mainly of two types of information:

- The *applicability* of a test in different dominant soil types, and
- the *reliability* of a test to determine certain geotechnical parameters (assuming ideal ground conditions and taking into account all necessary correlations).

Additional information that could influence the selection of in-situ test methods are the *test objective*, *unit cost*, and *test frequency*. These are explained below.

The *test objective* could be defined by the type of information for which a test is primarily used for. In-situ tests can be divided into logging test methods, specific test methods (Robertson, 1985; Orchant *et al*, 1988) and combined test methods (Robertson, 1985). The logging test methods (e.g. the Penetration Tests), provide mainly stratigraphic information, although they may also be used to provide estimates of the soil properties through empirical and semi-empirical correlations. The specific test methods (e.g. the Pressuremeter Tests), are employed for the measurement of properties at a point and are usually more specialized and so, are often slower and more expensive to perform than the logging methods. The combined test methods, (e.g. the Cone Pressuremeter Test), form a new group of in-situ tests that combine the good features of the logging and the specific test methods.

The ideal procedure for conducting a subsurface exploration using in-situ test methods, according to Robertson (1985), is to first use a good logging test method to define the soil stratigraphy and to provide estimates of geotechnical design parameters. Based on this data, critical areas (where specific data may be required) are identified and if the additional information is considered necessary, specific in-situ test methods should be selected.

Another factor in assessing the applicability of an in-situ test technique is the familiarity of the engineering community with the method (Orchant *et al*, 1988). Tests must be field proven before design engineers will accept the validity of their results. An indication of the familiarity of the engineers with a test method could be obtained by knowing the frequency with which a test is used in Site Investigations, i.e. if it is a *routine* test, *less common* test or a *special purpose* test.

The cost of an in-situ technique also influences the applicability of the technique to a given project. According to Orchant *et al* (1988), the main factors affecting the overall cost of performing a particular test include the equipment and personnel requirements, capital cost of equipment, test duration, interpretation requirements, and mobilization and access requirements.

Knowledge about these additional factors has also been included in the system, where available.

4.3.2 Knowledge identified in published literature

As has already been argued, the knowledge relating to in-situ tests mainly required for inclusion in the system is their *reliability* in obtaining geotechnical parameters and their *applicability* in different ground conditions. It has been found to be difficult to identify this type of knowledge from the published literature for all the many types of field tests. The majority of the relevant publications, when presenting a test method, usually concentrate on some of the limitations or applications of the test and often discuss them in different levels of detail; as a result, the required knowledge in many cases is missing or is difficult to identify. Davey-Wilson and May (1989), in the course of the development of a

KBS for the selection of groundwater control methods, recognized the difficulty of obtaining a consistent knowledge base from published material. It is even more difficult to apply ratings in a consistent way for all the tests under consideration just by going through the literature, without personal experience, as each author expresses judgements in his own way.

The references that do provide relevant information in a suitable form are discussed below. In these references the ability of a test to obtain various geotechnical information and/or its use in different gound conditions, are graded using a 'four grade' rating of applicability:

High applicability

Moderate applicability

Limited applicability

Not applicable

Extensive use has been made of work by Robertson (1985, 1986). Robertson (1986) presents a table listing the major in-situ test methods available, their perceived applicability for use in different ground conditions (such as *Hard rock*, *Soft rock-Till*, *Gravel*, *Sand*, *Silt*, *Clay* and *Peat-Organics*), and their use in obtaining various geotechnical information (such as *Soil type*, *Profile*, *Piezometric pressure*, *Angle of friction*, *Undrained shear strength*, *Density*, *Compressibility characteristics*, *Rate of consolidation*, *Permeability*, *Modulus*, *In-situ stress*, *Stress history* and *Stress-strain curve*). The author notes that the grade assigned is based on his current personal experience and that it will vary according to one's own experiences and applications.

Orchant *et al* (1988) present a table that provides information on the range of soil types (such as *Gravel*, *Sand*, *Silt* and *Clay*) in which particular in-situ tests can be employed. Sand and Clay are divided in two categories according to their consistency: *loose Sand - dense Sand*, *soft Clay - stiff Clay*.

Mullarkey (1985), presents a table that summarises the current and potential types of information that the Cone Penetration Test and the Piezocone Penetration Test can provide. This information includes logging capabilities (such as *Soil type* and *Soil profile*), engineering parameters (such as *Relative density*, *Stress history*, *Coefficient of consolidation*, *Angle of friction* and *Undrained shear strength*) and design values (such as *Bearing capacity of piles*, *Settlement*, *Liquefaction*).

Knowledge about *test objective* was identified in Robertson's work (1985,1986) and Orchant's work (1988). In Table 4.1 values of the attribute test objective for the various categories of tests (as perceived by the author although based on the above references) are presented. Special Penetrometer Tests are not included in Table 4.1 since a unique value does not apply to all the individual tests belonging to this group; these are presented separately in Table 4.2.

Tests	Test objective
Penetration Tests	Logging test method
Pressuremeter Tests	Specific test method
In-situ Stress Measurement Tests	Specific test method
Shear Tests	Specific test method
Bearing Tests	Specific test method
In-situ Density Tests	Specific test method
Permeability Tests	Specific test method
Geophysical Surveying Tests	Logging test method

Table 4.1 Perceived values of the attribute *test objective* for various categories of field tests

Special Penetrometer Tests	Test objective
Flat Plate Dilatometer Test	Logging test method
Cone Pressuremeter Test	Combined test method
Lateral Stress Cone Test	Combined test method
Seismic Cone Test	Combined test method
Vibratory Cone Test	Combined test method
Nuclear Density Probe Test	Combined test method
Electrical Density Probe Test	Combined test method
Electrical Conductivity Cone Test	Combined test method
Thermal Conductivity Cone Test	Combined test method
Acoustic Cone Test	Logging test method

Table 4.2 Perceived values of the attribute *test objective* for Special Penetrometer Tests

Also, qualitative values (High, Medium, Low) of the *unit cost* of certain in-situ tests are given by Orchant *et al* (1988).

4.3.3 Knowledge obtained from the Questionnaire

In order to expand the body of knowledge, found in published literature, for all the in-situ test methods shown in Figures 4.1 and 4.2, and to incorporate other experts' impressions of the various field tests, a knowledge elicitation exercise in the form of a questionnaire was carried out.

The questionnaire (Appendix E) was designed so that the in-situ tests under consideration are listed on seven individual sheets, each containing related categories of test. It was felt that it would be difficult for one person to complete the questionnaire for all tests; therefore the above form of presentation would allow the distribution of the individual sheets of one questionnaire to experts having most familiarity with the particular category of test method. These categories were based on the groupings adopted in the final version of the test hierarchy (Appendix C).

The information required of the experts is to identify the *reliability* of the tests for obtaining geotechnical information (assuming ideal ground conditions and taking into account all the necessary correlations needed to derive the geotechnical information), and their *applicability* for use in different ground conditions. Under the heading Geotechnical Information the most common soil parameters, used in geotechnical design, are included in the way they were identified by Robertson (1986). The same principle has been adopted for the dominant soil types included under the heading Ground conditions. The questionnaire was based on Robertson's work in order to obtain results directly comparable with his perceptions. Another reason was to allow Roberton's work and the results obtained from the questionnaire to be complementary to each other.

The experts were also asked to specify the familiarity they have with each test, and how frequently these tests are used in Site Investigation. This could provide a feel for the familiarity of the engineering community with each of these tests.

The experts were required to give their expertise using the following ratings according to the heading under examination:

Geotechnical Information	
H	High reliability
M	Medium reliability
L	Low reliability
N	None reliability

Ground Conditions	
H	High applicability
M	Medium applicability
L	Low applicability
N	None applicability

Familiarity with Test	
H	High familiarity
M	Medium familiarity
L	Low familiarity
N	None familiarity

Test Frequency	
R	Routine test
L	Less common test
S	Special purpose test

Space was provided on each individual sheet to be used by the experts for their comments. An appendix was also provided giving alternative names for some of the tests in order to avoid confusion.

The questionnaire was sent to thirty experts as a pilot study. Only eight completed questionnaires were received back. Three more questionnaires were made up, each containing the knowledge provided by Robertson (1986), Mullarkey (1985), Orchant *et al* (1988) respectively.

The results obtained from the eleven questionnaires are presented in Tables 4.3-4.9. The method of analysis is now discussed.

For each category of tests four sets of results were produced, shown in Tables E.1-E.21 (Appendix E): a) sum of all available answers of the questionnaire, for all tests included in each test category, in correspondance with the defined ratings (each fill-in box presents total numbers of answers that are in favour of each of the applicable ratings (H, M, L, and N or R, L and S) for a particular question), b) sum of the 'high familiarity' answers only (neglecting results where the experts indicated they had only 'medium', 'low' or 'none familiarity' with a particular test), for all tests included in each test category, in correspondance with the defined ratings, c) Average values of the results that take into account all answers (A_I) and d) Average values of the results that take into account only the 'high familiarity' answers (A_{II}). It should be noted here that values provided by the references (made-up questionnaires) were considered as 'high familiarity' answers.

The average values for the *reliability*, *applicability* and *test frequency* were calculated using the following numerical scales:

Reliability/Applicability	Test Frequency
H = 1	R = 1
M = 2	L = 2
L = 3	S = 3
N = 4	

The values produced were rounded up to the nearest integer. The scale was chosen so that on rounding up the most conservative solution is obtained.

For example, the four sets of results produced for the *Penetration Tests* are presented in Tables E.1-E.3. Table E.1 aggregates the answers obtained from all experts, independently of their familiarity with the tests whereas Table E.2 shows only the answers obtained from the experts having high familiarity with these tests. Finally, the average values (A_I) and (A_{II}) of the results presented in Tables E.1 and E.2 respectively are given in Table E.3 . When all the experts which provided knowledge for this test have high familiarity with it, one average value (A) is shown. For instance, for the *Piezocone Test* concerning the *angle of friction* taking into account all answers (Table E.1) the following results have been obtained: no 'high reliability' answers (indicated by a hyphen), five 'medium reliability' answers, one 'low reliability' answer and two 'none reliability' answers. Taking into account only the 'high familiarity' answers (Table E.2) the results are: four 'medium reliability' answers and no answers for the other three applicable ratings. In Table E.3 the corresponding calculated average values are: *low* reliability (A_I) and *medium* reliability (A_{II}). A hyphen in these sets of results indicates that there is no information obtained to allow the computation of the corresponding average value.

The final value is taken as the average value of the 'high familiarity' answers (A_{II}), provided that the difference between this value and the average value of all answers (A_I) is not more than one rating (upwards or downwards); for the example discussed above, the final rating obtained for the *Piezocone*

Test concerning the *angle of friction* is *medium* reliability. When the difference between A_I and A_{II} was more than one rating, individual cases had to be considered. These are discussed later on in this section. When there were no 'high familiarity' answers available, the final value was usually taken as the average value of all answers unless the corresponding data were considered to be of suspicious reliability. In the latter case it was assumed that no knowledge was obtained. A hyphen, (-), is used where applicable, to indicate that there is no knowledge available.

In addition, the results obtained from all the experts for three test methods, the Standard Penetration Test, the Electrical Penetrometer Friction Test and the Self-boring Pressuremeter Test are presented as histograms in Figures 4.3-4.8. In these histograms different shading has been used, according to the degree of familiarity (High, Medium, Low, None) of each answer as well as for the answers obtained from the three references.

In general it can be observed from the histograms that there is reasonable agreement between the experts. However there are cases where the answers are spread, covering the whole scale of applicability/reliability, as happens for instance in the case of Self-boring Pressuremeter test for the rating of its reliability to obtain piezometric pressure (Figure 4.7). It is also interesting to note that in most cases the experts having high familiarity with a particular test seem to have common impressions for it. This is not however always the case as can be seen from the answers given for the Standard Penetration Test (Figures 4.3 and 4.4), where all the respondees say they have high familiarity with it.

Before going into further details on the analysis of the questionnaire it should be noted that the sample is very small, therefore it is difficult at this stage to be confident about the results. An additional problem was that none of the returned questionnaires (or the made-up questionnaires), were fully completed, mainly due to the size of the questionnaire and the large amount of information required.

In the rest of this section the followings points will be discussed briefly:

1) Remarks on certain questionnaires

2) Problems identified in the analysis of the questionnaires

3) Comments expressed by the experts

4) Remarks on the results obtained from the exercise in general

1) Remarks on certain questionnaires

Made-up Questionnaires:

- Questionnaire 2 (Mullarkey's work):

Mullarkey provides ratings for the reliability of the Electrical Cone Penetration Test and the Piezocone Penetration Test. It was assumed for the made-up questionnaire that the values given for the Piezocone Test are applicable to the Piezocone Friction Penetration Test as well.

- Questionnaire 3 (Orchant *et al*'s work):

The ratings for the applicability of in-situ tests in common soil conditions, given by Orchant *et al* are not directly applicable to the form of the questionnaire as the authors distinguish between *loose* Sand-*dense* Sand and *soft* Clay-*stiff* Clay. The way this knowledge was incorporated in the form of the questionnaire was to accept the highest applicability rating given for the same dominant soil type; the other is considered as the exception and is included in the knowledge of the system as such.

It was also assumed that the values given for the Cone Penetration Test with pore pressure measurements are applicable for both the Piezocone Penetration Test and the Piezocone Friction Penetration Test.

Returned Questionnaires

A general remark is that in many cases there was no consistency in the way the respondees completed the questionnaire. It was observed in 4 out of 8 questionnaires that the experts left many blanks in their answers about a test. In some cases this could be because they were not particularly familiar with that specific test. In other cases, however, it seems as if they have been tired of filling in boxes and assuming that it was obvious that certain answers were negative (i.e. None applicability/reliability), due to the nature of the test, they left the corresponding boxes blank. It was felt, however, that it would be more consistent to ignore the empty boxes in all cases in the analysis, instead of guessing the experts' impressions about the tests.

- Questionnaire 4:

The respondee's answer for the Self-boring Ko meter Test is ignored since it was based on the impression that this test is the same as the Self-boring Pressuremeter Test (identified by the respondee in the comments section).

- Questionnaire 6:

In Special Penetrometer Tests the expert provides knowledge about the reliability of some test methods to obtain geotechnical parameters for tests although he has no familiarity with them. In most cases these answers are negative and are part of a general answer given for a certain parameter for the corresponding sequence of tests (i.e. a column corresponding to a particular rating had been shown as a particular rating for all tests on that page). In all these cases no knowledge is provided for the applicability of the test in different ground conditions. These answers were taken into account, as complete knowledge about the geotechnical information was provided for these tests.

A similar problem appeared in the expert's answers for some of the Shear and Bearing Tests. In these cases, however, no knowledge was provided about geotechnical parameters that are of interest for the

particular tests; only negative answers given in the way discussed above were supplied. In this case the answers for these particular tests were ignored.

- Questionnaire 7:

The partially completed answer for the Nuclear Air Gap Test was ignored for the same reasons explained in Questionnaire 6 for the case of the Shear and Bearing Tests.

- Questionnaire 8:

The expert provided no knowledge for any of the Special Penetrometer Tests, apart from filling the first three boxes under the heading Geotechnical Information for the Flat Plate Dilatometer Test. All the three answers were negative. As he has no familiarity with any of these tests including the Flat Plate Dilatometer Test his answers were ignored.

Also, the answers given for the Self-boring Ko meter Test were ignored as it seems that they are based on the assumption that the test is the same as the Self-boring Pressuremeter Test.

2) Problems identified in the analysis of the questionnaires

Special Penetrometer Tests

- Cone Pressuremeter Test

There is only one 'high familiarity' answer available for this test. From the results given by the respondee it seems that the contribution of the 'cone penetration part' of the test is ignored. For this reason, a final value is taken as the average based on all answers (A_I).

- Vibratory Cone Test

None of the experts responding to the questionnaire had high familiarity with this test. Only one expert provided information about the reliability of the test to obtain engineering parameters. This expert has no familiarity with the test and provides no values for the applicability of the test to different ground conditions. It was felt that there were not sufficient data provided for this test, therefore it was considered that no knowledge is available for it.

- Electrical Conductivity Cone Test

The two averages calculated for the reliability with which the parameter *density* can be obtained from the Electrical Conductivity Cone Test differ by more than one rating. The high familiarity average was considered to be more reliable; therefore it was taken as the final value. It has to be said however, that only two experts provided information on this test, hence the results are indicative only.

Pressuremeter and In-situ Stress Measurement Tests

- Push-in Pressuremeter Test

In four cases (*soil type*, *piezometric pressure*, *rate of consolidation* and *permeability*), the two calculated averages differ more than one rating range. It was decided to take A_I as the final value.

- Hydraulic Fracturing Test

The same problem appears in the two averages calculated for the applicability of the test for use in Clay. It was decided to take A_{II} as the final value.

- Self-boring Ko meter Test

Two answers were ignored as they were based on the assumption that the Self-boring Ko meter Test is the same as the Self-boring Pressuremeter Test.

In-situ Density Tests

In-situ Density Tests is a good example to demonstrate the problem identified earlier on in this section, concerning the blank spaces left by the experts in their answers about a test.

- Water Replacement Test, Rubber Balloon Test

In both these tests there is one high familiarity answer that provides knowledge concerning the reliability only for two parameters (which are of interest for these tests). Consequently, for these parameters A_{II} was taken as the final value. The knowledge concerning the reliability for the other parameters is completed by A_{I}.

Permeability Tests

- Pumping Tests

In one case (*piezometric pressure*), the two calculated averages differ more than one rating range. It was decided to take A_{II} as the final value.

3) Comments obtained from the experts

Some interesting comments were received from the experts as part of their answers to the questionnaire. These are briefly discussed below.

An interesting point on the content of the questionnaire was made by the experts answering questionnaires 8 and 9. They recommend that the category *soft rock - till* existing under the heading Ground Conditions, should be separated since Tills cover a wide range of strength and stiffness and they have special problems due to cobbles and boulders. In questionnaire 4, the expert thinks that it may be worth identifying permeabilty tests in piezometers separately.

Some confusion seemed to exist between the experts, relating to two test methods: The Self-boring Ko meter Test and the Cone Pressuremeter Test. This became obvious either by their comments or their responses. In questionnaires 4, 7 and 8 the experts are under the impression that the Self-boring Ko meter Test is the same as the Self-boring Pressuremeter Test. The rest of the experts that responded to this part of the questionnaire seem to recognise the difference between these two tests. A similar problem occured in the case of the Cone Pressuremeter Test. One expert has based his answers on the assumption that a piezocone is used (Questionnaire 4) whilst others that it is an electrical cone with no pore pressure measurements facilities (Quetionnaires 5 and 6). In particular one of them (Questionnaire 6) seems to ignore completely the 'cone penetration part' of the test.

Four experts commented on existing restrictions concerning the applicability of some test methods in certain ground conditions. The expert of the Questionnaire 6 remarks that the use of cone testing (Penetration Tests) is restricted by risk of damage to probe and that its application in soil is restricted by density. The same expert identifies the risk of damaging the cone probes used in Special Penetrometer devices, in soft rock and gravel. The expert of the Questionnaire 9, identifies that Rising and Falling Head Tests (Permeability Tests) are unsuitable for very compressible clays. Also the Rising Head Test can cause piping in loose sands. Finally, he comments that permeability tests of any kind can be difficult in very permeable gravels. Finally, in Questionnaires 6 and 7, the experts comment that the applicability of the Geophysical Surveying Tests depends really on the amount of contrast between different conditions on site.

A more general comment has been made by the expert of questionnaire 10, relating to the Nuclear Backscatter Test. The expert reports that an ASTM standard exists for this test.

Finally, the expert of questionnaire 11 provides some interesting clarifications for some of his answers; for example, he comments on the 'high reliability' ratings that he gave to the Standard Penetration Test for obtaining information on *soil type* and *profile*. He states that these answers are based on the

assumption that the material retained in the split barrel sampler is examined. Also in the case of the Constant Head Test (Permeability Tests) he gave a 'high reliability' rating for deriving information on *rate of consolidation* (c_v, c_h) assuming that the parameter is derived indirectly using the coefficient of volume compressibility (m_v) obtained from laboratory testing. Two more general comments that he made were that the Mechanical Cone Penetration testing has largely been replaced by the Electrical Cone Penetration testing and that drained or undrained In-situ Shear Tests may be carried out.

4) Remarks on the results obtained from the exercise in general

The results of the questionnaire could be summarized as follows:

- No knowledge has been collected for the following tests:
  - Weight Sounding Test
  - Vibratory Cone Test
  - Pressurized Chamber Test
- Tests with partially available knowledge
  - Nuclear Air Gap Test
- No 'high familiarity' knowledge has been collected for the following tests
  - Lateral Stress Cone Test (not at all)
  - Thermal Conductivity Cone Test (not at all)
  - In-situ Shear Test (not at all)
  - Scoop Test (not at all)
  - Water Replacement Test (partially)
  - Rubber Balloon Test (partially)
  - Nuclear Air Gap Test (not at all)
  - Self-boring Permeameter Test (not at all)
  - Resistivity Test (partially)

Gravimetric Test (partially)

Magnetic Test (partially)

In general the results of the questionnaire were found promising because i) only for 4 tests out of 60 is there no knowledge at all or there is incomplete knowledge available and ii) in the majority of the tests examined there were not major disagreements between the experts. It must be noted, however, that the sample (eleven questionnaires) is very limited to allow represenative results to be obtained. In addition to that, in many tests (most of which were identified as less common or special purpose), not all the experts provided answers. There were cases where two (or even just one) experts provided knowledge. All these limitations make some of the results obtained from the knowledge elicitation exercise to be only indicative.

Another factor that created difficulties in obtaining this knowledge is the large amount of information required from the experts. Although this factor was taken into account when the questionnaire was being designed, it was decided to go forward with this exercise for two main reasons. The first reason was that this knowledge is essential for the system and there is no other means of obtaining it, as it is mostly gained through experience. Secondly, it was thought to be a good exercise that would allow useful observations and remarks to be noted.

An interesting point that has already been raised, is the evaluation of the raw data received from questionnaires. It is usually very difficult to avoid inconsistencies in the way respondees give their answers. The question that arises is to what level these raw data can be controlled or manipulated. It is difficult to decide on the right approach in cases where there have been oversights by the respondee (that could be assumed or ignored), or obvious errors (that could be ignored or accepted). Two examples are given below.

It was mentioned earlier in this section that a common oversight of the respondees was to leave blanks in places where it is most likely that the answer would be negative. Another example that demonstrates this problem was identified in the questionnaire 6 in In-situ Density Tests, where the respondee uses twice the letter M for answers relating to *test frequency* (R, L, S). All the other answers that he provided for this category of tests concerning the *test frequency* were expressed either by the letter R or the letter S. It could be assumed that the respondee meant to use the letter L for those two answers. However, the approach adopted for this case was to ignore these answers.

An obvious error that was identified in questionnaire 4, is that the reliability of obtaining the *piezometric pressure* for the Piezocone Test was given as *None*, although in the same questionnaire the reliability of obtaining the piezometric pressure for the Piezocone Friction Test was given as *High*. It is believed that it is difficult to control the answers of the respondees at that level (checking for obvious errors in each filled box), consequently this case and any other similar cases that were identified were taken into account.

It is believed that most of the problems described above or mentioned earlier on in this section, would be overcome (or minimised), by having a large number of responses. In addition, the knowledge obtained with a larger sample would be more reliable. Due to time constraints, it was not possible to circulate the questionnaire to a larger pool of experts; a complete knowledge elicitation exercise that would take into account any relevant comments expressed by the experts, should be conducted in further development of the system.

A relevant issue that could also be investigated in further developments of the system is the possible application of existing models for the management of uncertain information from human sources in the analysis of the questionnaire. A Source Control System (Bokma, 1991; Garigliano and Bokma, 1992) has been developed based on the fundamental principle that the uncertainty of information from people can, in the majority of situations, successfully be assessed through source models which record factors concerning the respective source's abilities and trustworthiness.

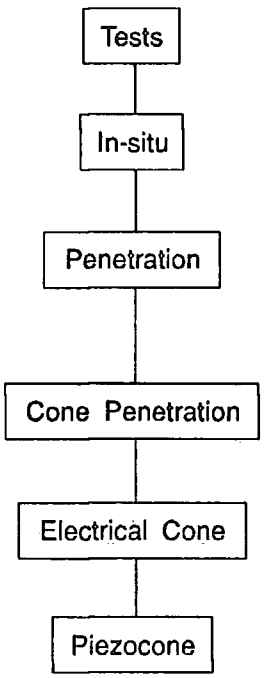
4.4 Implementation in Prolog

It has been possible to represent both the soil information (described in Chapter 3) and the test information using the same structures. The Tests Knowledge Base is implemented in PDC Prolog, as described in chapter 3 for the Ground Knowledge Base. Hence the implementation of the in-situ test model will only be discussed briefly here. The reader should refer to section 3.3 for futher details.

The predicate *class* was used to describe the in-situ test hierarchy (Figures 4.1 and 4.2) and the relationships between the classes of the hierarchy down to the most detailed level, the individual test methods (Standard Penetartion Test, Flat Plate Dilatometer Test, Ménard-type Pressuremeter Test, Total Stress Cell Test, Vane Test, Screw Plate Test, Small Pouring Cylinder Test, Rising Head Test, Seismic Refraction Test, etc.). At the predicate *class* level, attributes that allow identification of the exact position of an object within the structure (i.e. *test category*, *test nature*, *test group*, *test type*), as well as attributes that express general knowledge about an object of the structure such as *test objective*, *test frequency* and *unit cost*, were included. These are illustrated in Figure 4.9 that shows a path through the structure, from *Tests* to *Piezocone* Test.

The attributes *test category*, *test nature*, *test group*, *test type*, *test objective*, are identified at one level and are passed down the hierarchy to apply to lower level objects by inheritance. The attributes *test name*, *test frequency*, *unit cost* are defined at an instance level.

In the case of the Special Penetrometer Tests the attribute *test objective* is defined at the instance level, because the same value does not apply to all the individual tests (instances) that form this category. For example the attribute *test objective* takes the value *combined test method* for the Cone Pressuremeter Test whilst the same attribute takes the value *logging test method* for the Flat Plate Dilatometer Test.



Attributes	Values
Test category	In-situ Tests
Test nature	Penetration Tests
Test objective	Logging test method
Test group	Cone Penetration Tests
Test type	Electrical Cone Penetration Tests
Test name	Piezocone Test
Test frequency	Less Common
Unit cost	Medium

Figure 4.9 - A path through the in-situ tests hierarchy

It can be observed from the example that the representation of the in-situ tests requires only one type of attribute; there are no attributes which appear at several levels within the structure and whose values become more specific upon descending the structure, as was the case for the Ground Knowledge Base.

The predicate *modifier* is also used to handle specific information concerning the individual in-situ tests of the hierarchy, such as their *reliabilty* to obtain geotechnical parameters and their *applicability* in different dominant soil types. Knowledge about their *applicability* in different dominant soil types described by a modifier (e.g. dense sand), is also included where available.

The actual code in PDC Prolog for defining the class Penetration Tests at the predicate *class* level and the instances Standard Penetration Test and Flat Plate Dilatometer Test using both predicates *class* and *modifier*, is given below.

The definition of the class Penetration Tests is given by the predicate *class* as follows:

```
class (penetration_tests, [standard_penetration_test, dynamic_probing __tests,
                           cone_penetration_test, weight_sounding_test,
                           static_dynamic_penetration_test],
       [att ( test_nature,
              [ val ( [penetration_tests ],        fact ( [ ] )        )]),
        att ( test_objective,
              [ val ( [ logging_test_method ],     fact ( [ ] )        )])]).
```

The definitions of the instances Standard Penetration Test and Flat Plate Dilatometer Test are given by the predicate *class* as follows:

```
class ( standard_penetration_test, [ ],
       [att ( test_name,
              [ val ( [ standard_penetration_test ],     fact ( [ ] )        )]),
        att ( test_frequency,
              [ val ( [ routine ],                       fact ( [ ] )        )]),
        att ( unit_cost,
              [ val ( [ medium ],                        fact ( [ ] )        )])]).

class ( flat_plate_dilatometer_test, [ ],
       [att ( test_name,
              [ val ( [ flat_plate_dilatometer_test ],   fact ( [ ] )        )]),
        att ( test_frequency,
              [ val ( [ special_purpose ],               fact ( [ ] )        )]),
        att ( unit_cost,
              [ val ( [ low ],                           fact ( [ ] )        )])]).
```

The more detailed representation of the instances Standard Penetration Test and Flat Plate Dilatometer Test, as given by the predicate *modifier*, is presented below. The values that are specified for the attributes *applicability* and *reliability* have been taken from the results of the knowledge elicitation exercise (Tables 4.3-4.9).

```
modifier ( standard_penetration_test,
   [att ( applicability,
          [ val ( [ high ],    fact ( [ sand ] ) ),
            val ( [ medium ],  fact ( [ soft_rock, gravel, silt, clay ] ) ),
            val ( [ low ],     fact ( [ organic_sand, organic_silt, organic_clay, peat ] ) ),
            val ( [ none ],    fact ( [ hard_rock ] ) )]),
    att ( reliability,
          [ val ( [ high ],    fact ( [ ] ) ),
            val ( [ medium ],  fact ( [ soil_type, profile, angle_of_friction, density, modulus,
                                        undrained_shear_strength] ) ),
            val ( [ low ],     fact ( [ compressibility ] ) ),
            val ( [ none ],    fact ( [ piezometric_pressure, rate_of_consolidation, permeability,
                                        in_situ_stress, stress_history, stress_strain_curve] ) )] ) ]).

modifier ( flat_plate_dilatometer_test,
   [att ( applicability,
          [ val ( [ high ],    fact ( [ sand, silt ,clay, organic_sand, organic_silt, organic_clay,
                                        peat ] ) ),
            val ( [ medium ],  fact ( [ dense_sand, stiff_clay ] ) ),
            val ( [ low ],     fact ( [ ] ) ),
            val ( [ none ],    fact ( [ hard_rock, soft_rock, gravel ] ) )]),
    att ( reliability,
          [ val ( [ high ],    fact ( [ ] ) ),
            val ( [ medium ],  fact ( [ undrained_shear_strength, compressibility, in-situ_stress,
                                        modulus, stress_history, stress_strain_curve] ) ),
            val ( [ low ],     fact ( [ soil_type, profile, angle_of_friction, density ] ) ),
            val ( [ none ],    fact ( [ piezometric_pressure, rate_of_consolidation,
                                        permeability ] ) )] ) ]).
```

The extended inference mechanism described in chapter 3, section 3.3.4 is directly applicable to the 'Representing Geotechnical Field Tests' application. The search rules described there are also used for the test model in order to allow inheritance and transitivity inferences as well as information retrieval from the facts describing the in-situ tests.

PENETRATION TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Standard Penetration Test	M	M	N	M	M	M	L	N	N	M	N	N	N			N	M	M	H	M	M	L
	S	Dynamic Probing Light Test	N	H	N	L	L	L	L	N	N	N	N	N	N			N	L	L	M	M	M	M
	L	Dynamic Probing Medium Test	L	M	N	L	L	L	L	N	N	L	N	N	N			N	L	M	M	M	M	M
	S	Dynamic Probing Heavy Test	L	H	N	M	L	L	L	N	N	L	N	N	N			N	L	M	M	M	M	M
	S	Dynamic Probing Superheavy Test	L	M	N	L	L	M	N	N	N	L	N	N	L			N	L	M	H	M	M	M
	R	Mechanical Cone Resistance Friction Test	M	H	N	M	M	M	L	N	N	L	N	N	N			N	M	L	H	H	H	M
	L	Electrical Cone Resistance Friction Test	M	H	N	M	M	M	M	N	N	M	N	N	N			N	L	L	H	H	H	H
	L	Piezocone Test	M	H	M	M	M	M	M	M	L	M	N	L	L			N	L	L	H	H	H	H
	L	Piezocone Friction Test	H	H	H	M	M	M	M	M	M	M	L	M	L			N	L	L	H	H	H	H
	-	Weight Sounding Test	-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	Static - Dynamic Penetration Test	L	M	N	M	M	L	M	N	N	M	N	N	N			N	M	M	M	M	M	M

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Table 4.3 Final ratings obtained from the questionnaire for the Penetration Tests

SPECIAL PENETROMETER TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
	S	Flat Plate Dilatometer Test	L	L	N	L	M	L	M	N	N	M	M	M	M
	S	Cone Pressuremeter Test	L	M	N	M	M	L	L	M	L	M	L	L	M
	S	Seismic Cone Test	M	H	L	L	L	L	L	L	L	H	L	L	L
	S	Lateral Stress Cone Test	N	N	N	N	L	N	L	N	N	M	M	L	L
	-	Vibratory Cone Test	-	-	-	-	-	-	-	-	-	-	-	-	-
	S	Nuclear Density Probe Test	N	N	N	M	N	H	N	N	N	N	L	N	L
	S	Electrical Density Probe Test	M	M	N	M	L	H	L	N	N	L	N	N	N
	S	Electrical Conductivity Cone Test	H	M	N	L	L	H	M	L	L	M	L	L	L
	S	Thermal Conductivity Cone Test	N	N	N	N	N	N	N	N	N	N	N	N	N
	S	Acoustic Cone Test	M	M	N	L	L	L	L	N	N	L	N	L	N

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	N	N	N	H	H	H	H
	N	L	L	M	H	M	M
	N	L	L	H	H	H	H
	L	M	L	L	M	M	M
	-	-	-	-	-	-	-
	N	N	N	H	H	M	H
	N	L	N	H	H	H	H
	N	N	N	H	H	H	M
	N	N	N	H	H	H	H
	N	L	N	H	H	H	H

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Table 4.4 Final ratings obtained from the questionnaire for the Special Penetrometer Tests

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	S	Ménard-type Pressuremeter Test	L	L	N	M	M	N	N	N	N	M	L	L	M			M	H	M	M	M	H	M
	S	Push-in Pressuremeter Test	L	L	N	L	M	L	L	L	N	M	L	L	L			N	N	N	M	H	H	M
	S	Self-boring Pressuremeter Test	L	L	M	M	M	L	L	L	L	H	H	L	M			L	M	L	M	H	H	H
	S	Total Stress Cell Test	N	N	N	N	N	N	N	N	N	N	M	M	N			N	N	N	N	L	H	H
	S	Iowa Stepped Blade Test	N	N	N	N	N	N	N	N	N	N	M	M	N			N	N	N	M	H	H	M
	S	Hydraulic Fracturing Test	N	N	H	N	N	N	N	L	L	N	M	M	N			M	M	L	L	M	H	L
	S	Self-boring K_o meter Test	M	M	N	N	N	N	N	N	N	N	H	H	N			N	N	N	M	H	H	H

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Table 4.5 Final ratings obtained from the questionnaire for the Pressuremeter and In-situ Stress Measurement Tests

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (Dr)	COMPRESSIBILITY (mv, Cc)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Vane Test	N	L	N	N	H	N	N	N	N	N	N	N	N			N	L	N	N	M	H	M
	R	Self-boring Vane Test	L	L	N	N	H	N	N	N	N	N	N	L	N			N	N	N	L	L	H	L
	-	Borehole Shear Test	L	L	N	M	L	N	N	N	N	L	N	L	N			M	M	L	M	M	L	L
	S	In-situ Shear Test	N	N	N	H	H	N	N	L	N	L	N	N	L			N	M	N	N	M	M	N
	S	Plate Loading Tests	N	N	N	N	M	N	M	L	N	M	N	N	M			M	H	H	H	H	H	M
	S	Screw Plate Test	L	L	N	L	M	M	M	L	L	H	L	M	M			N	N	N	H	H	H	H
	-	Self-boring Plate Test	M	M	N	L	M	M	M	L	L	H	M	H	L			N	N	N	M	H	H	M
	-	Pressurized Chamber Test	-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	In-situ California Bearing Ratio Test	N	N	N	N	L	N	N	N	N	L	N	N	N			N	L	M	M	M	M	L

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Table 4.6 Final ratings obtained from the questionnaire for the Shear and Bearing Tests

IN-SITU DENSITY TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Small Pouring Cylinder Test	L	N	N	N	N	H	N	N	N	N	N	N	N			N	M	M	H	H	H	L
	R	Large Pouring Cylinder test	L	N	N	N	N	H	N	N	N	N	N	N	N			N	M	M	H	H	M	M
	S	Scoop Test	N	N	N	N	N	M	N	N	N	N	N	N	N			N	L	L	M	M	M	L
	R	Core Cutter Test	N	N	N	N	N	H	N	N	N	N	N	N	N			N	L	N	N	L	H	M
	L	Weight in Water Test	N	N	N	N	N	M	N	N	N	N	N	N	N			M	L	N	N	N	N	N
	S	Water Replacement Test	L	N	N	N	N	M	N	N	N	N	N	N	N			M	M	N	N	N	L	N
	S	Rubber Balloon Test	M	N	N	N	N	M	N	N	N	N	N	N	N			N	N	L	L	L	L	L
	R	Nuclear Backscatter Test	N	N	N	N	N	M	N	N	N	N	N	N	N			L	M	M	M	M	M	L
	R	Nuclear Direct Transmission Test	N	N	N	N	N	M	N	N	N	N	N	N	N			L	M	M	M	M	M	L
	S	Nuclear Air Gap Test	L	-	-	-	-	M	-	-	-	-	-	-	-			L	L	L	L	L	L	L

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Table 4.7 Final ratings obtained from the questionnaire for the In-situ Density Tests

PERMEABILITY TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Rising Head Test	N	N	L	N	N	N	N	N	M	N	N	N	N			L	L	H	H	H	L	L
	R	Falling Head Test	N	N	N	N	N	N	N	N	M	N	N	N	N			L	L	H	H	M	L	L
	R	Constant Head Test	N	N	L	N	N	N	N	N	M	N	N	N	N			L	L	M	M	M	L	L
	S	Self-boring Permeameter Test	N	N	L	N	N	N	N	N	H	N	N	N	N			N	L	N	M	H	M	L
	L	Pumping Tests	N	N	H	N	N	N	N	N	H	N	N	N	N			M	M	H	H	M	L	L

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Table 4.8 Final ratings obtained from the questionnaire for the Permeability Tests

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Seismic Refraction Test	L	H	N	N	N	M	L	N	L	L	N	N	N			H	H	H	H	H	H	L
	L	Seismic ReflectionTest	L	H	N	N	N	M	L	N	L	L	N	N	N			H	H	H	M	M	M	L
	S	Seismic Cross-Hole Test	L	L	N	N	N	N	N	N	N	H	N	N	N			H	H	H	H	H	H	H
	S	Seismic Down-Hole Test	L	L	N	N	N	N	N	N	N	H	N	N	N			H	H	H	H	H	H	H
	S	Surface Wave Test	L	L	N	N	N	N	N	N	N	H	N	N	N			H	H	H	H	H	H	H
	L	Resistivity Test	L	M	L	N	N	L	N	N	N	N	N	N	N			L	M	H	H	H	H	M
	S	Gravimetric Test	L	M	N	N	N	M	N	N	N	N	N	N	N			H	H	H	H	H	H	H
	L	Magnetic Test	L	M	N	N	L	L	N	N	N	N	N	N	N			H	H	H	H	H	H	H

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Table 4.9 Final ratings obtained from the questionnaire for the Geophysical Surveying Tests

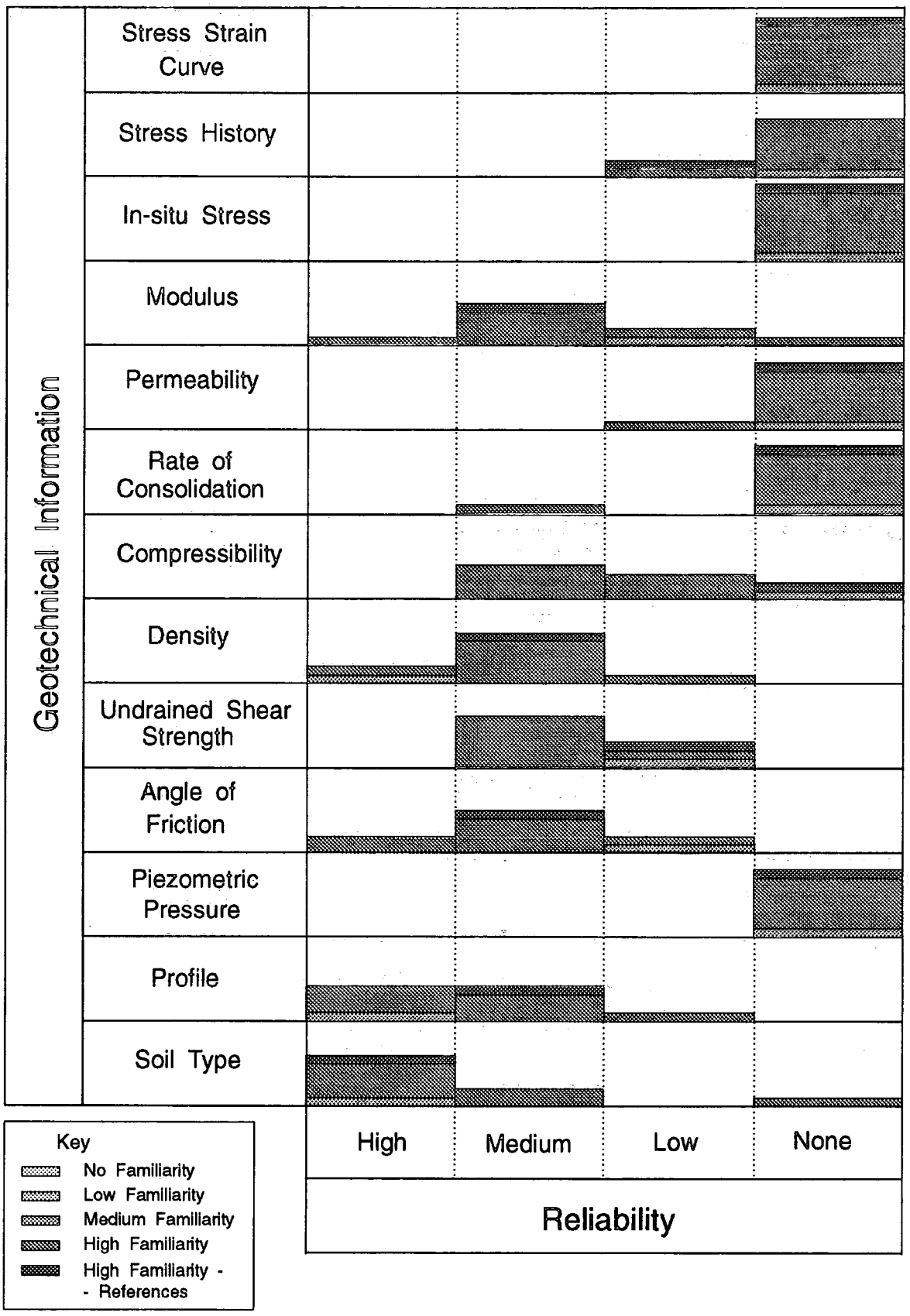


Figure 4.3 Results obtained from the questionnaire for the SPT concerning geotechnical information

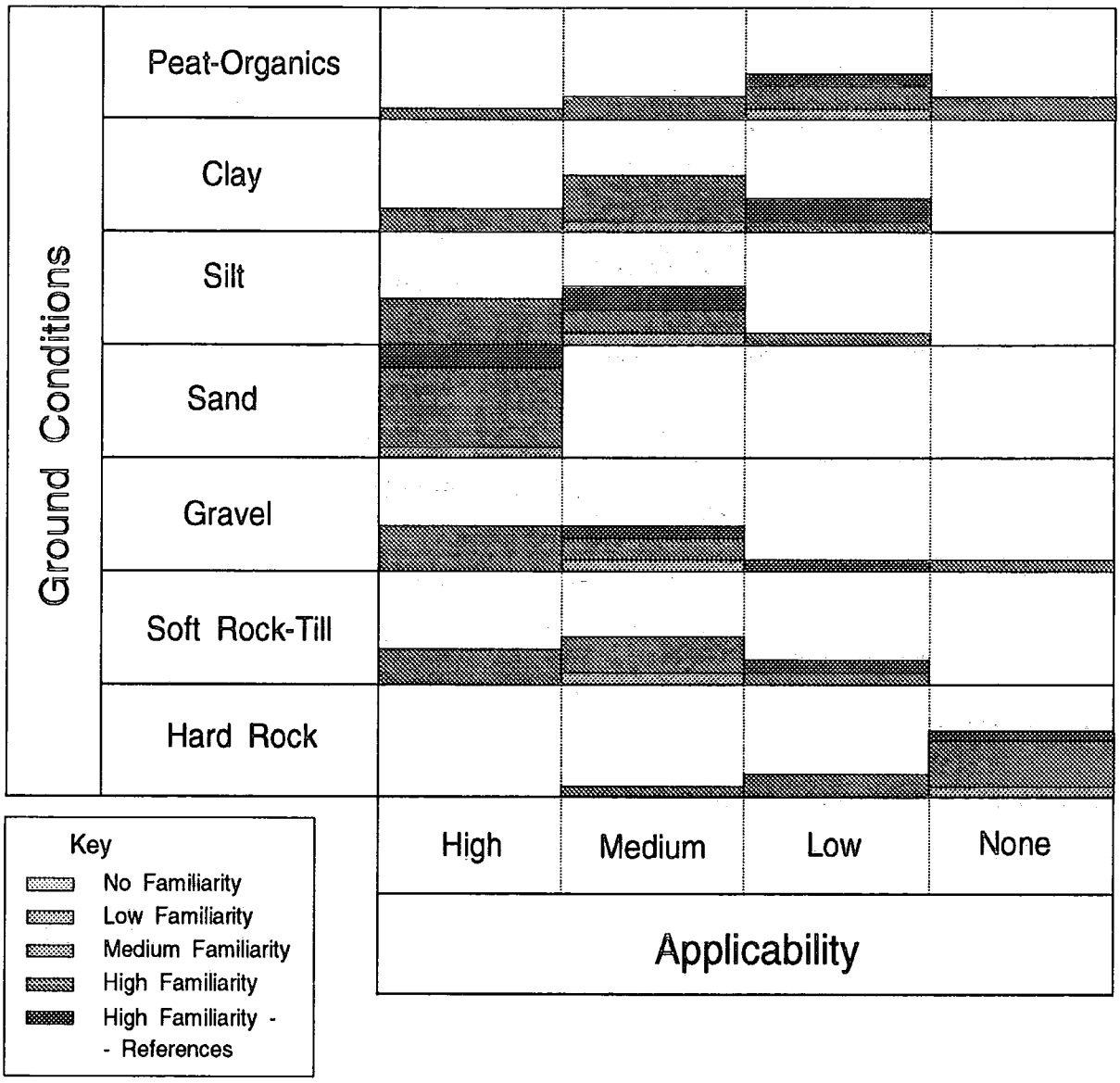


Figure 4.4 Results obtained from the questionnaire for the SPT concerning the different ground conditions

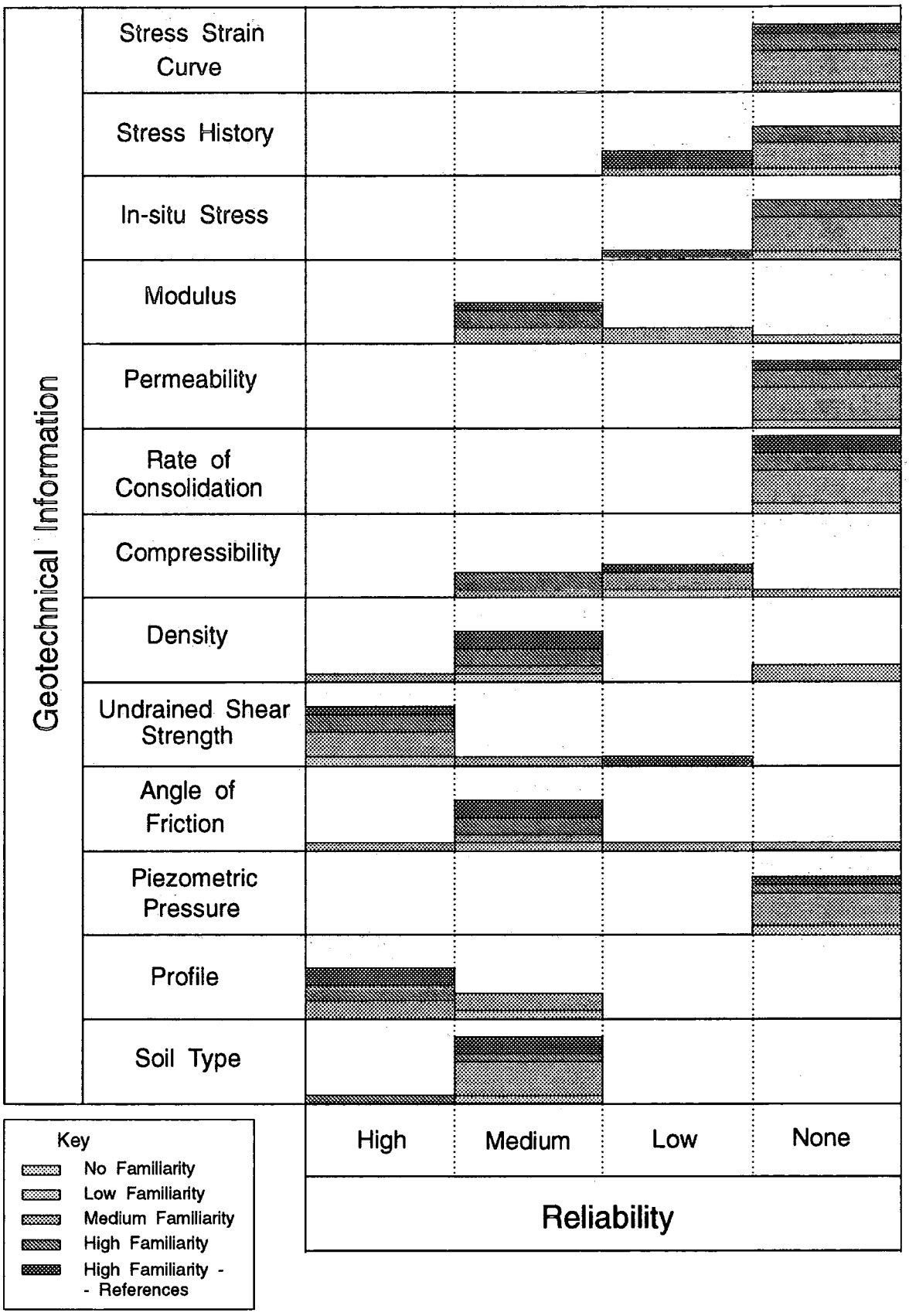


Figure 4.5 Results obtained from the questionnaire for the electrical CPT concerning geotechnical information

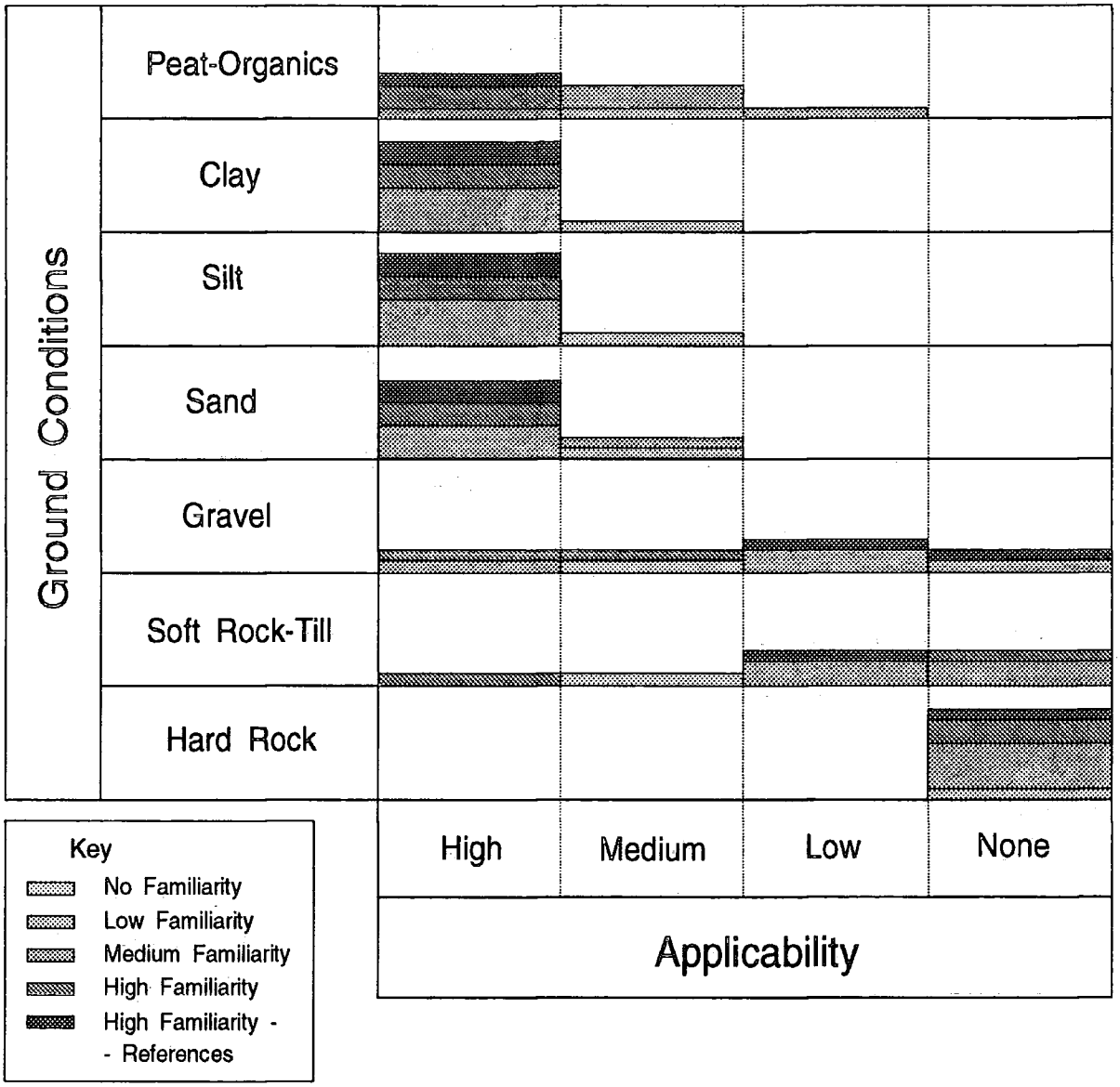


Figure 4.6 Results obtained from the questionnaire for the electrical CPT concerning the different ground conditions

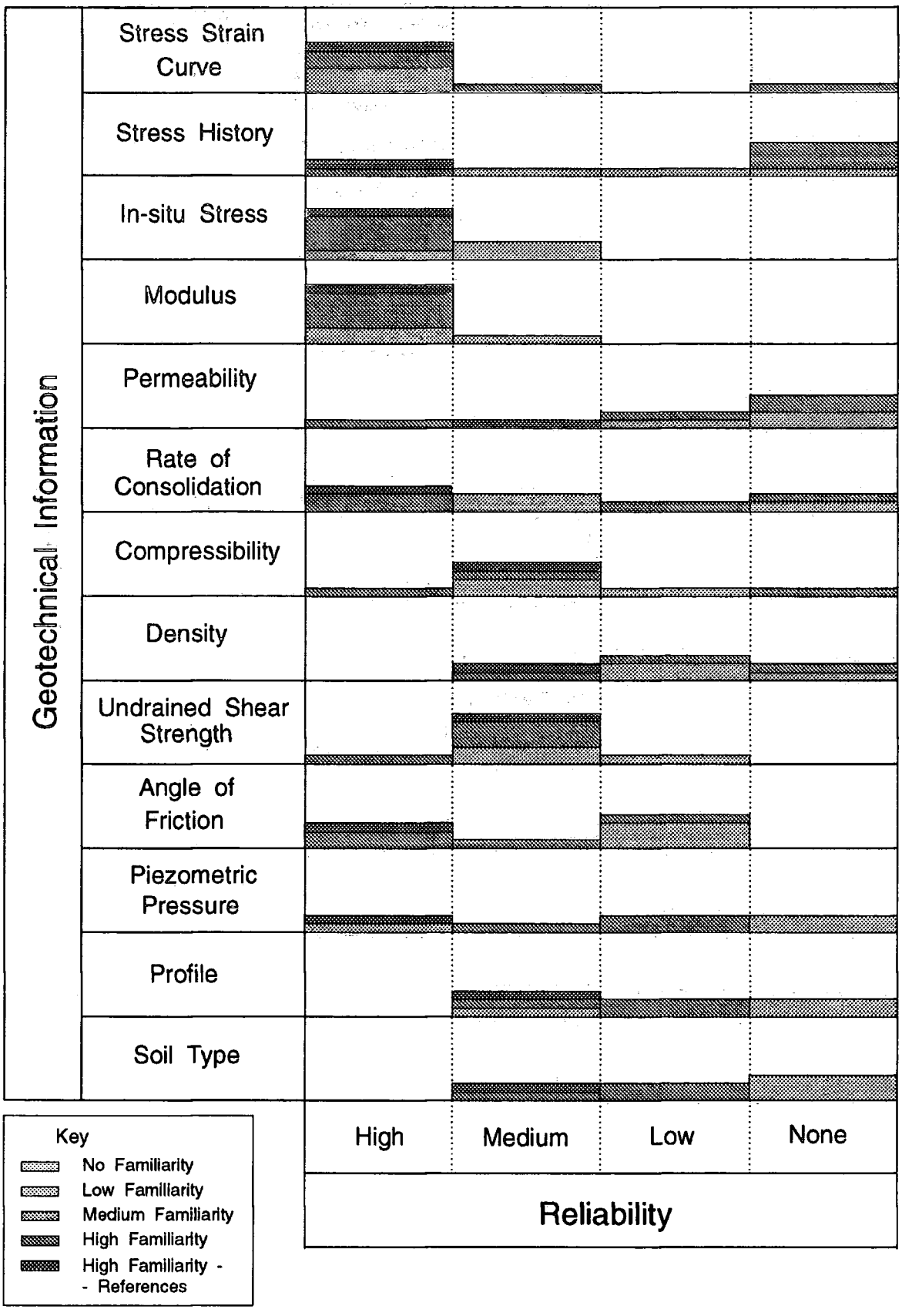


Figure 4.7 Results obtained from the questionnaire for the Self-boring Pressuremeter concerning geotechnical information

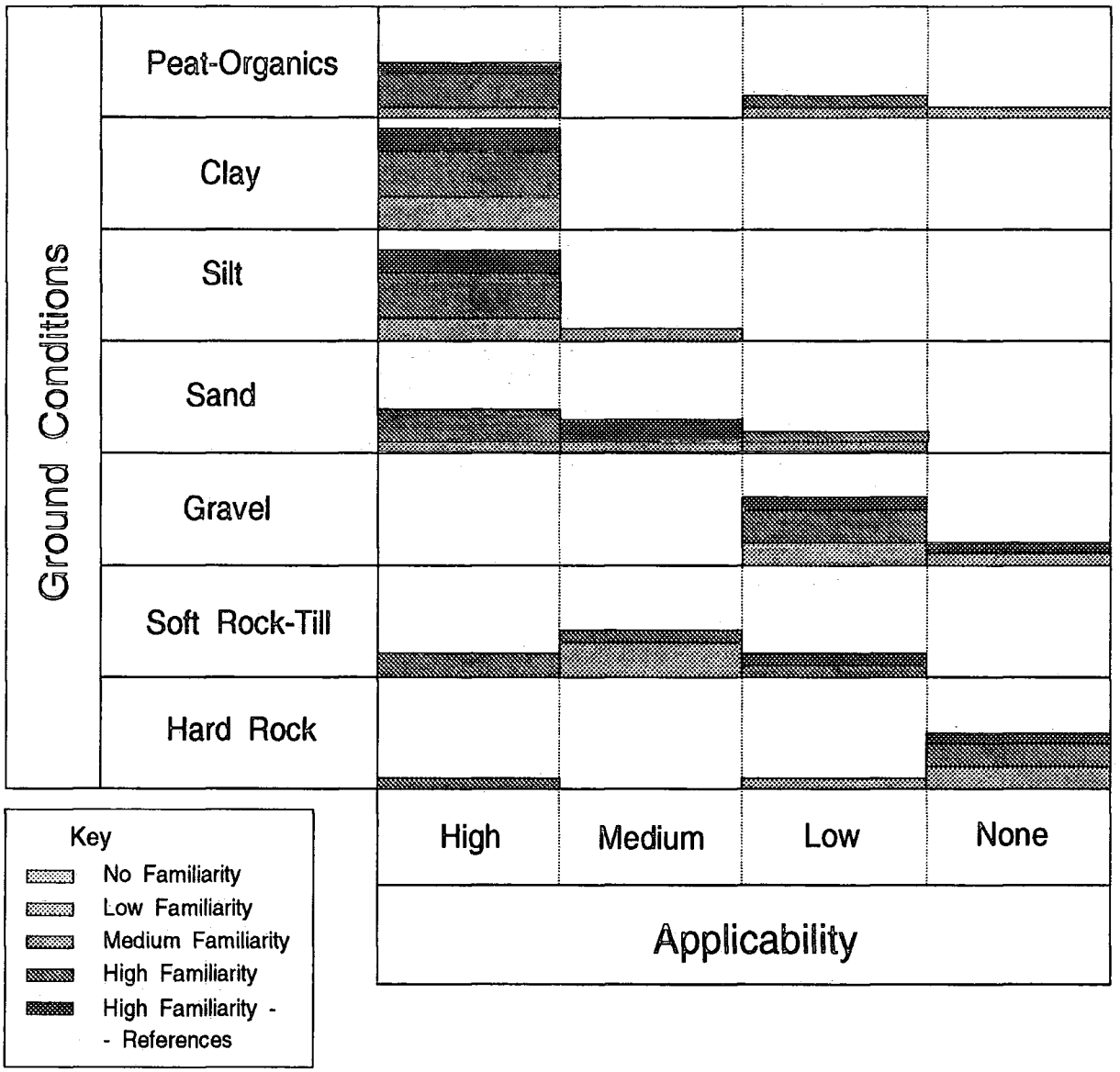


Figure 4.8 Results obtained from the questionnaire for the Self-boring Pressuremeter concerning the different ground conditions

CHAPTER 5

A KNOWLEDGE-BASED SYSTEM TO ASSIST IN THE SELECTION OF IN-SITU TESTS

5.1 Introduction

A prototype knowledge-based system has been developed in order to assist in the selection of appropriate geotechnical field tests. The system performs two functions:

1. General querying of the two knowledge bases,
2. Advise on selecting field tests.

On activating the knowledge-based system the user has the option to select one of these two functions from a menu.

The first option allows the user to interrogate separately the two knowledge bases included in the system in order to retrieve information from the facts that make up these knowledge bases. The user initiates the searching process by selecting one out of six menu items activating the corresponding rule built into the system. The rules included in the system, as described in section 3.3.4, permit a search to be carried out at many levels within the facts. The second option provides assistance in the selection of appropriate field tests. The selection of this option activates a rule that queries sequentially both knowledge bases in an alternating way, which produces information about possible in-situ tests according to the user's input. The user's input in this case is only menu-driven. The user's input for the system as a whole is mainly menu driven except in two cases where the user is prompted to input numerical values. This type of user interface makes the system easy to use.

The system has been implemented using PDC Prolog on a Personal Computer. It can be described as a model-based knowledge-based system as it supports a model for representing the knowledge and rules to manipulate the included knowledge. The knowledge-based system consists of three files. The first file is called KNOWBASE.PRO and contains the Ground and Tests knowledge bases as presented in Chapters 3 and 4 respectively. The second file is called INFINT.PRO and contains the process that manipulates the knowledge bases which consists of the Extended Inference Mechanism (described in Chapter 3) and a rule for assistance in the selection of appropriate tests. INFINT.PRO also provides the rules for the user interface developed for the system in order to facilitate the consultation process. The third file, GENERIC.PRO contains all the generic rules required by the system. Full listings of these three files are given in Appendix A. The knowledge-based system is superimposed on top of the built-in Prolog inference mechanism that supports backward chaining and depth-first search.

An important feature of the system is considered to be the domain independent inference mechanism used to interrogate both knowledge bases (that forms the Extended Inference Mechanism). This inference mechanism allows inheritance and transitivity inferences as well as information retrieval from any set of facts represented in a similar way. The user interface has also been implemented at a general level allowing any number of knowledge bases (relating to any domain) to be interrogated. The inference mechanism and the user interface developed could be considered as a basic expert system shell. However, in the current version no other facilities (such as help facilities) are provided.

Unlike the approach adopted for the development of the rules used to search the knowledge bases and the corresponding user interface, the advisory rule and the user interface developed for it are domain specific in order to produce efficient solutions.

The system has been implemented using the Phar Lap DOS-Extended version of PDC Prolog 3.30 (1992) on a 286 Nimbus AX/2 IBM-compatible Personal Computer. Initially PDC Prolog version 3.20 was used on the same PC with 1 Mbyte internal memory. This combination of software-hardware was

soon found inadequate to handle the requirements of the system as the execution of the program was terminated during run time giving a 'Heap Overflow' error. It was understood that Prolog did not have enough addressable memory required due to the large amount of knowledge incorporated into the system, as this Prolog version was not able to utilise any memory above 640 Kbytes allowed by the MS-DOS operating system. The memory problems that were preventing continuation of the development of the system were eliminated by using the Phar Lap DOS-Extended version and by expanding the PC's internal memory to 3 Mbytes.

5.2 General Description of the System

Descriptions of each part that constitute the knowledge-based system developed are presented in the following sections.

5.2.1 Knowledge Bases

Two knowledge bases have been implemented in the system:

- The *Ground Knowledge Base* and
- The *Tests Knowledge Base*.

Ground Knowledge Base

The knowledge included in the Ground Knowledge Base and its implementation in Prolog are described in full detail in Chapter 3.

The Ground Knowledge Base contains a model of the ground. The level of detail introduced is a broad classification based on the British Standards (BS 5930, 1981). The knowledge base contains the relationships between the different levels of description used by this classification to describe the

ground - from the higher level *classes* (such as Soil or Rock) to the lowest level *instances* (such as Clay, Silt, Sand etc.).

Knowledge about *grain size, liquid limit, consistency, permeability, compressibility* and *secondary soil types* is included as attributes attached to each object. These properties have been represented by the use of multi-level compound data objects that allow the property values to be subdivided into finer ranges depending on descriptive terms.

Tests Knowledge Base

The Tests Knowledge Base has been implemented applying the representation scheme used for the Ground Knowledge Base.

As described in Chapter 4, the Tests Knowledge Base contains knowledge about the different types of geotechnical tests that form the test hierarchy shown in figures 4.1 and 4.2. The knowledge consists mainly of two types of information:

- The *applicability* of a test in different types of ground.
- The *reliability* of a test for obtaining specific geotechnical information (assuming ideal ground conditions and taking into account all necessary correlations).

Additional knowledge concerning the *test objective, unit cost* and *test frequency* of the various types of in-situ tests has also been included. The knowledge has been obtained in two ways: i) from published material and ii) carrying out a knowledge elicitation exercise in the form of a questionnaire. It should be noted that the knowledge obtained and included in the system is not complete, mainly due to the large volume of information required and time constraints.

5.2.2 Generic Rules

The file GENERIC.PRO contains the definition of some classic list-processing predicates that the rules in the main program make use of. These predicates express relationships involving lists, allowing useful concepts, such as the membership of a list, to be defined. The following list-processing predicates have been defined:

- **members(X, List)**, that generates all the individual elements from a list.
- **member(X, List)**, that checks if an element is a member of a list.
- **first(List, X)**, that finds the first item of a list.
- **last(List, X)**, that finds the last item of a list.
- **min_number(List, X)**, that computes the minimum of a list of numbers.
- **max_number(List, X)**, that computes the maximum of a list of numbers.
- **append(List1, List2, List3)**, that adds one list to another to make up a new list.
- **reverse(List1, List2)**, that reverse the order of the elements of a list.
- **remove_duplicates(List1, List2, List3)**, that deletes all multiple occurences of the items of a list.
- **split_list(X, List1, List2, List3)**, that divides a list into two sublists having as a criterion a specific element of the list.
- **simplify_lists(List1, List2, List3)**, that converts a list of lists into a simple list.
- **delete_item(X, List2, List3)**, that deletes an element of a list.
- **delete_list(List1, List2, List3)**, that deletes a sub-list of a list.

The predicates that define the membership relationship, the first and last relationships, and the minimum and maximum relationships are used to process the elements of a list in order to identify the desired relationship. The predicates that define the append and the reverse relationships as well as the predicates delete_item and delete_list are used to create a new list by processing the items of an existing list. Finally the predicates remove_duplicates, split_list and simplify_lists have been defined making use of other list predicates such as the member predicate, the append predicate and the first predicate.

5.2.3. Extended Inference Mechanism

The Extended Inference Mechanism consists of structure dependent rules that can be used to search both the Ground and the Tests knowledge bases, as they have been represented using the same structures. In general these rules are domain independent.

The rules, which are described in detail in Chapter 3, extend the built-in inference engine of PDC Prolog, by providing facilities for inheritance and transitivity, as well as facilities for information retrieval that could be used to search any knowledge base represented in a similar way (section 3.3.3).

The basic rules developed can be divided into three categories according to the inferences they allow:

- Inheritance rule (*get_all_attributes*)
- Transitivity rule (*discover_members*,
  find_ancestors)
- Information retrieval rules (*find_attribute_and_value*,
  find_modifiers,
  find_objects_and_modifiers)

The rules *get_all_attributes*, *discover_members* and *find_ancestors* are totally domain independent whereas the rules *find_attribute_and_value*, *find_modifiers* and *find_objects_and_modifiers* have a weak domain dependency as described in section 3.3.4.

5.2.4. Advisory Rule

The advisory rule, **investigate**, is used to assist in the selection of appropriate geotechnical in-situ tests. The rule **investigate** acts as follows:

investigate

Input values:

- Object-name of the ground hierarchy - class or instance (e.g. coarse)
- Geotechnical information required to be identified
  (e.g. angle_of_friction)
- Reliability desired for a test in obtaining the required geotechnical information (e.g. high)
- The names of test attributes that the user wants information on in addition to the applicability and reliability (e.g. [test_objective])

Output values:

- The members of the ground hierarchy object - soil types (e.g. [gravel, sand])

N times

(where N denotes the number of alternative solutions)

- The name of an in-situ test that can be used to obtain the required geotechnical information with the desired reliability
  (e.g. in_situ_shear_test)
- The applicability of this test for use in each of the soil types
  (e.g. [none, none])
- The modified soil types, for which a different value of applicability applies for this particular test (e.g. [], denoting that no such knowledge has been specified for this test)
- The applicability of this test for use in each of the modified soil types
  (e.g. [])
- The names of the additional attributes that are defined for this test
  (e.g. [test_objective])
- The value(s) of the additional attributes under consideration
  (e.g. ([specific_test_method])

The advice rule is sequentially model dependent, interrogating each model as required. The way the rule *investigate* acts is diagramatically shown in Figure 5.1. Initially it searches the ground model using the rule *discover_members* in order to identify the members of the soil category (soil types) specified as input value. A soil type, that has no members, can also be used as input. It then identifies, going through the modifier facts of the Ground Knowledge Base, the modified soil types that could exist for each soil type forming the soil category. Taking into account the geotechnical parameter required and the desired reliability, it finds the first suitable in-situ test that it encounters in the Tests Knowledge Base and provides its applicability for use in the derived soil types forming the soil category. For the same test, the advisory rule also retrieves from the Tests Knowledge Base the modified soil types for which a different applicability rating is applied as well as the applicability value defined for them. Finally, taking into account the input additional attributes (if any), the rule returns those of the additional attributes that are defined for this test and their value(s). All alternative in-situ test methods that fulfil the requirements of the user are generated through backtracking. The same type of information provided for the first test, is given for all the others. The user can then compare the knowledge provided for each alternative test by the knowledge bases through the *investigate* rule and also consider additional factors (not incorporated in the system) that he/she finds relevant in order to make the final selection.

The *investigate* rule searches both sets of facts (predicates *class* and *modifier*) for both models. This rule is domain dependent as it can only be applied to the two models included in the system. However, it should be noted that addition or deletion of knowledge included in any of the two knowledge bases will not affect the rule.

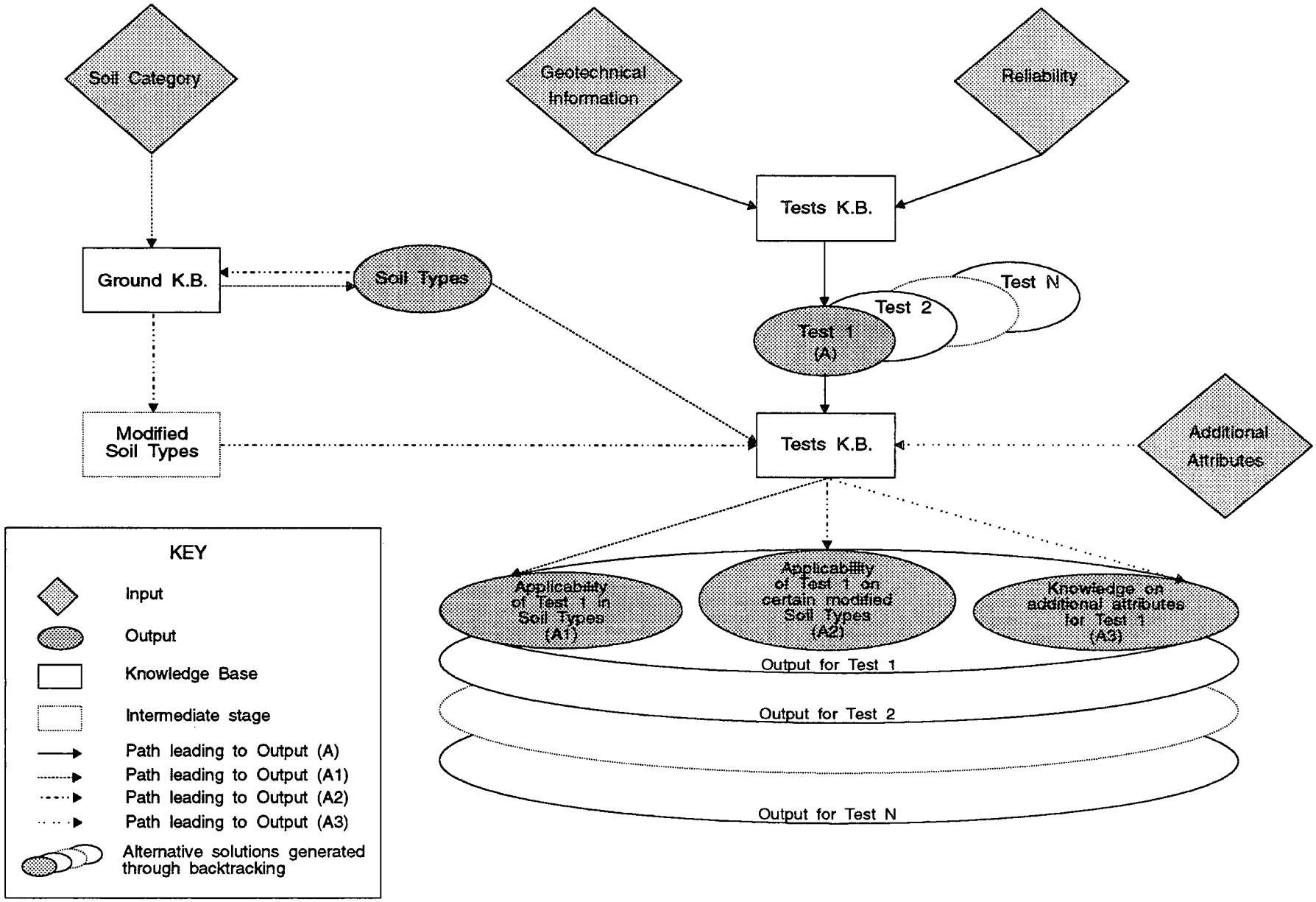


Figure 5.1 Schematic Representation of **investigate** rule

5.2.5. User Interface Facilities

A customised user interface has been developed using the tools provided by PDC Prolog (PDC Prolog Toolbox, 1990). The user interface is menu driven. This provides ease of use and accessibility.

The following tools have been used for the development of the user interface:

- *Status Lines*, displayed at the bottom of the screen, which are used to guide the user.
- *Longmenus*, allowing the user to select an option scrolling through arbitrarily long lists of menu items. More specifically, the *longmenu_repeat* toolbox predicate has been used in most cases as it allows re-selections to be made. In one case the *longmenu_mult* predicate has been used that allows multiple selection from the menu rather than a single selection.
- *Lineinput*, that accepts input from the user in a given screen field. The toolbox predicate *lineinput_repeat* has been used as this allows new text input.

On invoking the system the user is presented with a menu listing the two options that the system can offer:

- Query Knowledge Bases,
- Assist Selection of In-Situ Tests.

On selecting the first option, another menu appears listing the actions that the system can activate in order to search a knowledge base included in the system. These actions activate the rules forming the extended inference mechanism. On selecting one of the actions a third menu appears on the screen, listing the knowledge bases currently included in the system. According to the action chosen, several menus (and a lineinput in some cases), are presented to the user in order to collect the desired input values required by the triggered rule.

It is interesting to note that the user interface implemented for the first option is domain independent. In order to achieve this, additional domain independent search rules were implemented for use in collecting information from the knowledge bases in order to enable the user to select its input values.

For example, the rule **find_all_attrib_names** retrieves all the attribute names of an object, the rule **get_all_names_with_factors** finds all the objects that have attributes with factors specified, the rule **get_all_fact_list** provides all the factors defined for an object, etc. In cases where the user is required to enter numerical value(s), rules have been provided that produce the allowable input range. For example, the rule **find_all_num_value_attr** produces the minimum and maximum value of an attribute of an object whereas the rule **find_all_general_range** provides the minimum and maximum value that an attribute can take within the whole model.

Also, the user interface is able to recognise (through the rules **case** and **situation** activated on selecting the actions 'find modifiers' and 'find objects and modifiers' respectively) the attributes that have numerical values and the ones that have symbolic values in order to display a lineinput or a menu of selections to the user, allowing him/her to input the required attribute value(s). Additionally, in the cases of a lineinput, a data validation is performed and an error message is displayed if the input value is incorrect (rule **condition**, rule **state**).

A major advantage of the user interface is considered to be the fact that there is no need to specify how many and which models are included as knowledge bases in the system. The roots of the existing hierarchies can be recognised by the system (rule **find_all_roots**) and are presented to the user for selection in order to get his/her preference on the knowledge base he/she desires to question. Using the rule **find_root_tree** the set of facts that correspond to the chosen knowledge base can be recognised, allowing the inspection only of these facts where necessary.

On selecting the second option a number of menus are presented to the user in order to collect the desired input values required by the activated rule, which in this case is the **investigate** rule. The user interface developed to assist in the selection of appropriate in-situ tests is domain dependent, as is the **investigate** rule that it triggers.

At each level of the interrogation process the user is allowed to re-select an option from a given menu or re-input value(s) or return to the previous menu. The user is always able to return to the main menu (listing the two options offered by the system) and restart the consultation or exit the system.

The above are better illustrated in example consultations with the system, which are presented in the following section.

5.3 Example consultations with the system

In this section example screen dumps generated during execution of the prototype system are presented in Figures 5.2-5.8. In Figures 5.2-5.7 execution of the program is shown when the first option, **Query Knowledge Bases**, has been selected whereas in Figure 5.8 the second function, **Assist Selection of In-situ Tests**, has been activated.

In Figures 5.2a and 5.2b the user selects the action **get attributes** in order to interrogate the knowledge bases about the attributes of an object. In Figure 5.2a he/she is interested in searching the **ground** knowledge base in order to find the attributes of the object **sand** and in particular he/she queries about the values of the attribute **grain_size**, which are displayed in the Answer window. As can be observed from the output, two levels of detail have been specified in the knowledge base for the attribute grain size; the more general level is displayed to the user who is given the choice to query if he/she desires to know information about the more detailed by typing the character 'y'. In a similar manner, the user questions the **tests** knowledge base (Figure 5.2b) about the attributes of the **standard_penetration_test** and in particular about the attribute **applicability**. It is interesting to note that in this case there is only one level of detail defined in the knowledge base for this attribute (the more detailed one), which is displayed to the user in the Answer window.

In Figure 5.3 the user interrogates the **ground** knowledge base selecting the action **find ancestors** in order to find out the ancestors of the object **silt**. As silt has two parents, two alternative solutions are displayed to the user in the Answer window. In Figure 5.4 the user selects the action **discover members** in order to search the **tests** knowledge base for the members-instances of the category **penetration_tests**.

In Figures 5.5a and 5.5b the action **find attribute and value** is selected in order to discover the attribute name and value(s) that correspond to a modifier of an instance. In Figure 5.5a the user is interested in searching the **ground** knowledge base to find the attribute name and value(s) that correspond to the modifier **loose** of the instance **gravel** while in Figure 5.5b the **tests** knowledge base is interrogated in relation to the modifier **modulus** of the instance **self_boring_pressuremeter_test**.

The screen dumps shown in Figures 5.6a and 5.6b are generated on selecting the action **find modifiers**. In Figure 5.6a the **ground** knowledge base is interrogated and the modifier(s) that correspond to the input range of values **(50,80)** kPa of the attribute **undrained_shear_strength** of the instance **clay** are derived and displayed in the Answer window. In general all alternative solution are produced. The input range of values in this case covers more than one predefined range, therefore the output produced **(firm_to_stiff)** combines the modifiers firm and stiff defined in the knowledge base. In figure 5.6b the user is interested in searching the **tests** knowledge base in order to find out the modifier(s) that correspond to the value **high** of the attribute **reliability** of the instance **piezocone test**.

In Figure 5.7 the user selects the action **find objects and modifiers** in order to learn which object(s) and modifier(s) (if any) defined in the **ground** knowledge base correspond to input range of values **(2,70)** mm of the attribute **grain_size**. All alternative solutions are generated and displayed to the user in the Answer window. If the solution generated by the system requires more lines than the output window automatic scrolling occurs. It can be observed from Figure 5.7 that due to scrolling the same information can be seen into two subsequent Answer windows.

As can be noted from the examples presented in the last two cases (actions **find modifiers** and **find objects and modifiers**), according to the type of attribute selected either a lineinput or a menu of selection is provided to the user for his/her input. In the case of a lineinput the relevant allowable range of values is also given as guidance to the user. The user can input either one value or a range of values that lie in the allowable input range.

Finally, in Figure 5.8 example screen dumps are generated of a consultation with the system for assisting in the selection of in-situ tests. The user queries the system about possible applicable in-situ test methods specifying that the ground conditions to be tested consist of **fine** soil, the geotechnical parameter to be derived is the **undrained_shear_strength** and the reliability required is **high**. In addition the user desires to consider other attributes as well for each test that will be generated, such as the **test_frequency**, the **unit_cost**, the **test_objective**, and the **test_nature**. All alternative solutions based on the parameter required and the reliability specified are derived and displayed in the Answer window. For each of these tests the following information is presented:

- its applicability to the soil types-members of the category fine (silt and clay),
- its applicability to the modified soil types for which a different applicability rating has been defined in the tests knowledge base, and
- the values of each of the additional attributes under consideration.

The user can then compare all the information provided by the system for each of these tests and taking into account other factors as well (not incorporated in the knowledge bases), make his/her final selection.

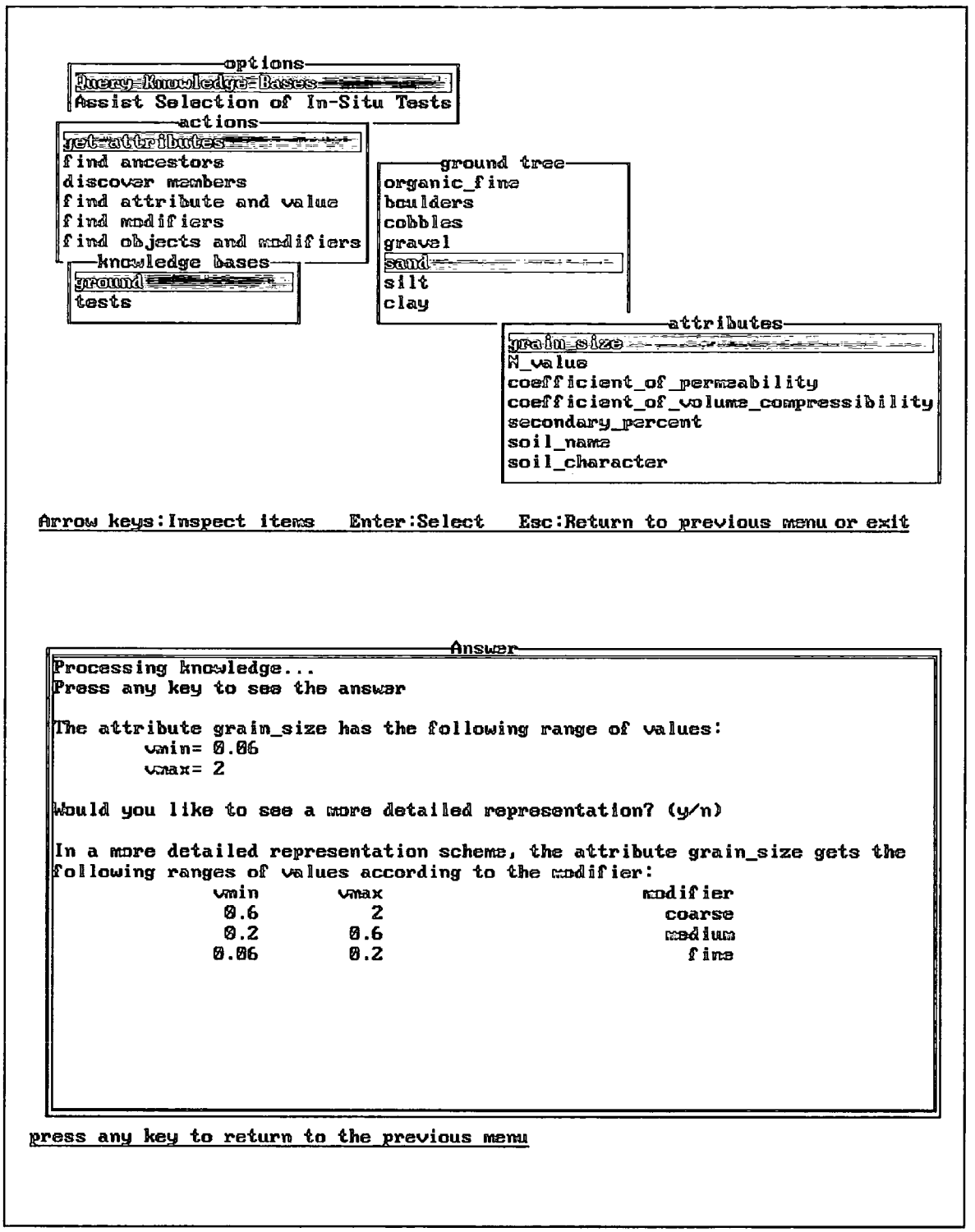


Figure 5.2a Example screen dumps for interrogating the Ground Knowledge Base about the attributes of an object. On selecting an attribute of the chosen object, the attribute values are displayed in the Answer window.

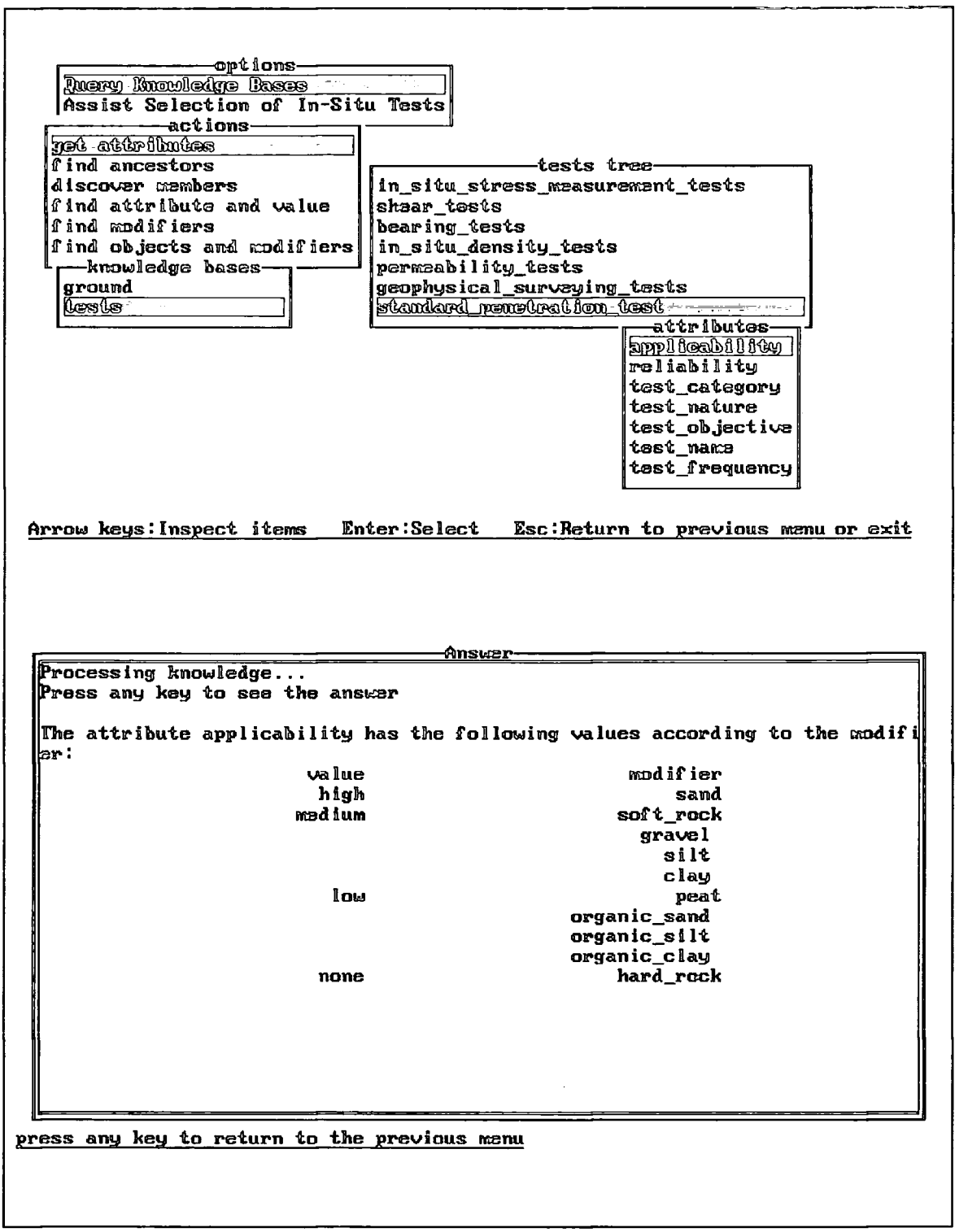


Figure 5.2b Example screen dumps for interrogating the Tests Knowledge Base about the attributes of an object. On selecting an attribute of the chosen object, the attribute values are displayed in the Answer window.

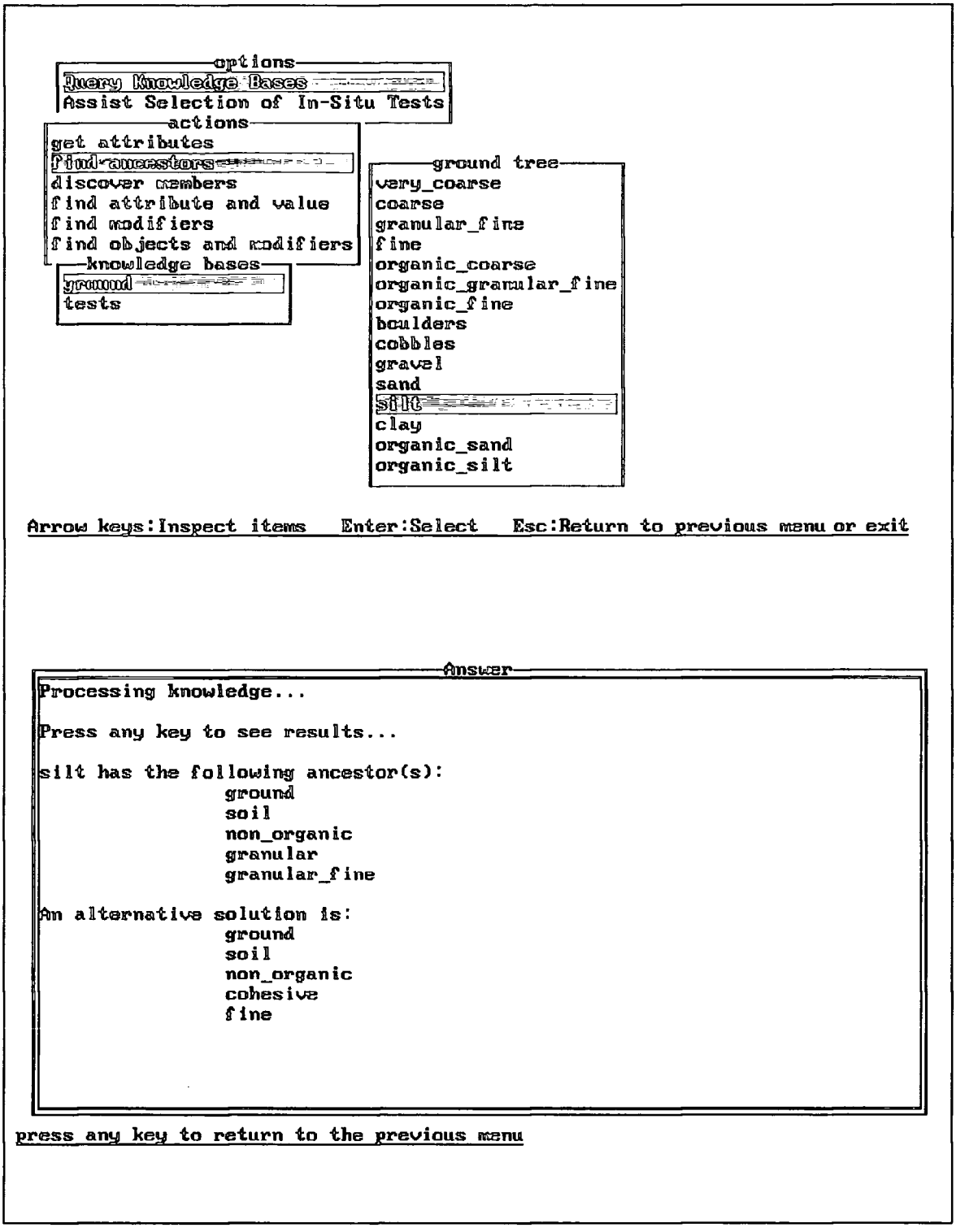


Figure 5.3 Example screen dumps for interrogating the Ground Knowledge Base about the ancestors of an object

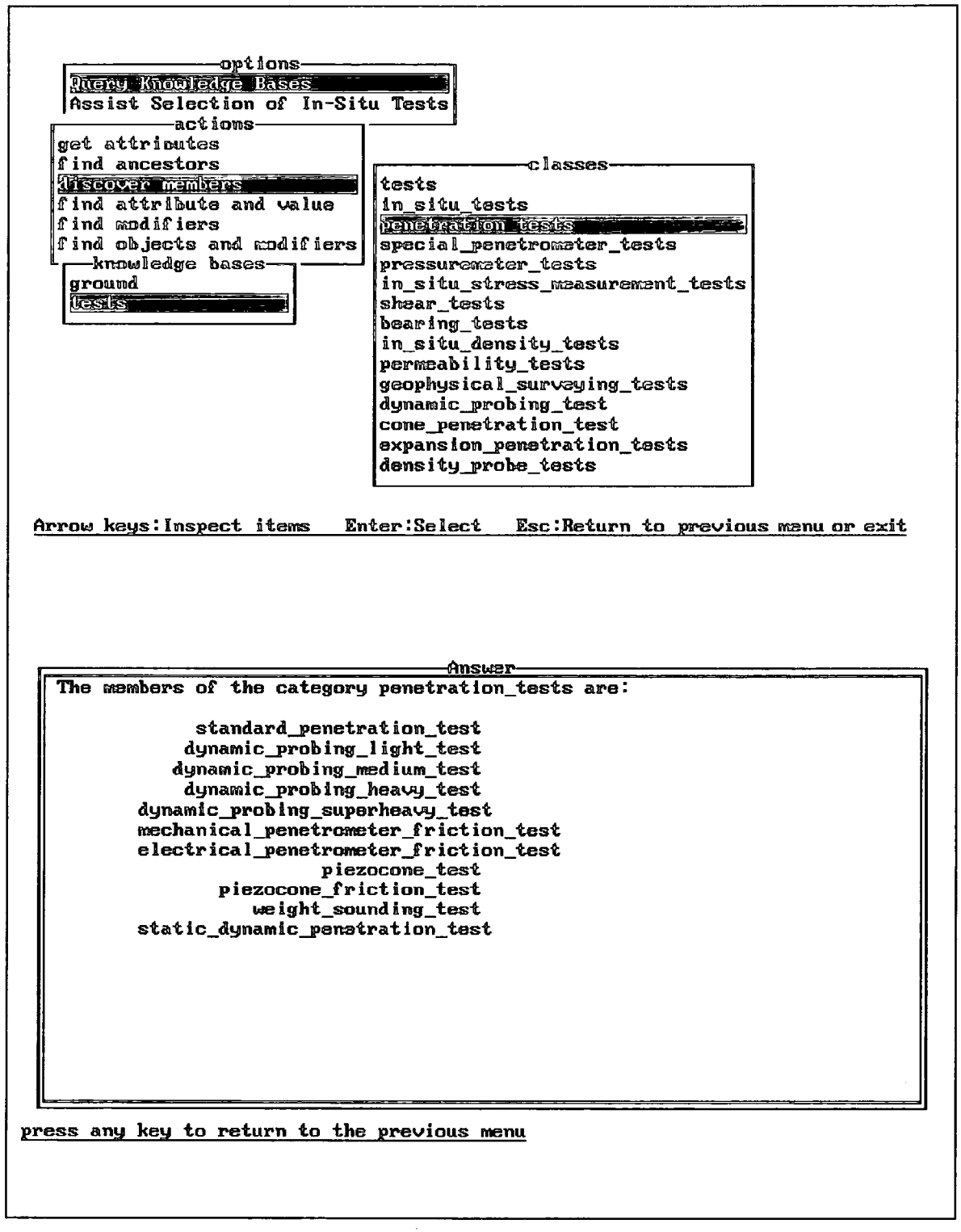


Figure 5.4 Example screen dumps for interrogating the Tests Knowledge Base about the members-instances of an object

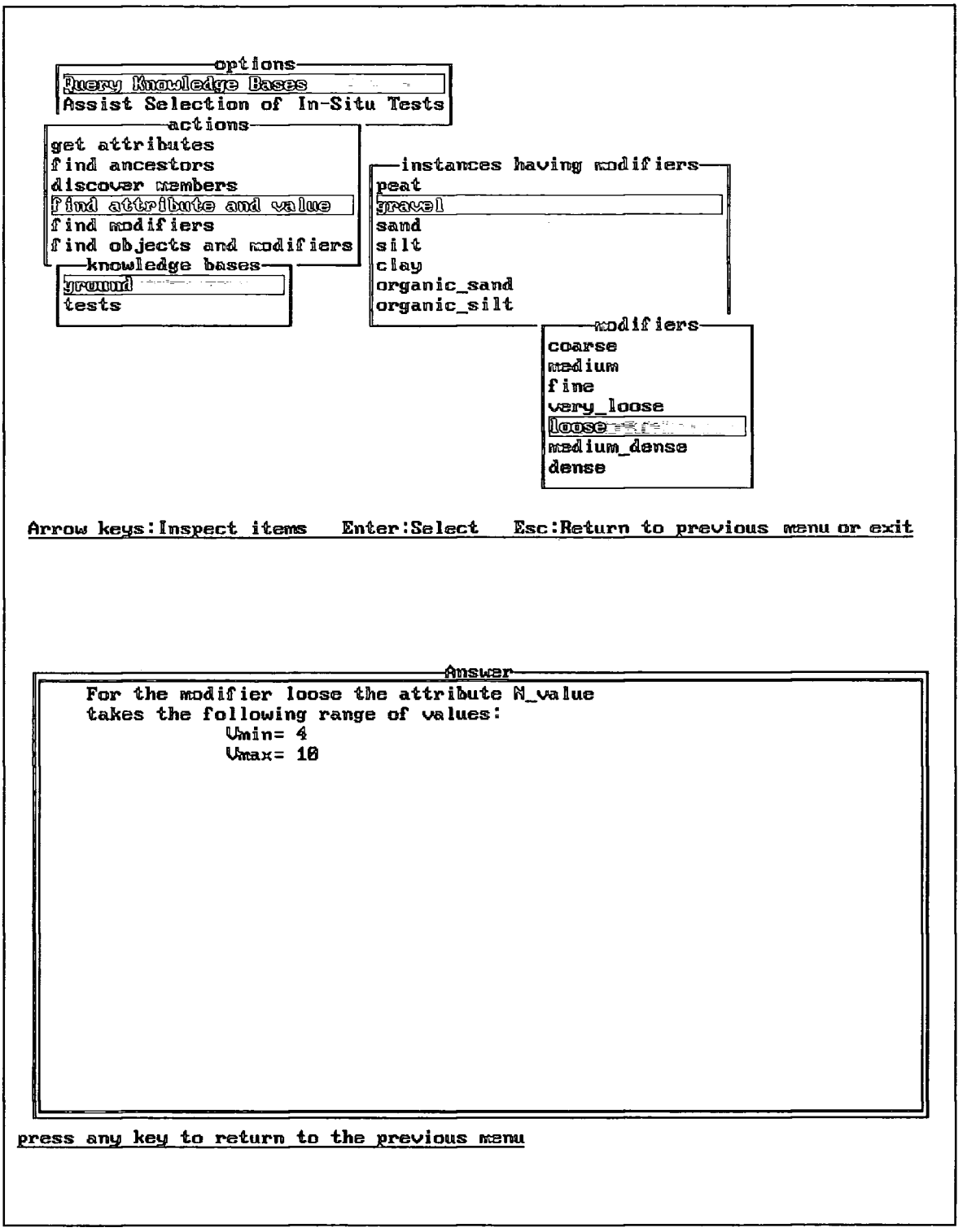


Figure 5.5a Example screen dumps for interrogating the Ground Knowledge Base about the attribute name and attribute value(s) of an instance, which correspond to a chosen modifier of that instance

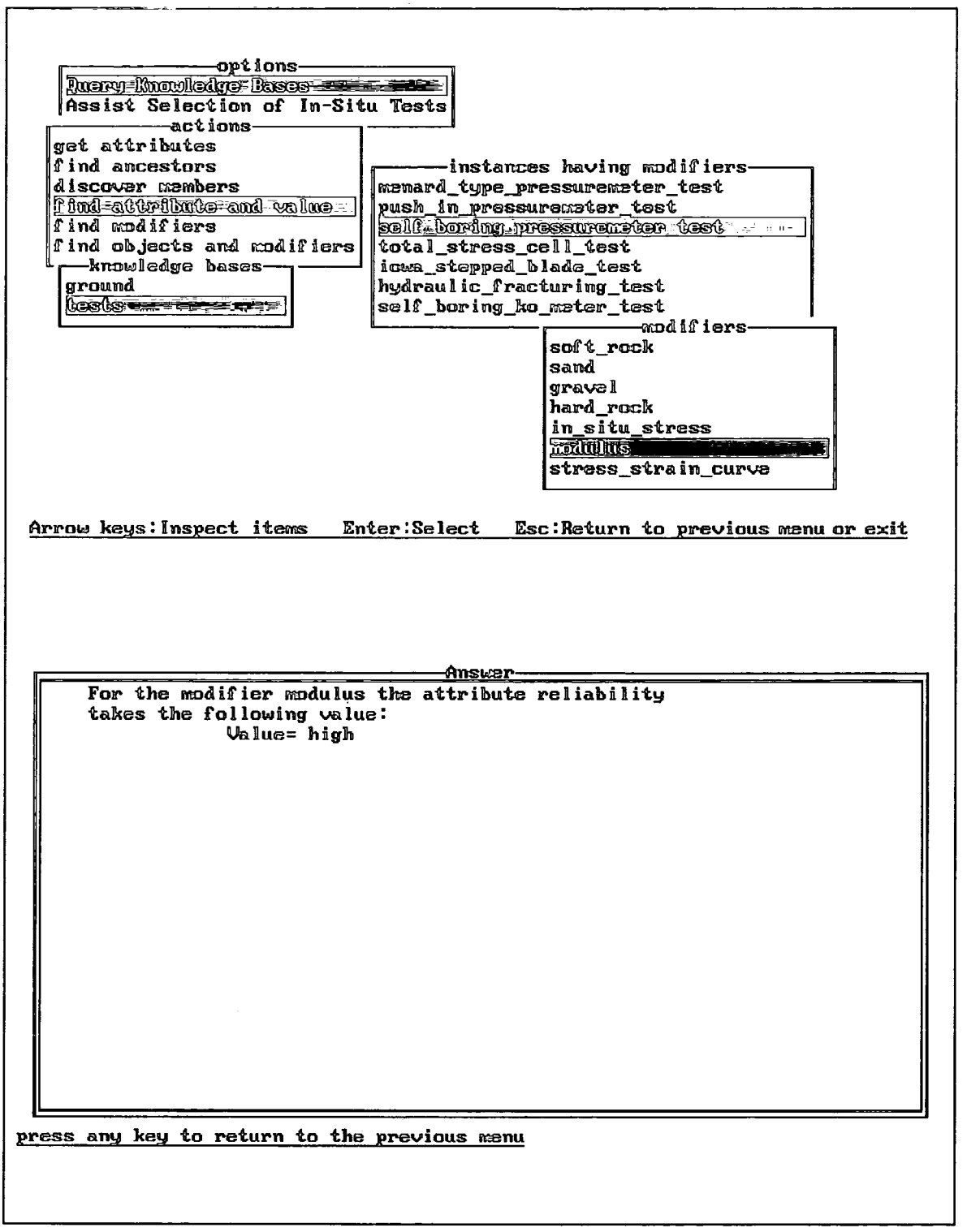


Figure 5.5b Example screen dumps for interrogating the Tests Knowledge Base about the attribute name and attribute value(s) of an instance, which correspond to a chosen modifier of that instance

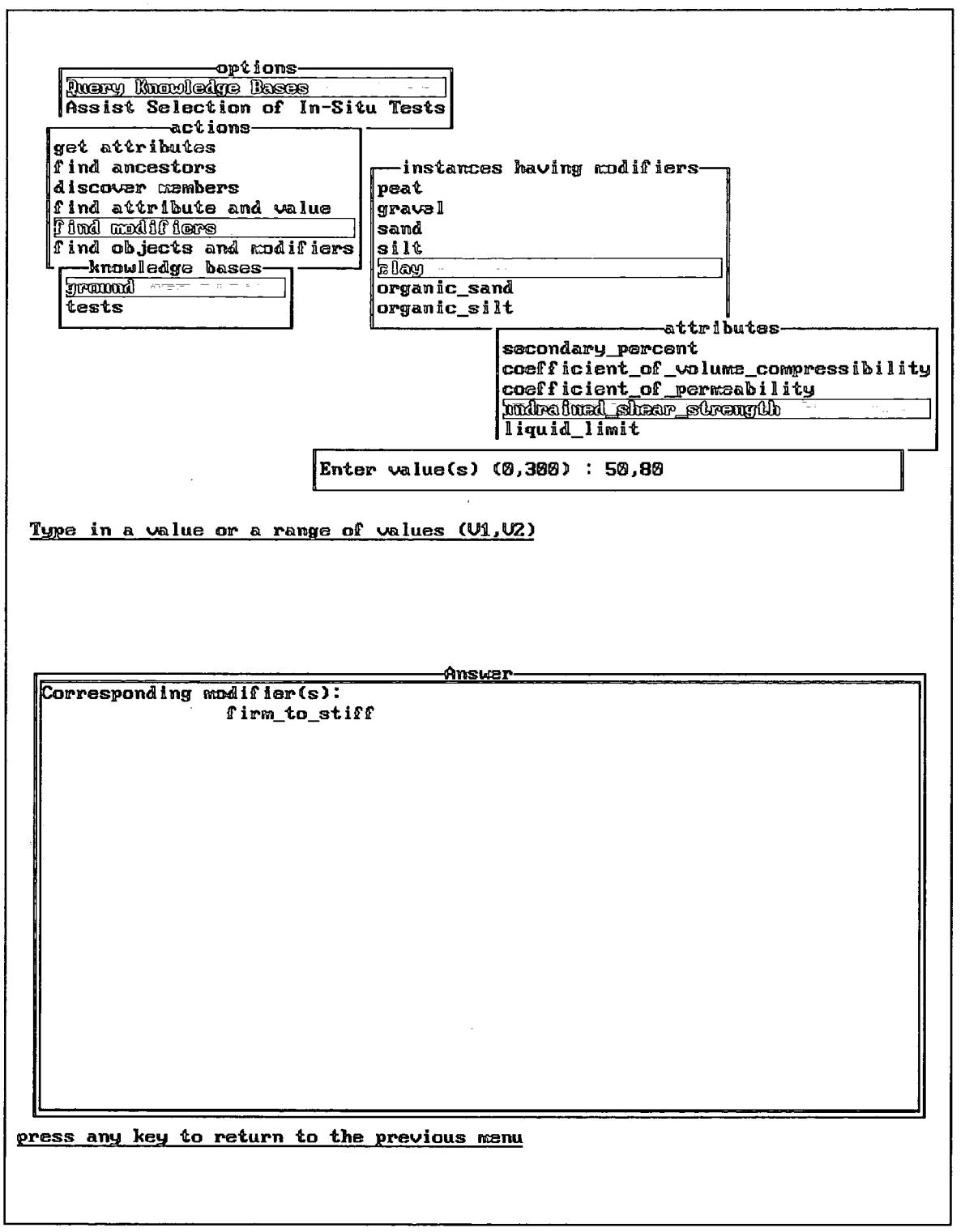


Figure 5.6a Example screen dumps for interrogating the Ground Knowledge Base about the modifier(s) that corresponds to an attribute name and attribute value(s) of a chosen instance

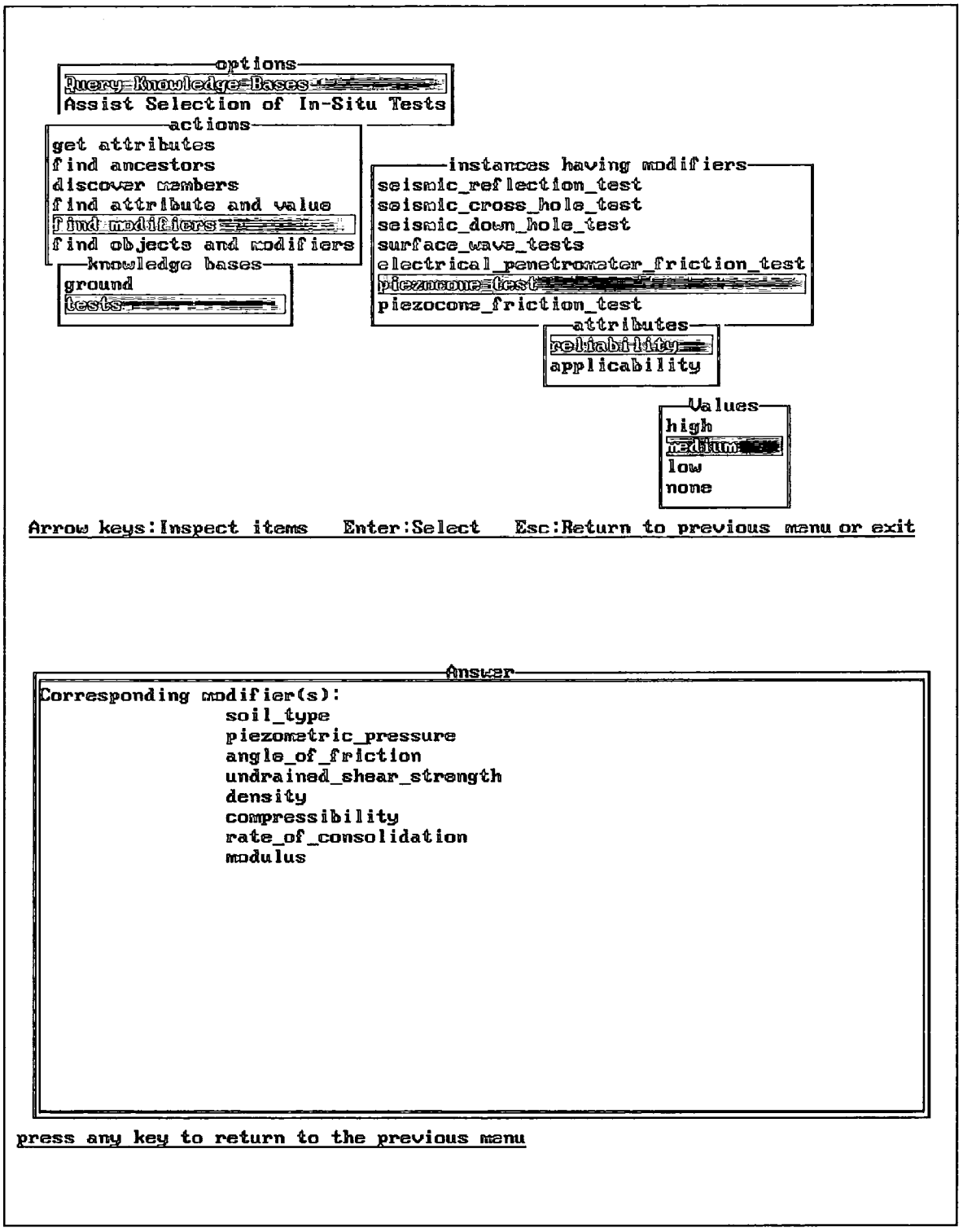


Figure 5.6b Example screen dumps for interrogating the Tests Knowledge Base about the modifier(s) that corresponds to an attribute name and attribute value(s) of a chosen instance

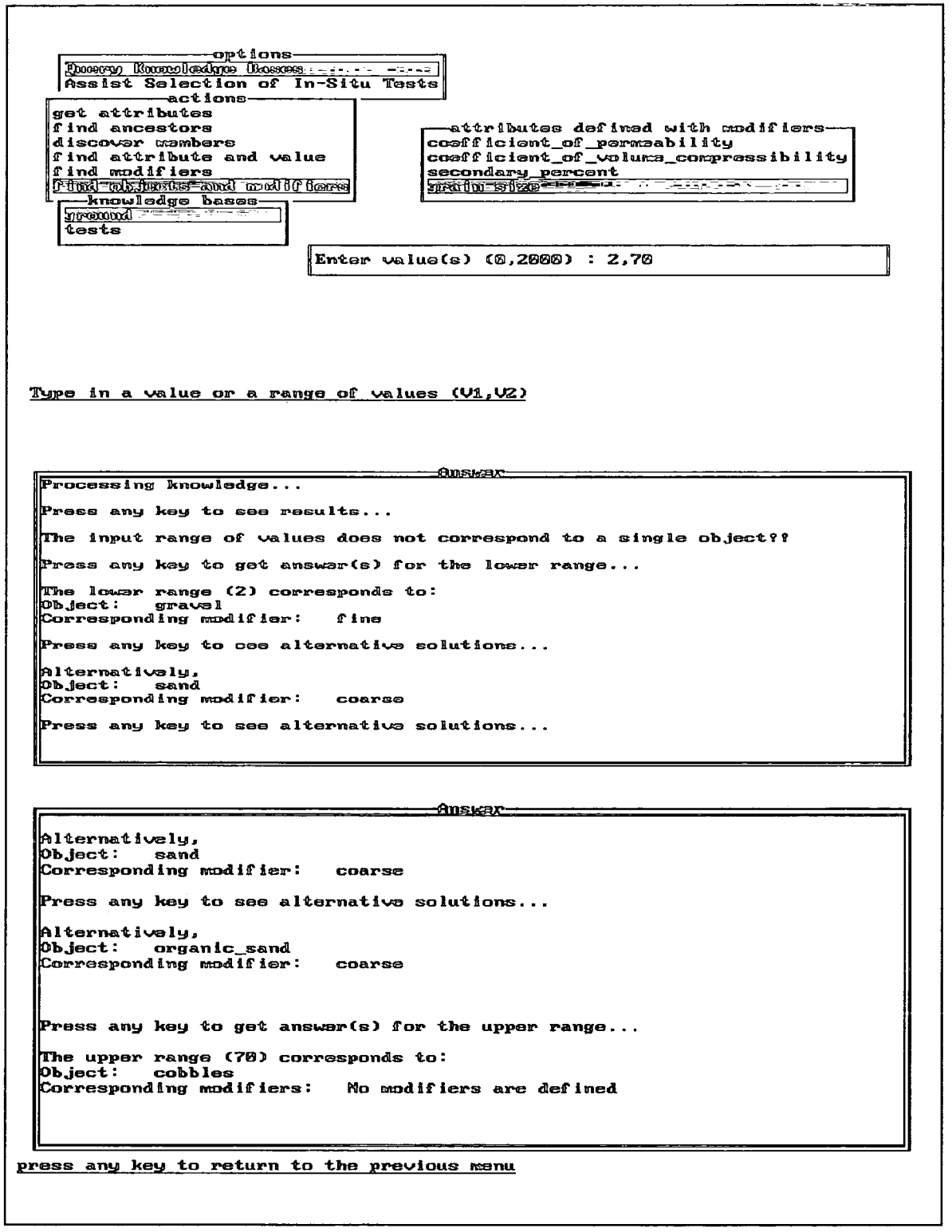


Figure 5.7 Example screen dumps for interrogating the Ground Knowledge Base about the object(s) and modifier(s) that correspond to an attribute name and attribute value

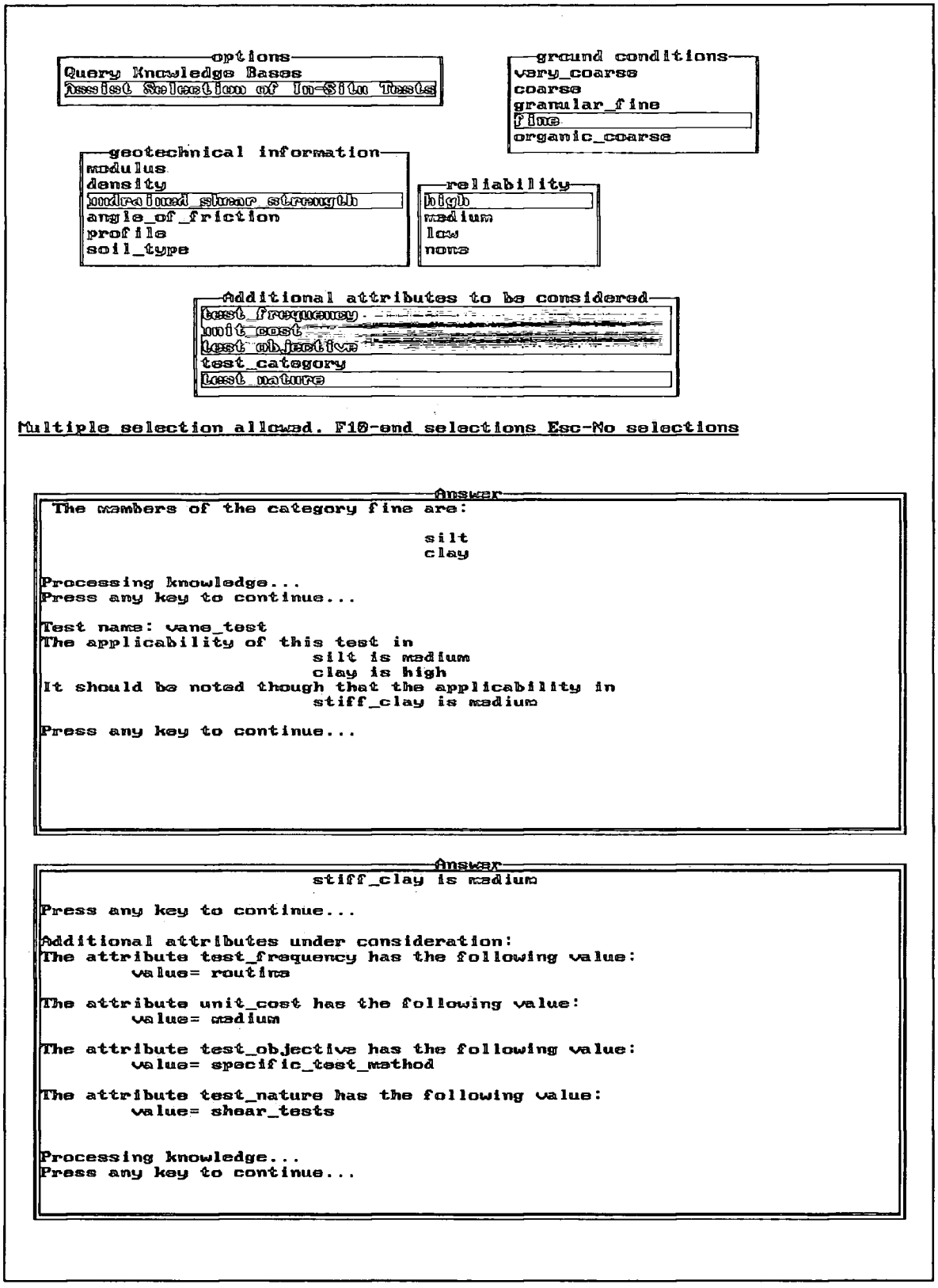


Figure 5.8 Example screen dumps of a consultation for assisting the selection of in-situ tests

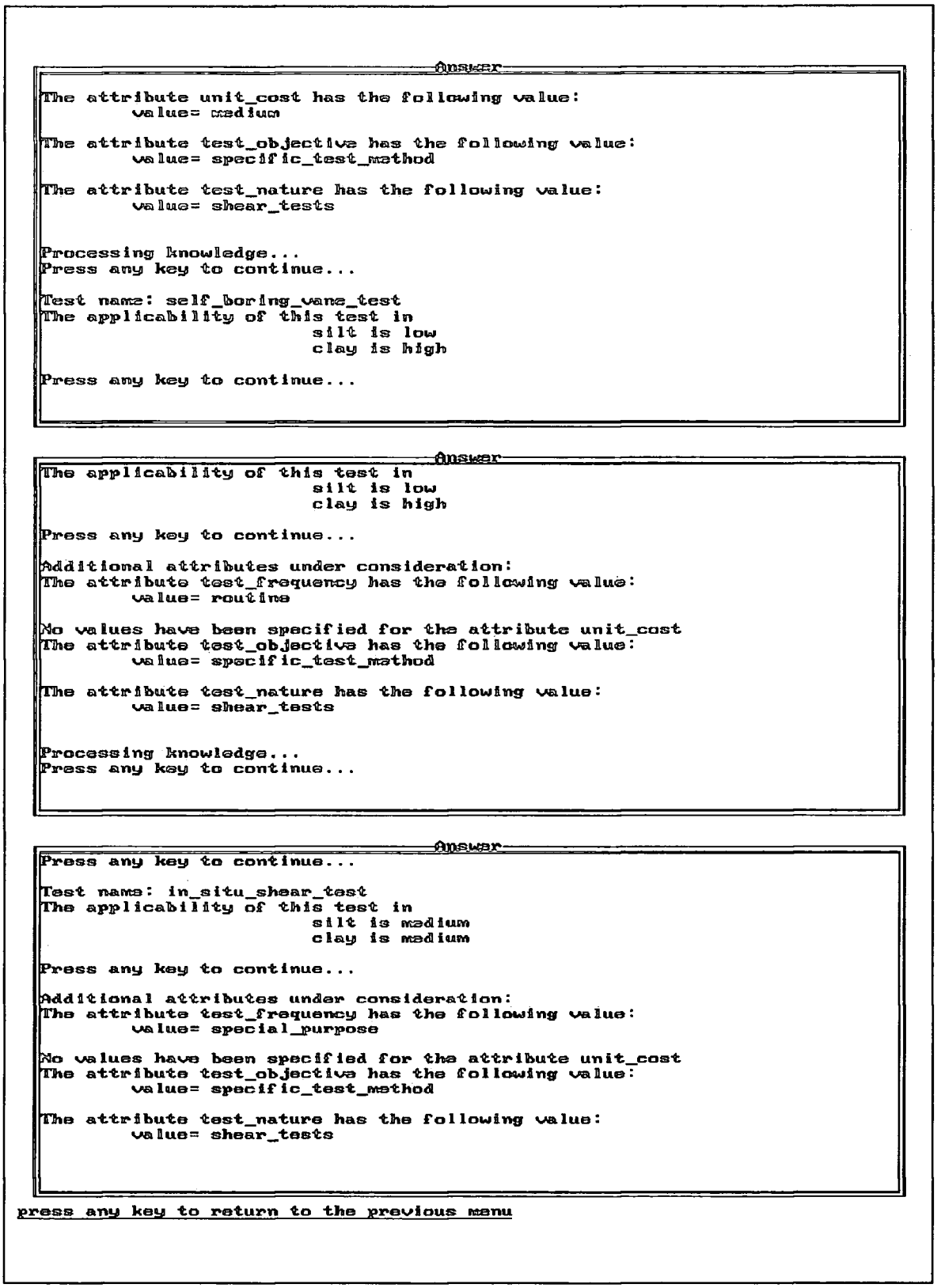


Figure 5.8 Example screen dumps of a consultation for assisting the selection of in-situ tests (Cont'd)

CHAPTER 6

IMPLEMENTATION OF GROUND INFORMATION IN PROKAPPA - - A COMPARATIVE EXERCISE

6.1 Introduction

The 'Representing the Ground' application as implemented in PDC Prolog on a Personal Computer, has been presented in Chapters 3 and 5. Near the end of this project the PROKAPPA software and a Sun Sparkstation 2 became available so a comparative exercise was carried out by implementing the ground model in PROKAPPA.

The purpose of this exercise was not to implement a fully operational application offering all the functionality of the Prolog program; it was to develop a rapid working prototype in order to appreciate the differences between the two implementation schemes of the 'Representing the Ground' application and to identify possible advantages and disadvantages that each one of them might offer. For this reason, only a part of the ground information is included in the PROKAPPA application and the system developed is not as general as that written in PDC Prolog. However, the same principle was followed of developing domain independent functions.

In section 6.2 the main features of the PROKAPPA system are presented in order for the reader to become familiar with the terminology used and to illustrate some of its capabilities that have been used in the 'Representing the Ground' application. In section 6.3 the actual implementation of the application is described in detail. Finally, in section 6.4 both implementations are discussed in a comparative way.

The 'Representing the Ground' application model as implemented in the PROKAPPA environment, is part of a knowledge-based system for the interpretation of site investigation information currently being developed in the University of Durham (Toll *et al*, 1992). It will be used in conjuction with a data checking module in order to check that values of properties entered are consistent with qualitative information from soil descriptions.

6.2 PROKAPPA in general

The PROKAPPA system provides an environment for developing and delivering multiplatform software applications. It is a C-based software development system that integrates object-orientated programming, rule-based reasoning and SQL database access in an easy to use graphical environment. Some of the main features of the PROKAPPA system that were used in building the 'Representing the Ground' application are discussed in some depth below, whilst the others are just introduced briefly. All these are discussed in great detail in the PROKAPPA manuals (PROKAPPA User's Guide, 1991).

Object System.

In PROKAPPA the basic structure for representing data is called an *object*. Objects can hold descriptive data about the entity, thing, item, concept, category or template being represented and can contain special functions which define behaviour for the thing being represented.

The PROKAPPA system has two kinds of objects: *classes* and *instances*. Classes are templates for sets of entities with common characteristics, and instances represent individual objects in the application domain. Classes and instances are organised hierarchically, so that information specified in a class is inherited by its instances. The terms *subclass* and *superclass* are used to describe relationships between objects of a hierarchy; subclass denotes a class further down the hierarchy from a specified class and superclass denotes a class further up the hierarchy from the specified class. Additional terms that serve

the same purpose are: *parent*, the class directly above a specified object (class or instance) in the hierarchy, *child*, the object directly below a specified class in the hierarchy, *ancestor*, a class at some level above a specified object in the hierarchy and finally, *descendant* an object at some level below a specified class in the hierarchy.

Both classes and instances have *slots* which represent characteristics or attributes of objects. Slots represent three type of information: i) Attributes or descriptive information about an object, ii) Actions, called *methods*, that the object can perform, iii) Relationships to other objects in a system. There are three kinds of slots: i) *Single-value* slots, which are used to store values as symbols, strings or numbers, ii) *Multi-value* slots, which can hold an arbitrary number of values of any type represented as a list of values and iii) *Method* slots which contain procedures that define the behaviour of an object.

The object system supports *inheritance*. There are two types of inheritance in PROKAPPA: a) slot inheritance which is the inheritance of the existence of slots down the object hierarchy to lower level objects and b) value inheritance which is the inheritance of slot values down the object hierarchy to lower level objects that have inherited the slot. Slot inheritance, or value inheritance only, may be blocked at any level in the object hierarchy preventing the slot or the slot value from being inherited further down. Since slots represent structures common to all instances of a class, they can be created only at class level; slot values only may be modified at instance level. Objects with multiple parents inherit information from all parents.

Slots can be further described by the use of *facets*. Facets are descriptors attached to slots which allow additional information about slots or slot values to be expressed. Like slots, facets have structures and values (a single value or multiple values) and can be inherited. Facets can be created at class or instance level.

The PROKAPPA object system supports arbitrarily complex hierarchies of objects. Object hierarchies are stored in collections called *object bases*. Objects and object hierarchies may be static models as well as dynamic as they can be created, modified and deleted at runtime. Also, information on objects can be changed at runtime. The data in an object can be accessed and/or changed by functions, rules and methods. The object system is supported by an extensive library of functions for creating and manipulating objects.

ProTalk Language

In the PROKAPPA system two languages can be used to implement applications, the C language as extended by PROKAPPA and the ProTalk language.

The PROKAPPA environment supports an ANSI standard compatible version of the C programming language plus several libraries of C functions for use specifically within a PROKAPPA application.

The ProTalk language is a language developed for use in the PROKAPPA system which can be used as an alternative to, or in combination, with C. It is particularly useful for writing code that expresses relationships between objects and facts and performs searches over object bases.

The ProTalk language incorporates a set of predefined functions for interacting with object bases and manipulating objects and provides syntax for referring to information in an object base that can be used for manipulating or retrieving information about objects, slots and facets. The ProTalk language also offers several programming constructs such as assignment of values to variables, basic arithmetic operations, comparison operators, conditional statements and iteration constructs. It has the ability to call C functions and incorporate C code. In addition to all that, the ProTalk language is a non-deterministic language which supports backtracking.

The syntax for referring to information in a object base forms a type of expression called *knowledge expression.* The major types of knowledge expressions are:

- Slot values: *object.slot*
- Facet values: *object.slot..facet*
- Instances of a class: *instanceof* class
- Subclasses of a class until the instance level: *subclassof* class
- Ancestors of an instance: *classof* instance
- Ancestors of a class: *superclassof* class

The last four knowledge expressions can be modified by the use of *direct*, to restrict the expressions to the direct (one level below or above) instances, classes, subclasses or superclasses.

In order to change or retrieve information from an object base, the knowledge expressions can be used in conjuction with the value changing operators or the search modifiers respectively.
The search modifiers used with knowledge expressions for deterministic searches are:

no modifier: For use with single value slots and facets only. Generates a single value or *Null* if there is no value.

all: Generates a single list of all the values, or the empty list if there are no values.

The search modifiers used with knowledge expressions for non-deterministic searches are:

find1: Generates one solution. Fails if there is no value.

find: Generates one solution each time the statement is executed. Fails if there is no solution. Can be re-evaluated if the system backtracks to it.

find N: Generates one solution each time the statement is executed. Fails if there is no solution. Can be re-evaluated if the system backtracks to it, at the most N times.

The ProTalk language is a hybrid language combining aspects of both procedural and rule-based languages. It can be used for writing functions and rules.

A ProTalk function is made up of one or more ProTalk statements. Each simple statement ends in a semi-colon. A compound statement is a sequence of zero or more statements wrapped in a pair of curly brackets ({}). Each statement consists of some combination of ProTalk operators, expressions, programming constructs, function calls and variables. In ProTalk there is no need to declare variables before using them, as is required when writing code in C. A function is defined by placing the keyword *function* in front of the function name, which is followed by a pair of parenthesis enclosing its arguments separated by commas.

Rules can only be written in the ProTalk language. These are a combination of ProTalk statements grouped together in *rulesets* and can be either forward chaining or backward chaining as well as mixed forward /backward chaining rules.

User interface tools

The Prokappa system allows building customised end-user interfaces to applications and provides two tools for their development:

- The *ActiveImages* system
- The *dialog box* system

The ActiveImages system is a tool for building business and instrumentation images to represent slot values graphically. The ActiveImages library provides users with a variety of output (display information only) and input images (display information and accept input information as well). This tool has not been utilised in developing the user interface for the 'Representing the Ground' application, therefore it will not be discussed in any more detail.

The dialog box system is used for obtaining arguments or options required by a command or process a program is about to execute as well as to display information, for instance, on the progress of a processing action. A PROKAPPA dialog box is a window that displays information or provides the facility to input information. A dialog box allows the user to input information in a variety of formats, using the keyboard or the mouse.

The components of a dialog box used to display information, accept information, or initiate action are called *controls*. In effect, a dialog box gets its functionality from the dialog box controls. The dialog boxes and each of its controls are implemented as instances of appropriate classes incorporated in a system object base called DialogBoxApp. These classes represent the types of dialog boxes and dialog box controls supported by the PROKAPPA system.

There are three categories of controls:

1. *Display controls*, that display a value or set of values to the user, but allow no input. These can be divided into *TextDisplay* that displays text and *PixmapDisplay* which displays bitmap images.
2. *Input controls*, that allow information to be entered by typing or by selecting one or more items from a list of choices. The input controls are: *EntryBox*, *RadioButtons*, *CheckButtons*, *ListBox* and *OptionMenu*. An *EntryBox* allows the user to type a value into the dialog box. The other four controls provide a variety of ways to present lists of choices to the user. *RadioButtons* allow the user to specify one choice out of many; only one button can be selected at a time. *CheckButtons* allow the user to select several choices out of many. A *ListBox* holds a list of items which the user can select. The display capacity of a list box can be set. It is also possible to specify whether single or multiple selections are allowed. An *OptionMenu* displays the currently selected value out of a number of possible values. The user can make a list of all possible values pop up and make a new selection.
3. *Action controls*, that initiate actions when the user clicks the mouse on the control. It is possible to have either a *PushButton* control or a *PushButtonRow* control. Whatever activity is associated with

the push button is performed at the time it is depressed. The push button row allows specification of a row of push buttons with one object. The system creates as many push buttons as specified and arranges them in a horizontal row. All dialog boxes have by default a push button row control which is called *command row control*, and contains two command buttons (push buttons), labelled *OK* and *Cancel*. Additional command row buttons can be created and the labels of the default ones can be changed. The buttons are used to either initiate or cancel the behaviour of the dialog box.

Each non-display control in a dialog box has an associated React! method which defines what happens when the user interacts with that control, e.g. depressing a push button.

Additional features

The PROKAPPA object system supports *monitors*, which are objects attached to slots that cause a function to be run when the slot value is changed or accessed. These monitors can be caused to trigger before the data is entered into a slot, after, or on demand.

The PROKAPPA substrate supports a number of data types found in symbolic programming languages but not native to C, like PROKAPPA lists which may be of arbitrary length and may contain any number of elements of any data type including other lists.

The PROKAPPA system automatically allocates and deallocates memory for PROKAPPA data structures through the substrate's memory management facilities. Automatic memory management can be turned off, if required.

The PROKAPPA development environment supports an interactive *Developer's User Interface* for the rapid prototyping and development of applications. The PROKAPPA Developer's User Interface consists of the *Application Browser*, that manages the creation, editing, loading and compiling of the

different components of an application, the *Object Browser* which is a graphical environment for the creation, modification, viewing and saving of objects, slots and facets, the *C Workbench* which is a code interpreter as well as a source code C debugger, the *ProTalk Workbench* which is a tool for debugging ProTalk code and the *Interface Workbench* that gives the ability to the developer to graphically create dialog boxes for end-user interfaces.

The PROKAPPA Data Access System supports links to either flat files or SQL-based industry standard relational databases through database mapping.

Before discussing the implementation of the 'Representing the Ground' application in the PROKAPPA system it is worth explaining that a PROKAPPA application is defined by its .app file. This file contains the information required by the PROKAPPA system to correctly load all the relevant components of the application (such as object bases, C files, ProTalk files, user defined modules, system modules required by the application, etc.) into the development environment.

6.3 Implementation

The 'Representing the Ground' application comprises the user defined application Represent and the user defined module RepresentUI. In the former the objects that make up the hierarchical model of the ground are included whilst the latter contains the objects required for the development of the user interface of the application. These are illustrated in the .app file of the application which is shown in Appendix F. It can be observed from the definition file that the system application DialogBoxApp is also required. In addition, the definition includes ProTalk files (called GRinit.ptk, GRfunc1.ptk, GRfunc2.ptk, GRfunc3.ptk, GRfunc4.ptk, GRmisc.ptk) which contain the functions required to search the ground hierarchy and to implement the user interface of the application. The development of the Ground Object Base and the implementation of the search routines and the user interface module are described in detail in the rest of this section.

The object system that PROKAPPA supports, facilitated the representation of the ground. The object base was created using the *Object Browser* and not programmatically. It consists of a hierarchical structure starting with *Ground* as the top level class and culminating in specific instances of the different soil types. The Ground hierarchy included in the Ground Object Base is presented in Figure F.1 in Appendix F. The *Non-Organic* branch of the hierarchy leads to the instances *Boulders*, *Cobbles*, *Gravel*, *Sand*, *Silt*, *Clay* which represent, as was said before, the non-organic dominant soil types. The most detailed level following the *Organic* branch consists of the instances *Organic Sand*, *Organic Silt*, *Organic Clay* and *Peat*. As the object system supports arbitrarily complex hierarchies it was possible to represent the instance *Silt* as the child of two parents *Granular-Fine* and *Fine* which are subclasses of *Granular* and *Cohesive* respectively. The same applies to *Organic Silt*. Utilising the PROKAPPA system's inheritance facilities the slots were created once at the appropriate class level and they were inherited by all the subclasses and instances of that class. The values assosiated with these slots became more specific progressing further down the hierarchy, where necessary. The above are better illustrated in the following example. It should be noted that only knowledge concerning *grain size* and *liquid limit* has been included in the system. This information is not shown in Appendix F as it is not incorporated in the application programmatically.

Following the branch of *Non-Organic* the slot *grain size* was created at the level of the class *Non-Organic* as a multi-value slot and it was automatically inherited by all its subclasses *Granular*, *Cohesive*, *Very Coarse*, *Coarse*, *Granular-Fine* and *Fine* and all its instances *Boulders*, *Cobbles*, *Gravel*, *Sand*, *Silt* and *Clay*. The value of the slot *grain size* as defined at the level of the class *Non-Organic* is the range 0 - 2000 mm. This range is actually represented as a list of values, (0, 2000). The inherited slot at the level of the class *Granular* takes as value the range 0.002 - 2000 mm, whilst at the level of the class *Coarse* the range becomes 0.06 - 60 mm. At the level of the instance *Sand* the value of the inherited slot is again modified to the range 0.06 - 2 mm becoming even more specific. The instance *Silt*, whose both parents have the same slot, inherits it only once and the slot's value is defined at this level as the range 0.002 - 0.06 mm. The slot *liquid limit* on the other hand, was created

at the level of the class *Cohesive* having the structure of a multi-value slot and its value was defined as the range 0 - 200 %. Both the slot and the slot value were inherited by the subclass *Fine* and the instances *Silt* and *Clay* of the class *Cohesive*.

It was found that PROKAPPA can handle a more detailed representation scheme like the one described in section 3.2, with the use of facets, which are in effect slots on slots. As was discussed before, the instance *Sand* has inherited a multi-value slot called *grain size* whose values are the lower and upper limit of the range 0.06 - 2. This range can be further subdivided into more specific ranges such as 0.6 - 2 for a *coarse* grained *Sand*, 0.2 - 0.6 for a *medium* grained *Sand* and 0.06 - 0.2 for a *fine* grained *Sand*. Three multi-value facets were attached to the slot *grain size* of the instance *Sand* in order to represent this additional information. The facets were named *coarse*, *medium* and *fine* after the descriptive terms that the above ranges express. Each of these three facets has a list of values containing the lower and upper limit of the corresponding range. The grain size subdivisions for the instances *Gravel* and *Silt* were represented in the same way. The slot *liquid limit* can have a more refined representation as well. The range 0 - 200 % can be subdivided into five smaller ranges. Five facets have been attached to the slot *liquid limit* of the instances *Silt* and *Clay*, having the names *low plasticity*, *intermediate plastisity*, *high plasticity*, *very high plasticity* and *extremely high plasticity* and the values 0 - 35%, 35 - 50 %, 50 - 70 %, 70 - 90 % and 90 - 200 % respectively.

This model, when combined with functions that are able to retrieve information from it and a user interface module, provides the functionality of a search-based application. The user is able to search the hierarchy to provide solutions to questions of varying degrees of detail. As mentioned before, the system developed provides a lot of flexibility as it is possible to make modifications to the model (adding or deleting information) without changing the searching routines. The functions and the user interface module developed are presented below. A full listing of the program, which is divided into separate files for clarity, is given in Appendix F.

The process part of the application consists of functions that are user defined or provided by the PROKAPPA system in order to retrieve information from the ground model. These were witten in the ProTalk language which is suited to writing code that references, finds, modifies or reasons over information stored in an object base (PROKAPPA User's Guide ,1991, pp. [6-13] - [6-15]).

Some system defined functions used in this implementation scheme to retrieve information stored in the Ground Object Base, are presented below. These are included in the ProTalk function libraries.

Function **ObjectSlots**

Input values: The name of an object in the hierarchy

Output values: A list of the names of all slots in the object

Function **GetValues**

Input values: The name of an object in the hierarchy

The name of a slot in the object

Output values: The current list of the slot values

Function **SlotFacets**

Input values: The name of an object in the hierarchy

The name of a slot in the object

Output values: A list of the names of all facets in the slot in the object

Function **GetFacetValues**

Input values: The name of an object in the hierarchy

The name of a slot in the object

The name of a facet attached to this slot

Output values: The current list of the facet values

In order to fulfil the requirements of the application, additional information retrieval functions had to be written to allow a more complicated search in the object base to be performed. These are described below.

Function **FindAncestors**

Input values: The name of an object in the hierarchy

Output values: A list of all the ancestors of the object

The function checks if the input object name is an instance or a class and finds its ancestors using the *classof* or *superclassof* knowledge expressions (in conjuction with the *all* search modifier) respectively.

Function **FindFacets**

Input values: The name of an object in the hierarchy that contains slots with defined facets

The name of a slot in the object that has facets attached to it

A value of a facet attached to that slot

Output values: A list of the names of the facets (modifiers) that this value corresponds to.

The function carries out a comparative information retrieval by searching all the facets in the slot in the object and checking in each one of them if the input value lies within the corresponding predefined range included in the object hierarchy. Its implementation in the ProTalk language is shown below.

```
function FindFacets(?obj, ?slt, ?f_val)
{
    bound inputs;
    ?ans_list = `();
    ?facet_list = SlotFacets(?obj, ?slt);
    for ?facet_name inlist ?facet_list;
    do
    {
        ?f_vallist = GetFacetValues(?obj, ?slt, ?facet_name);
        ?min = ListFirst(?f_vallist);
        ?max = ListNth(?f_vallist, 1);
        if
        {
            ?f_val >= ?min;
            ?f_val <= ?max;
        }
        then
        {
            ?ans=AppendStrings(ConvertToString(?facet_name), " ", ConvertToString(?obj));
            collect ?ans into ?ans_list;
        }
    }
    return ?ans_list;
}
```

The function initially generates a list (?facet_list - the question mark denotes a ProTalk variable) with the names of all facets in the slot (?slt) in the object (?obj) using the **SlotFacets** ProTalk function. For each member (?facet_name) of the list (generated using the *inlist* operator that provides iteration over elements) it checks whether the input facet value (?f_val) lies within the the minimum (?min) and maximum (?max) values of the range specified in the object base for that facet. If it does, the facet name is collected in a list (?ans_list), using the *collect into* operator. The corresponding values of the facets are retrieved using the system defined function **GetFacetValues**. The for/do construct was used to provide iteration over a statement. When all possibilities have been examined the list (?ans_list) containing the required facet names is returned.

This function is not as general as the corresponding Prolog rule as it only accepts for input value one numerical value. However it satisfies the purpose of this exercise which was to demonstrate the application of the concepts developed in the PDC Prolog program in another tool.

Function **FindObjectsAndFacets**

Input values: The name of an object chosen as the search-origin point
The name of a slot in an object within the hierarchy
A value of that slot

Output values: The name(s) of the object(s) that correspond to the input slot name and value
A list of the names of the facet(s) in that slot in each object that this value corresponds to (if any)

This function performs a guided search within the model, starting from the search-origin point and identifying its subclass in which the input slot name is defined and has as values a range that corresponds to the input value. This subclass becomes the new search-origin point and the same check is repeated. This selective search ensures that the path leading to the solution(s) is always being followed. In the same way the required instances are identified. For each of these instances the facet

name(s) (if any) that correspond to the input value are found, performing a comparative search in the same way as described for the function **FindFacets**.

Additional functions were written which are used to collect information from the object base enabling the user to make selections expected as input. For example, the function **ListObjs** finds all the objects in the hierarchy; the function **ListObjMods** provides all the objects having slots defined, as well as all the objects whose slots have facets attached to them; the function **GetSlotList** retrieves all the different slot names existing in the model; finally, the function **CheckSlots** returns the slot names of all the slots which are defined in an object having facets attached to them. Also, some functions provide allowable input ranges in cases where the user is required to enter a numerical value. For example, the function **FindSlotRange** finds the minimum and maximum value defined for a slot within the whole model. These functions, as well as the functions described earlier in this section, support in general the main functionality of the system which is the domain independency of the process mechanism. In the cases where this was not achieved (due to time constraints), it is considered that no major changes are required to allow complete domain independency.

It can be observed from the listing of the program (Appendix F) that some functions return a value or a list of values (e.g. the function **FindFacets**), others set the value(s) of a control of a dialog box of the user interface (e.g. the function **FindObjectsAndFacets**) while others set the value(s) of a slot in an object created to serve as global variable storage (e.g. the function **ListObjMods**). This is mainly due to the infamiliarity with the software and the limited time that was available for the development of the 'Representing the Ground' application in PROKAPPA (that took place during the initial parts of the learning curve). An interesting point that came out through these different approaches is that PROKAPPA's main feature is its ability to create and manipulate object bases. Therefore, values being stored as slot values of an object can be accessed at any point in the execution of a program very rapidly and efficiently. It is also worth noting that the only way that the ProTalk language provides for global variable storage is using objects and slots. Another advantage that was found using the latter approach

is that functions (such as the ListObjMods) that produce general information from the model, could be triggered when the system is initiated. This information is then available to be utilised during the consultation period, reducing the response time of the system.

The user interface module of the system is provided by dialog boxes, which are created through PROKAPPA to provide full X window capabilities. The user interface module has been developed programmatically, and not graphically by using the *Interface Workbench*.

The system is invoked by calling the function Main_Menu (). A dialog box, called Function Menu appears on the screen, consisting of a ListBox input control that holds a list of all the options offered by the system and a command row control that contains the OK command button. The user is then required to select one of these options, listed below, and initiate the appropriate actions by clicking the OK button.

- **List Ancestors**

  The user is required to select an object from a ListBox holding all the objects (classes and instances) of the hierarchy. The ancestors of the chosen object are identified.

- **List Slots**

  The user is required to select an object from a ListBox listing all objects within the hierarchy that contain slots. The slot names in that object are presented in a second ListBox and, if required by the user, their values are retrieved.

- **Find Object Modifiers**

  The input required by the user in this case is the name of an object (selection item in a ListBox containing all objects having slots with defined facets), the name of a slot in that object (all slots in that object are presented to the user after his/her first selection in a second ListBox, displaying also their allowable input range) and a value within the allowable range (entered by the user in an EntryBox). This option returns the modifier(s) (facet name(s)) that match the given data. A

data validation process is also performed and in the case of a wrong input value in the EntryBox an error message appears.

- **Find Objects and Modifiers**

  On selecting this option, the user is required to input the name of a slot (selection item in a ListBox containing all different slots existing within the object base; their allowable input ranges are also given for guidance), and a value of that slot (entered by the user in an EntryBox). Data checking occurs in this case, as well. This option produces the corresponding object and modifier(s) (if any) to the input data. Alternative solutions are also generated.

- **Exit**

  Allows the user to exit the system.

These are better illustrated in section 6.4 that presents example consultations with the system.

The user interface module is a dynamic module, as the dialog boxes required are created when needed at run time. On selecting any of the first four options the Function Menu dialog box is taken off the screen and replaced by an appropriate dialog box corresponding to the requirements of the selected function. This secondary dialog box is constructed at run time, from arbitrary dialog box controls that are already present in the interface module defined as instances. These secondary dialog boxes require display windows for outputting the results of their function, these being constructed in a similar manner. This allows a minimum of dialog box controls to be defined for the interface module, as they can be used in various combinations in all of the appropriate function interfaces. In this way the congestion of the object base and its associated tools is prevented.

At each level of the system, the user has the option to either make a new selection or return to the Function Menu dialog box to choose another option or exit the system.

6.4 Example Consultations with the System

In this section example screen dumps generated during execution of the 'Representing the Ground' application are presented in Figures 6.1-6.4. In Figures 6.1a-6.1b the user selects the option **List Ancestors** in order to retrieve from the object base the ancestors of an object. In Figure 6.1a the instance **Sand** has been selected and its ancestors are displayed in the Display Ancestors dialog box. In the screen dump illustrated in Figure 6.1b the ancestors of the instance **Silt** are identified. Silt has two different sets of ancestors (Fine, Cohesive, Non_Organic, Soil and Ground - Granular_Fine, Granular, Non_Organic, Soil and Ground); these are displayed in the Display Ancestors dialog box avoiding the repetition of the common ones.

In Figure 6.2 the user is interested in searching the object base, selecting the action **List Slots**, in order to find the attributes of the object **Sand** (which are displayed in the second ListBox) and then he/she queries about the values of the attribute **Grain_Size**, which are displayed in the Display Slot Values dialog box.

In Figure 6.3 the action **Find Object Modifiers** is selected that allows the user to retrieve the modifiers that correspond to an input value of **50 %** of the attribute **Liquid_Limit** of the instance **Silt**. These are displayed in the Display Object Modifiers dialog box.

Finally, in Figure 6.4 the user selects the action **Find Objects and Modifiers** in order to interrogate the object base about the object(s) and modifier(s) (if any) that correspond to an input value of **2** mm of the attribute **Grain_Size**. These are displayed in the Display Objects and Modifiers dialog box.

As can be observed from the examples presented in the last two cases (actions **Find Object Modifiers** and **Find Objects and Modifiers**) the relevant allowable range of values is also provided for each attribute displayed, in order to guide the user to input an appropriate value.

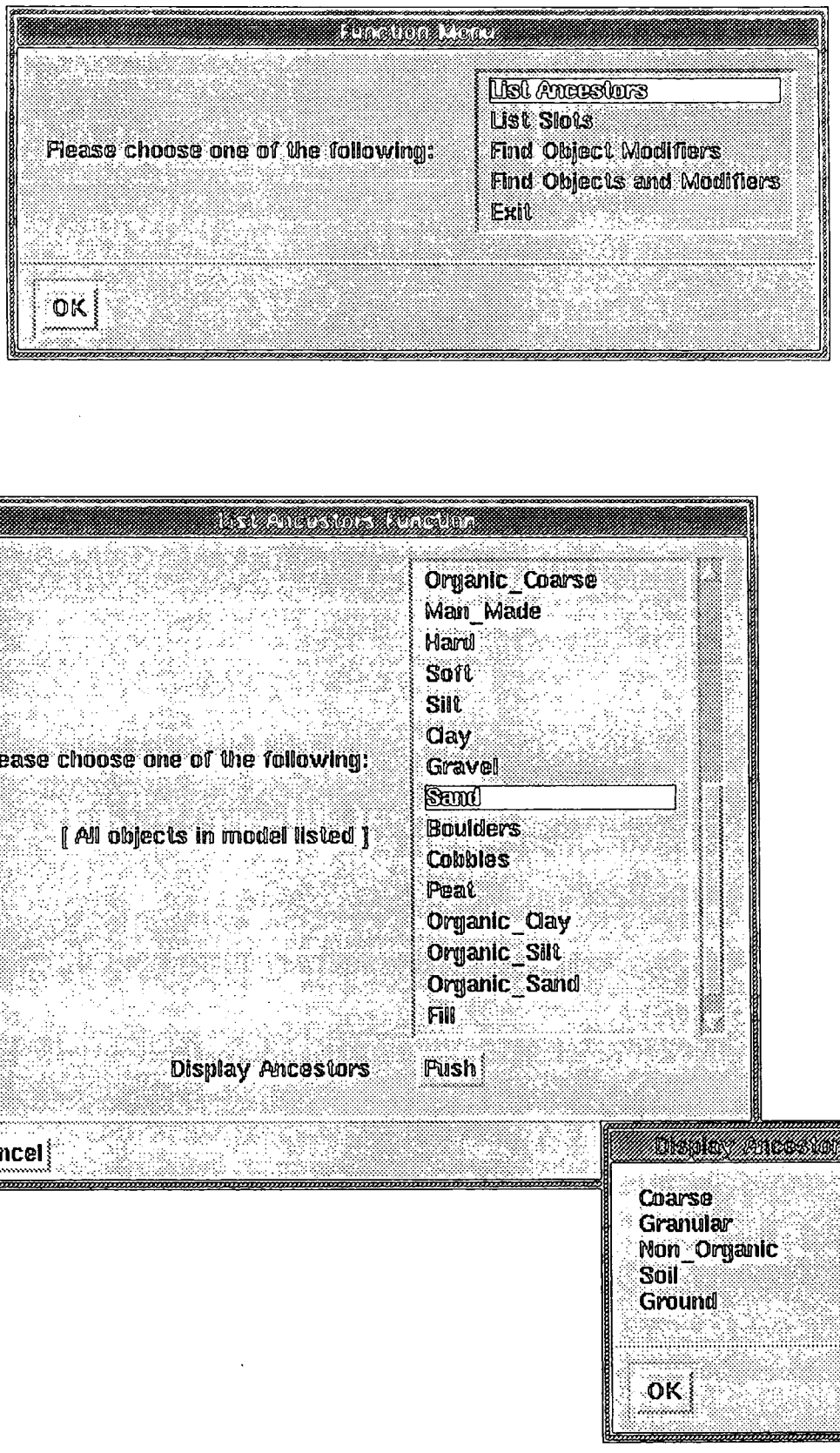


Figure 6.1a Example screen dumps for interrogating the object base about the ancestors of an object

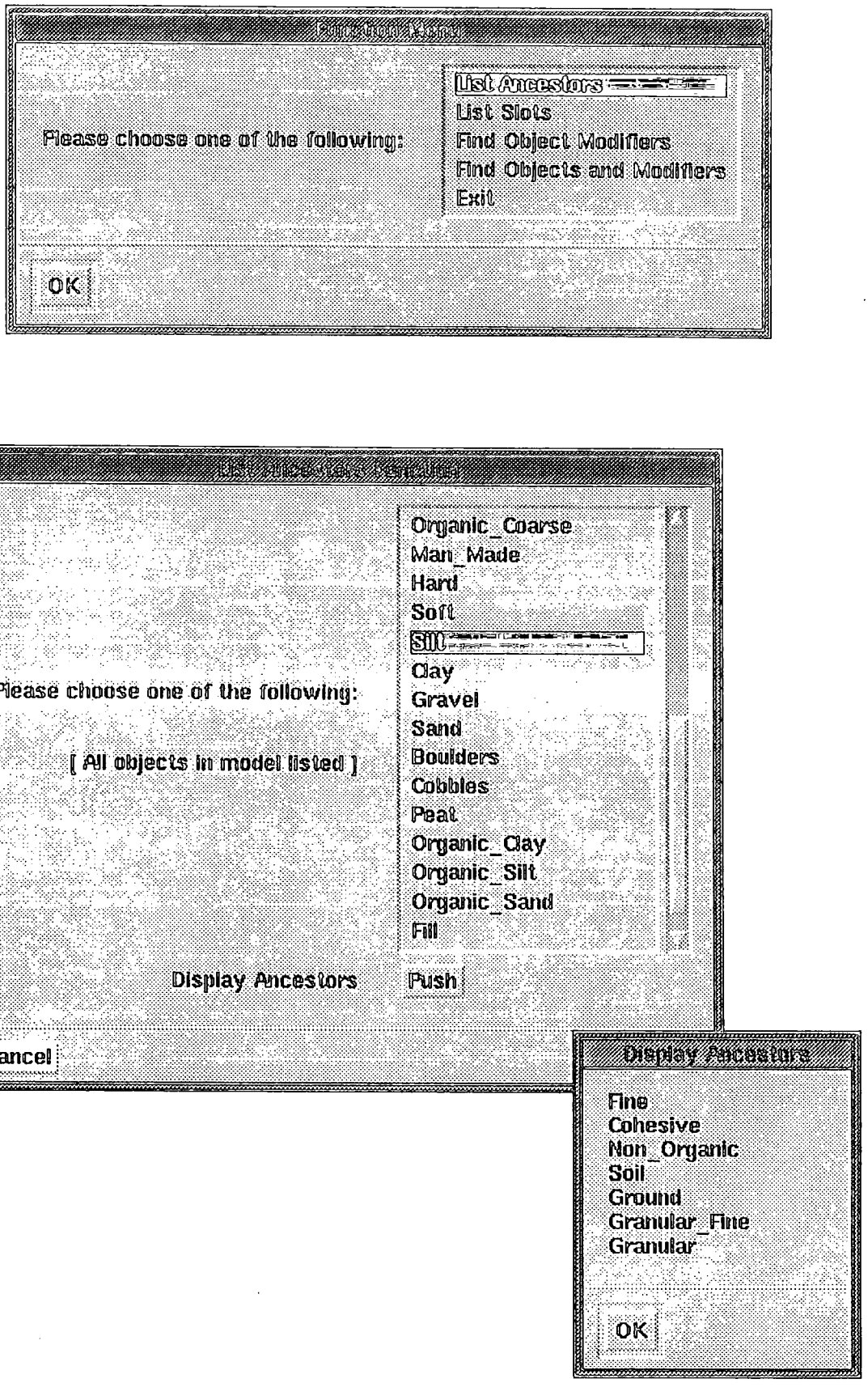


Figure 6.1b Example screen dumps for interrogating the object base about the ancestors of an object

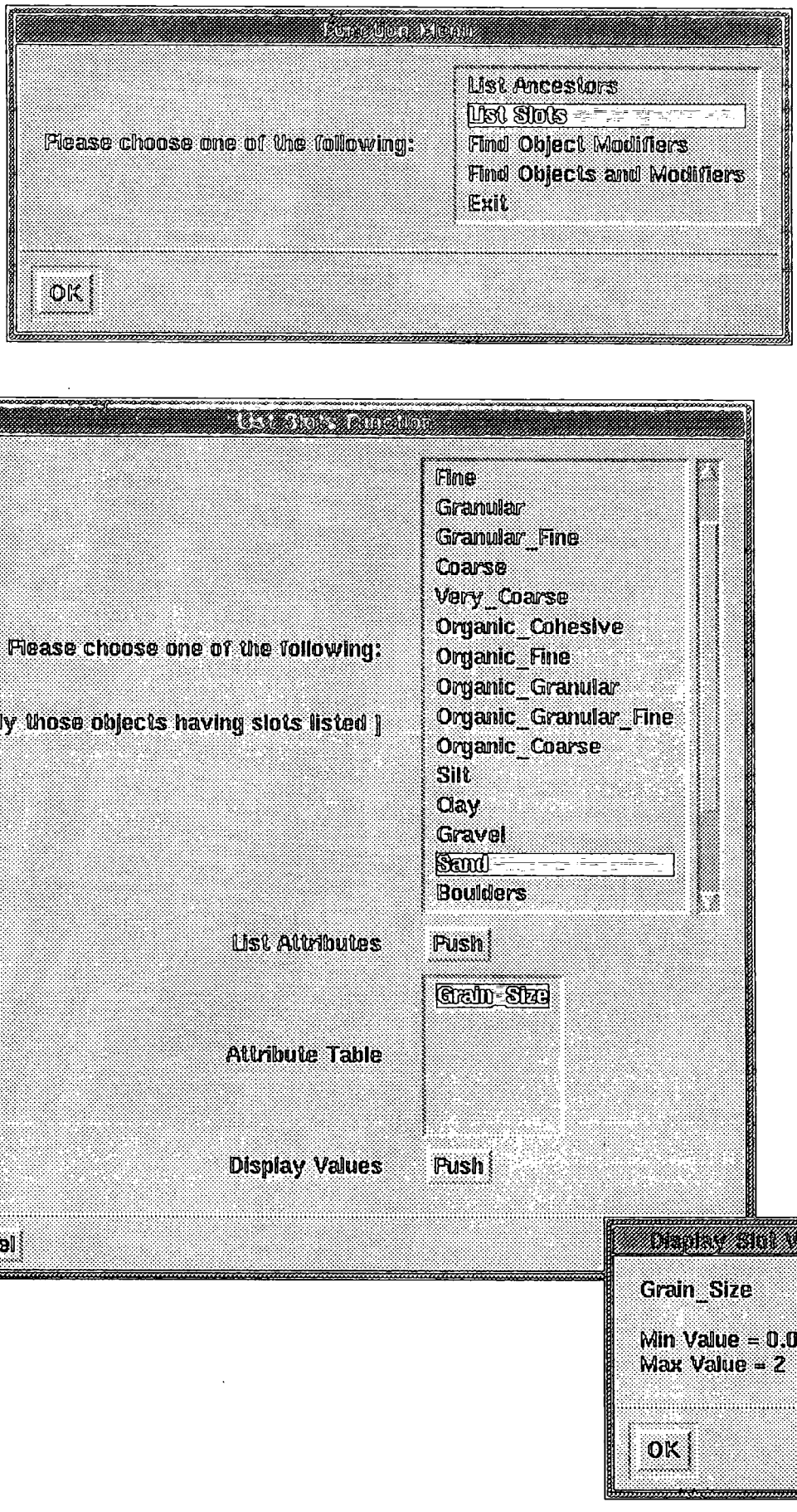


Figure 6.2 Example screen dumps for interrogating the object base about the attributes of an object

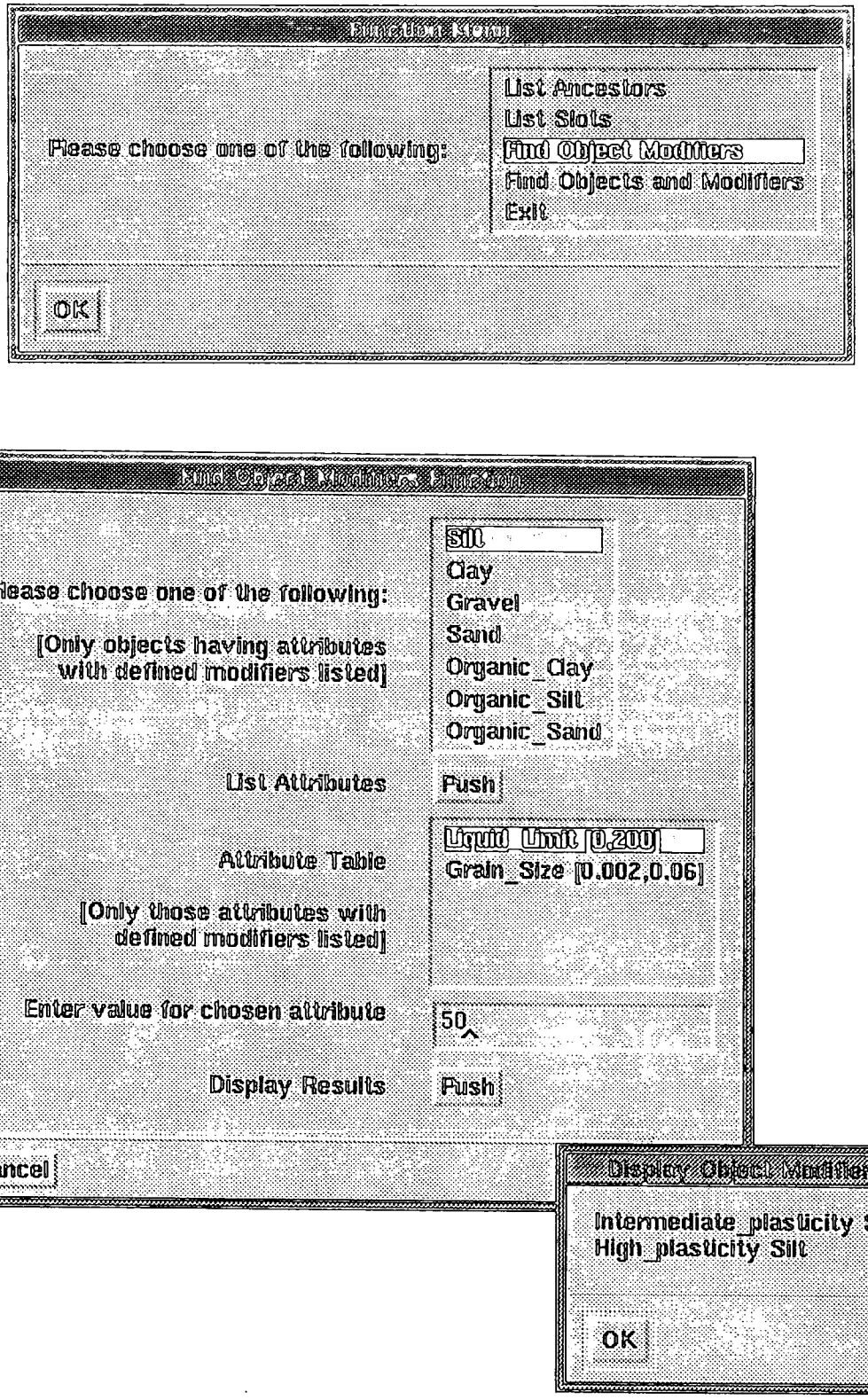


Figure 6.3 Example screen dumps for interrogating the object base about the modifier(s) that corresponds to an attribute-name and attribute-value of a chosen instance

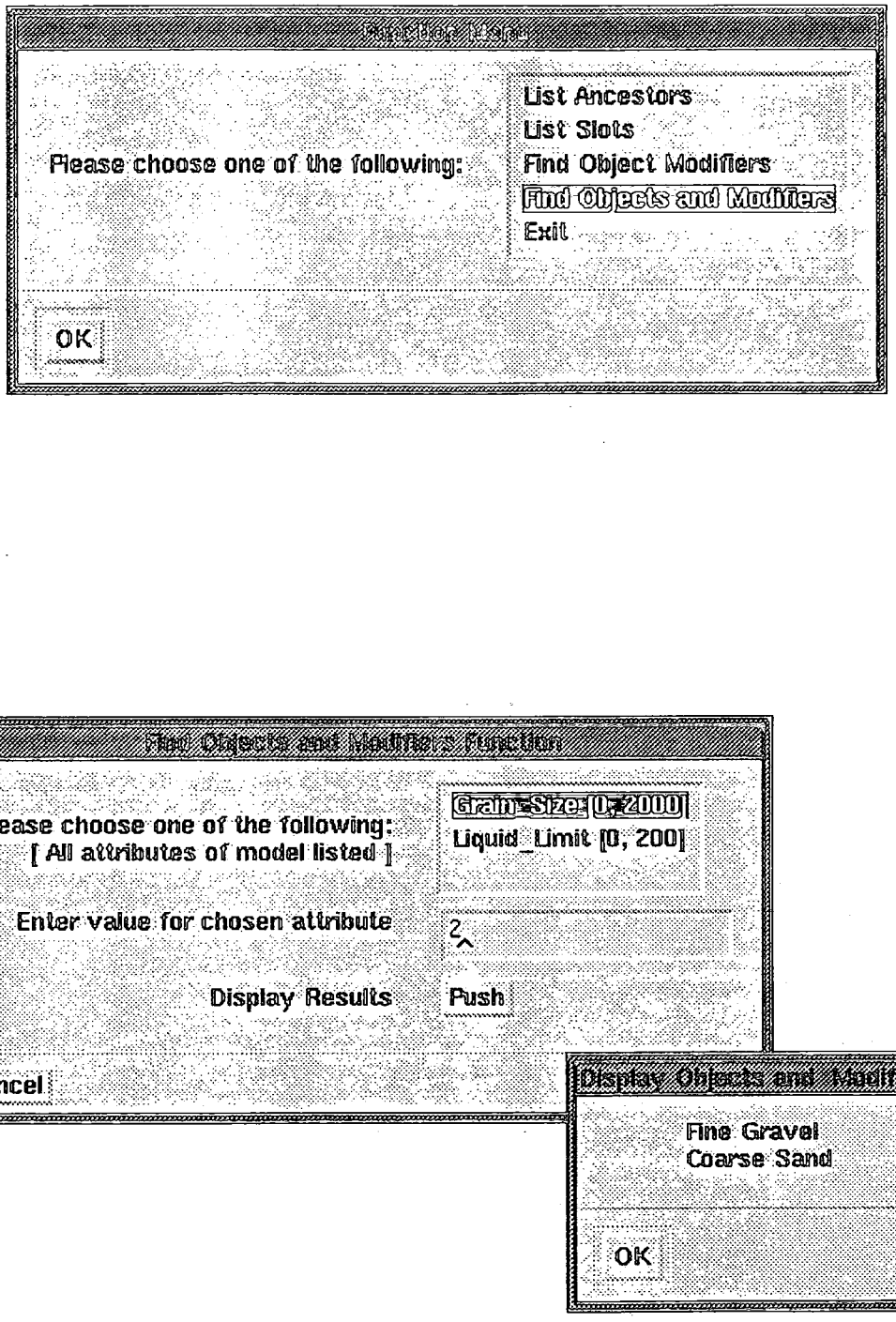


Figure 6.4 Example screen dumps for interrogating the object base about the object(s) and modifier(s) that correspond to an attribute-name and attribute-value

6.5 Comparative comments on the implementations in Prolog and PROKAPPA

Selecting the appropriate software and hardware for the development of a system is an important and crucial task because this decision may determine the future of the application. It is often the case however, that there is not much choice in the initial stages of the development of an application when decisions need to be made. This is usually due to the fact that at these early stages it is very difficult to identify the real needs of the system. Another common reason is a lack of financial resources, which results in limiting the range of choices.

The model of the ground has been implemented in PDC Prolog on a 286 Nimbus AX/2 Personal Computer (as presented in Chapters 3 and 5) and in PROKAPPA on a Sun Sparkstation 2 (as described in section 6.3). As has already been said in section 6.1, this has been done for two reasons: i) as a comparative exercise between the two software packages in implementing the 'Representing the Ground' application and ii) because the ground knowledge base implemented in PROKAPPA will be part of a knowledge-based system for interpreting geotechnical information from a site investigation which is currently under developement at the University of Durham (Toll *et al*, 1992). The main development environment for this system is PROKAPPA.

The 'Representing the Ground' application could be considered as an object-orientated search-based application. The knowledge domain has been represented by a model of the ground consisting of objects which are organised in a hierarchy and are defined by their properties using inheritance. Prolog is a general purpose representational language (Maher and Allen, 1987) and search and pattern matching are capabilities that the language features. It is worth noting as well that Ruggieri *et al* (1992) have presented the implementation of a Prolog-based object-orientated environment. Another advantage of Prolog is that in effect it is not just a programming language; it provides additional features such as a database system, a backward chaining inference engine (Marcellus, 1989, Reintjes, 1992), although it is not very expensive. PROKAPPA on the other hand, as it is an object-orientated software package, is

particularly suitable for this type of applications. It has to be mentioned though that PROKAPPA is an expensive and complex piece of software that requires a lengthy process in order to become familiar with it, as well as with the hardware required to run it and really being able to evaluate it. Its cost has to be justified by the need of implementing a complex system such as the one presented by Toll *et al* (1992).

In PDC Prolog, the hierarchy of the ground had to be described by Prolog facts defining each object of the hierarchy by its name, its members and its properties. The tree-like structure representing the ground is implicit; it only exists through the logical relations between the classes. The model represented is a general tree that accepts multiple parents. In the representation scheme achieved in PDC Prolog it is possible to distinguish between three types of objects: a) the top level class which is the root of the hierarchy and expresses the domain of the knowledge represented. No properties are specified for this class. b) subclasses which are the nodes of the hierarchy. The properties of these classes are inherited by their subclasses and instances. c) instances which are the leaves of the hierarchy; instances have no members.

The properties of each object are represented using PDC Prolog's multi-level compound objects in order to allow an attribute to have multiple lists of values according to a list of factors. It is possible to define properties at a class or instance level. Two types of attributes exist within the hierarchy: i) the attributes that are defined only once and are inherited by the levels below in the hierarchy. These may also allow identification of the position in the hierarchy and ii) the attributes that are defined at many levels and become more specific going further down the hierarchy. In this case the current level inherits the attribute name from the level the attribute was firstly defined but the value specified at the original definition is overwritten by the value specified at the current level.

It was possible in PDC Prolog to achieve a representation scheme that has another level of detail, introduced by the predicate *modifier*, to handle more detailed classifications concerning the instances of

the hierarchy. So, instances get their properties in three ways: they inherit properties from their ancestors, they have properties, defined at their level within the structure and they have properties defined in a more detailed level independently from the structure.

As PDC Prolog is a general purpose language it does not provide any facilities or tools for manipulating objects or object hierarchies required in an object-orientated application. Hence, inheritance and transitivity inferences, as well as information retrieval rules, had to be implemented by the developer.

A menu-driven user interface has been implemented for this application utilising the tools provided by PDC Prolog. These tools are mainly text-based and they do not include a high level windowing toolkit. For this reason the user interface developed, although is considered to be efficient, does not look professional.

The PROKAPPA object system significantly facilitated the implementation of the 'Representing the Ground' application. The model of the ground was created using the graphical environment that the Object Browser provides. This facility enables the rapid creation of object bases, as it does not involve any programming. The objects in the ground model are organised hierarchically starting with the top level class (Ground), going through subclasses to instances (dominant soil types). As PROKAPPA supports arbitrarily complex hierarchies the case of an object having multiple parents (e.g. Silt) was not a constraint.

The PROKAPPA ground hierarchy consists of two types of objects: i) the classes and ii) the instances. A special case of a class is considered the top level class that has no parents. Classes are defined by their properties which can be inherited by their subclasses and instances. Instances can inherit information from classes but it is not allowed to define properties at their level. However it is possible to modify the value of an inherited property at that level or to block the inheritance of the property totally. Properties are represented in PROKAPPA by slots, which can take one value or a list of values.

It was found possible to achieve in PROKAPPA the more detailed representation (implemented in Prolog by the use of multi level compound symbols) using facets which are descriptors attached to slots. The facet name corresponds to the factor in Prolog and the value(s) that a facet can have corresponds to a subdivision of the general range of values defined at the slot level. As many facets can be attached to the same slot, it becomes possible for a slot to have multiple values according to a factor.

In the PROKAPPA object system, instances may only exist as children of classes; so the second level of detail introduced in PDC Prolog by the user defined predicate *modifier* in order to hold more refined classifications of the instances (dominant soil types) was not possible in PROKAPPA. The information hold at that level of detail in Prolog had to be incorporated within the hierarchy.

Inheritance inferences need not to be implemented by the developer as they are provided by the object system. PROKAPPA also provides a number of functions for manipulating objects, slots and facets in a rapid and efficient way enabling the programmer to concentrate on the implementation of specific requirements of the application.

As the PROKAPPA interface directly utilises X-windows widgets, the user interface implemented in PROKAPPA for this application provides the look, feel and functionality of the X-window system.

In conclusion it could be said that both PDC Prolog and PROKAPPA proved adequate for the development of the 'Representing the Ground' application, each one providing different advantages to the programmer and to the final system. PDC Prolog being a general purpose representational language provides more flexibility, allowing the programmer to implement the application in the most appropriate way, with the only constraints being those of the language. These constraints should be appreciated at the initial stages of the development of an application, as they could prove critical at later stages especially if the aim of the implementation is a commercial system. PROKAPPA on the other hand being an object-orientated software package provides a more fixed way of implementing

applications, but offering to the developer a number of tools that significantly facilitate the development of applications that require an object-orientated approach, such as the 'Representing the Ground' application.

It is also worth noting a few more general points that arose during the implementation of this application in PDC Prolog and in PROKAPPA. PDC Prolog, was found to be a tool well suited for cost-effective, rapid prototyping of complex applications, whereas PROKAPPA being a complicated software package requires a lot of familiarity to be developed in order to produce a working prototype. Once the necessary level of familiarity has been achieved, however, the tools provided by the system facilitate the development process, also reducing the implementation time of an application. It must be stressed however, that committing to complex software without a good appreciation of their capabilities and limitations may prove to be a critical factor in the future development of the application.

Finally, both PDC Prolog and PROKAPPA provide tools for developing efficient customised user interfaces. However, the functionality of the user interface developed in PROKAPPA for the 'Representing the Ground' application looks more professional, a feature that is considered to be important especially if the aim of an implementation is to produce a commercial system.

CHAPTER 7

DISCUSSION

Geotechnical Engineering is concerned with the study of the earth materials for construction purposes. This involves the measurement of properties (such as strength, compressibility and permeability) in-situ, as well as in the laboratory.

The interest of the engineering community in in-situ test methods has increased rapidly during the last few years, as they provide a means of improving soil profiling and facilitating the rapid determination of soil parameters. Several benefits can be realised by employing in-situ techniques, rather than conventional drilling and laboratory tests, to obtain these data (Wroth, 1984; Robertson, 1985,1986; Orchant et al, 1988,).

A wide variety of in-situ tests has been developed and is still developing, each of these tests having different uses and limitations. The selection of appropriate in-situ tests allows a more efficient and cost-effective design to be achieved.

Selecting suitable test methods, however, is not an easy task; it requires a considerable amount of knowledge mainly gained through experience. Any computerised system that aims to assist in the decision making process should be able to incorporate and provide this information to the user in order to allow successful engineering judgements to be made. Knowledge-based system technology can be applied to such geotechnical problems as it provides a medium that can accommodate the representation and use of knowledge.

A Knowledge-Based System has been developed to assist in the selection of suitable geotechnical field tests. The system allows appropriate decisions to be taken by providing knowledge on different in-situ test methods. The system is not intended to replace a human expert; it should be considered as a decision-support system and as a learning tool.

The system incorporates two knowledge bases (the Ground Knowledge Base and the Tests Knowledge Base), an inference mechanism allowing the interrogation of the knowledge bases, an advisory rule aiming to aid the selection of suitable test methods and a user interface facilitating its use. Each part of the system will be briefly reviewed below and possible improvements will be discussed where applicable.

The Ground Knowledge Base, as described in Chapter 3, contains a model of the ground. The level of detail introduced in order to satisfy the system's requirements is a broad geological classification based on the British Standards (BS 5930, 1981). In this hierarchy the ground is described at the higher level by classes such as Soil or Rock and at the lowest level by instances such as Sand, Silt, Clay etc. Knowledge about grain size, liquid limit, consistency, permeability, compressibility and secondary soil types is included.

The Tests Knowledge Base, as described in Chapter 4, contains a test hierarchy at the most detailed level of which individual in-situ test methods can be identified. Knowledge about these test methods, that enables successful engineering decisions to be taken in respect to selecting appropriate tests, is included in the knowledge base. This knowledge consists mainly of two types of information, the reliability of a test for obtaining specific geotechnical information (assuming ideal ground conditions and taking into account all necessary correlations) and the applicability in different types of ground. In addition, knowledge concerning the test frequency, test objective, and unit cost has also been incorporated for the various tests.

In accordance with what has been identified in Chapter 2, the most difficult and time consuming task in the development of the system was found to be the knowledge acquisition. The knowledge required for the Ground Knowledge Base has been derived from the relevant literature, a fairly straightforward process. It was observed, however, that a consistent omission existed in the data; in most cases where a scale was provided for defining an attribute (e.g. uniaxial compressive strength of rocks, undrained shear strength of cohesive soils, etc.) the lower and the upper limits were not explicitly defined. Wherever the missing values were not obvious (e.g. a "0" value being the lower limit of a scale), additional references had to be consulted in order either to find the missing value explicitly stated or to assume it from typical values presented.

The development of the in-situ test hierarchy incorporated in the Test Knowledge Base proved to be a lengthy process. Since in-situ testing has developed rapidly during the last decade, most of the recent developments were not included in published textbooks or relevant standards. Hence, a thorough review of in-situ testing was conducted by identifying and consulting recent technical publications (papers and reports). A difficulty that was recognised during this process, also mentioned in Chapter 4, was that in many cases tests were described in the published literature under different names although the same test method was implied. The in-situ tests hierarchy achieved is considered to be a valuable compilation of Site Investigation procedures, providing a good indication of the wide variety of tests that have been developed and at the same time a framework for the inclusion of further developments. The list of the individual test methods included in it is by no means exhaustive; however, it demonstrates the current state of the in-situ testing, covering the major field testing techniques already accepted and used in the subsurface exploration industry, as well as the testing methods being at the late stage of research.

The knowledge about each test method required to be included in the Test Knowledge Base was found to be difficult to identify from published literature for all the many types of field tests, as this is mostly gained through experience. Hence a knowledge elicitation exercise in the form of a questionnaire was

also carried out in order to collect the required expertise. Although the results of the survey were found promising, providing the desired information for the vast majority of the individual test methods under consideration, they lack statistical robustness. This is mainly due to two reasons: i) some of the more 'exotic' tests included in the questionnaire were unknown to most or all the respondents and ii) in industry generally (and in the present recessionary climate in particular), the respondents did not feel able to devote the time to completion of the complex and comprehensive questionnaire. However, it would not have been satisfactory to dilute the questionnaire for industrial purposes. Having considered all these factors the only changes that would have been made would perhaps have entailed a more solicitous and earlier approach and this will be the case in further development of the system.

The system, as at present, is mainly concerned with in-situ tests performed in soil; hence, only these tests are incorporated in the Tests Knowledge Base and only soil information is represented in detail in the Ground Knowledge Base. In future development, the two knowledge bases should be completed by including rock information in the Ground Knowledge Base and in-situ tests used in rock in the Tests Knowledge Base. In this way expertise on field tests used in rock will also be provided by the system. In addition, the Tests Knowledge Base could be expanded to incorporate knowledge on the other categories of geotechnical testing, i.e. Large Scale Field testing, Back Analysis and Laboratory testing.

Both knowledge bases have been implemented in the same way. It is believed that the representation scheme achieved in this implementation allows the incorporation of additional knowledge, as well as the alteration of the existing knowledge, to be easily made without affecting the overall structure. This enhances the functionality of the system because it allows the existing knowledge to be completed or amended at a later stage of development as well as additional knowledge that has not been considered in the course of this research to be incorporated.

The Ground and Tests knowledge bases developed for this system are to be part of a Knowledge-Based System currently being developed at the University of Durham for interpreting geotechnical information

from a site investigation (Toll et al, 1992). The system is implemented in the PROKAPPA development environment and it is the intention to convert the Tests Knowledge Base implemented in PDC Prolog for use in the same environment, as happened to the Ground Knowledge Base. The development of the system is being done in a modular manner, operating around a central database of site investigation information and making use of general knowledge about geotechnical engineering organised in individual knowledge bases.

The inference mechanism of the system, as described in Chapter 3, allows inheritance and transitivity inferences as well as information retrieval facilities from the Ground and Tests knowledge bases. The rules developed are only structure dependent, they are not domain dependent. As both of the knowledge bases included in the system have been represented using the same structure, the same rules are used for their interrogation. This is considered to be an important feature as it makes the system general, providing the facility of searching any other knowledge base (independently of the knowledge being represented) as long as the knowledge it contains can be represented using this structure. For this reason, the inference rules implemented in the system could be considered as an Extended Inference Mechanism, on top of the built-in inference engine of PDC Prolog.

In the present version of the system information on units has not been incorporated for the attributes that take numerical values. The fact that the system is general, as discussed above, requires a general approach to be adopted in order to include such information in the knowledge bases. Although it is considered feasible to achieve this in the existing system, it has not been implemented due to time constraints.

Assistance in the selection of appropriate field tests is provided by the advisory rule that has been developed (rule *investigate*). This rule is sequentially model dependent, interrogating the two knowledge bases as required. The system, through this rule, is able to offer to the user possible suitable tests, that enable the derivation of the required geotechnical parameter with the desired reliability (both

of which are specified by the user). The system also provides the user with the applicability of each of these tests in the ground conditions that he/she specifies that the test is going to be performed in. Modified soil types are also considered (e.g. *dense* Sand, *silty* Clay, etc.). Moreover, the user is given the option to query additional information that has been included in the Tests Knowledge Base about these tests. The user can then compare the knowledge provided for each alternative test by the knowledge bases through the advisory rule, and also consider other factors, not incorporated in the system, that he/she finds relevant in order to make the final selection.

The advisory rule, as at present, does not perform any check for compatibility between the input values. For example, the user may input that the test is going to be performed in *coarse* soil and select the geotechnical parameter required as the *undrained shear strength*. Although these two values in reality are not compatible, the *investigate* rule will still produce possible solutions. An enhanced version of the rule should be able to recognise incompatible input values and inform the user about it.

In the current version of the system, the suitability of a specific test method is mainly based on the knowledge of the reliability with which the test is able to derive engineering soil parameters and of its applicability in different ground conditions. The ability of a test to relate to the type of project under consideration could also influence such a decision, as discussed in Chapter 4. As identified by Robertson (1985), Marsland (1986) and Orchant et al (1988) the appropriate tests should also be relevant to the particular problems being considered. For example, when deformation or strength parameters are required for the design, the stresses applied on the soil tested should be as close as possible to the stress conditions which occur on the soil in the full scale situation (Marsland, 1986). Knowledge of the foundation or earthwork problem being considered could also determine the degree of accuracy required in the determination of the relevant soil parameters. The relation between tests and type of construction has not been considered in the present implementation, due to time constraints.

In future developments of the system, this additional factor can also be incorporated without major changes in the current version. An additional knowledge base could be included without affecting the rest of the program, containing a hierarchy of possible types of construction, defining different applications. For each application included in this hierarchy, knowledge about the soil parameters required for the design, the reliability with which these parameters should be measured (for this type of application) and the test methods that are relevant to this type of construction can then be added. The advisory rule will be modified in order to accept, as input at the highest level, the application type under consideration and the type of the ground influenced by the construction. Searching initially the Applications Knowledge Base, the parameter(s) required to be measured, the tests that are likely to provide these parameters and the reliability required for their determination will be identified. The system, however, should also allow the user (if he/she desires) to specify the reliability with which the parameters need to be measured. The information derived could then be used as input to the existing advisory rule in order to identify which of the input tests provide the required reliability and the applicability of these tests in the type of the ground being considered. It is believed that major changes would not be required to enhance the existing advisory rule due to the modular way in which it has been implemented.

The user interface developed for the system is mainly menu driven providing ease of use to all potential users. On invoking the system the user is given the option to either query the knowledge bases, hence using the system as a learning tool or to seek assistance in the selection of in-situ tests, therefore using the system as decision support tool. The first function of the system when selected allows the activation of the rules forming the Extended Inference Mechanism. The user interface implemented for this option is domain independent as are the rules it triggers. The second function of the system activates the advisory rule. In this case the user interface developed is domain dependent as is the advisory rule.

An important feature of the prototype KBS, presented in this thesis, is considered to be the domain independent, extended inference mechanism and user interface implemented to be used for the

interrogation of the knowledge bases included in the system. This characteristic of the system allows the interrogation of any number of knowledge bases incorporated in it, relating to any domain. The Extended Inference Mechanism and the corresponding user interface could be considered as a basic expert system shell.

Possible improvements of the system in order to enhance the functionality of the expert system shell would entail the provision of a help facility to guide the non-familiar with the system user, a knowledge acquisition facility to enable the modification (addition or deletion) of the information incorporated in the existing knowledge bases, as well as the definition of additional knowledge bases. A hypertext facility in order to include additional information on the objects defined in the knowledge bases could also be useful.

In the existing system, such facilities would allow easy completion of the Ground and Tests knowledge bases as well as the incorporation of another knowledge base (e.g. the Applications Knowledge Base) by a domain expert and would make additional knowledge available to the user (such as detailed test procedures and information on the factors affecting the results of the various test methods).

During, as well as after, the development of any system, it is important to get feed back from potential users while consulting it. At present the only validation to the system has been done by colleagues, not necessarily familiar with the system. The general feeling was positive, stressing the fact that it seems to be a robust piece of software. After incorporating into the system the additional features discussed earlier in this chapter, the system should be validated by experienced and inexperienced engineers who are working in this area. In addition to that, when the knowledge included is complete, the system should be tested against case studies in order to check the recommendations of the system in real situations.

A final comment that is worth discussing concerns the comparative exercise carried out by implementing the 'Representing the Ground' application in the PROKAPPA system as well as in PDC Prolog. The purpose of this exercise, which is presented in Chapter 6, was to appreciate the differences between the two implementation schemes and to identify possible advantages and disadvantages that each one of them offers.

The interesting point of this exercise was that the implementation tools compared were a general purpose programming language (that could also be considered as a flexible expert system shell, as discussed by Marcellus (1989) and Reintjes (1992)) and an expert system development environment. Usually comparisons are carried out between tools of similar nature, for example between expert system shells (Adeli, 1988; Motamed et al, 1991). Through this exercise it was possible to identify a number of general factors that should be considered, among others, if an implementation tool has to be selected. These include the knowledge representation scheme and problem solving strategy required, type of machine available, cost of tool, time available for the implementation of the application under consideration and most importantly the aim of the implementation.

CHAPTER 8

CONCLUSIONS

In-situ testing has always played a major role in the art of geotechnical engineering. The developments achieved during the last decade and the growing interest of the engineering community in the use of field testing techniques during this time indicates that in-situ testing will play a progressively more dominant and important role in geotechnical engineering in the years to come.

The application of *knowledge-based system* technology in geotechnical engineering is a recent development. However, the existing KBSs demonstrate the potential of this technology to address a wide range of geotechnical engineering problems involving knowledge and experience, overcoming the limitations of algorithmic programming techniques.

A prototype Knowledge-Based System has been developed to assist in the selection of appropriate geotechnical in-situ tests. The system is an interactive, menu driven model-based system that performs two functions:

1. General querying of the knowledge bases,
2. Advising selection of in-situ tests.

The first option allows the user to interrogate separately the Ground and Tests knowledge bases that are included in the system, by activating the search rules which have been developed to provide inheritance and transitivity inferences as well as information retrieval facilities. These rules form an Extended Inference Mechanism on top of the built-in inference engine of PDC Prolog. The Extended Inference Mechanism, and the user interface implemented for it, form a basic expert system shell.

The second option provides assistance in the selection of appropriate field tests, by activating the advisory rule developed for this purpose. The system, through this rule, is able to offer to the user possible suitable tests, that enable the derivation of the required geotechnical parameter with the desired reliability (both of which are specified by the user). The system also provides to the user the applicability of each of these tests in the ground conditions that he/she specifies that the test is to be performed in. Modified soil types are also considered (e.g. *dense* Sand, *silty* Clay, etc.). Moreover, the user is given the option to query any other relevant information that has been included in the Tests Knowledge Base about these tests. The final selection is made by the user who can compare the information provided by the system on the alternative in-situ test methods, and consider at the same time additional factors not yet incorporated in the system.

The most difficult and time consuming task in the development of the system was the knowledge acquisition. The knowledge required was obtained in two ways: i) from technical literature and ii) from a small knowledge elicitation exercise in the form of a questionnaire. The representation scheme achieved is the same for both knowledge bases and allows modifications (additions or deletions) of the existing knowledge to be easily made.

A comparative exercise has also been performed by implementing the 'Representing the Ground' application in the PROKAPPA system as well as in PDC Prolog. Through this exercise, the differences between the two implementation schemes were appreciated and advantages and disadvantages that each one of them offers were identified. In addition, a number of general factors were identified (such as the knowledge representation scheme and problem solving strategy required, type of machine available, cost of tool, time available for the implementation of the application under considerartion and most importantly the aim of the implementation) which should be considered among others in order to select an appropriate implementation tool.

REFERENCES

Adams T.M., Christiano P. and Hendrickson C. (1989), *Some expert system applications in geotechnical engineering*, in Foundation Engineering: Current principles and practices, ASCE, New York, pp 885-902.

Adeli H. (1987), *Knowledge-Based Systems in Structural Engineering*, The Application of Artificial Intelligence Techniques to Civil and Structural Engineering, pp 71-78.

Adeli H. (ed.) (1988) *Expert Systems in Construction and Structural Engineering*, Chapman and Hall, London, England.

Alim S. and Munro J. (1987), *PROLOG-Based Expert Systems in Civil Engineering*, Proc. Instn. Civ. Engrs., Part 2, vol. 83, pp 1-14.

Allwood R.J., Stewart D.J and Trimble E.G. (1987), *Some Experiences from Evaluating Expert System Shell Programs and Some Potential Applications*, in Application of A.I. Techniques to Civil and Structural, (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 1-6.

Arockiasamy M., Radhakrishnan N., Sreenivasan G. and Lee S. (1991), *KBES Applications to the Selection and Design of Retaining Structures*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 391-402.

Asgian M.I., Arulmoli K., Miller W.O. and Sanjeevan K. (1988), *An expert system for diagnosis and treatment of dam seepage problems*, in Microcomputer knowledge-based expert systems in Civil Eng. (ed. Adeli H.), ASCE, New York, pp 118-126.

Attewell P.B. and Farmer I.W. (1976), *Principles of Engineering Geology*, Chapman and Hall Ltd, London.

Bageulin F., Jézéquel J.F. and Shields D.H. (1978), *The Pressuremeter and Foundation Engineering*, Series on Rock and Soil Mechanics, vol. 2, (1974/77), n. 4, Trans Tech Publications, 617 p.

Benchimol G., Levine P. and Pomerol J.C. (1987), *Developing Expert Systems for Business*, North Oxford Academic Publishers Ltd, Oxford, England.

Bokma A.F. (1991), *A Source Modelling System and its Use for Uncertainty Management*, PhD thesis, University of Durham.

Bradbury A. and Woodward R. (1988), *Turbo Prolog User's Handbook*, Version 2.0, McGraw - Hill Book Company Ltd, London.

Bratko I. (1990), *PROLOG Progarmming for Artificial Intelligence*, 2nd edition, Addison-Wesley.

British Standard 5930 (1981), *Code of Practice for Site Investigations*, British Standards Institution, London.

Carpaneto R. and Cremomini M.G. (1991), *Evaluation of Geotechnical Design Parameters by Expert System Techniques*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, no. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 422-433.

Chahine J.R. and Janson B.N. (1987), *Interfacing Databases with Expert Systems: a Retaining Wall Management Application*, Microcomputers in Civil Eng., vol. 2, n. 1, pp 19-38.

Chameau J-L. and Santamarina J.C. (1989), *Knowledge-Based System for Soil Improvement* , Journal of Computing in Civil Engineering, ASCE, vol. 3, n. 3, pp 253-267.

Child G.H. (1986), *Soil Descriptions-Quo Vadis?*, Site Investigation Practice: Assessing BS 5930, Geological Society, Engineering Geological Special Publication No. 2, (ed. Hawkins A. B.), pp 73-81.

Clocksin W.F. and Mellish C.S. (1981), *Programming in Prolog*, 1st edition, Srpinger-Verlag, Berlin.

Davey-Wilson I.E.G., May I.M., Tizani W., Alim S. and Munro J. (1988), *PROLOG-Based Expert Systems in Civil Engineering-Discussion*, Proc. Instn. Civ. Engrs., Part 2, vol. 85, pp 185-187.

Davey-Wilson I.E.G. (1989), *Development of a Prolog Based Expert System for Groundwater Control*, in Application of A.I. techniques to Civil and Structural Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 263-268.

Davey-Wilson I.E.G. (1991), *Geotechnical Laboratory Test Simulation using AI Techniques*, in Artificial Intelligence and Civil Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 119-124.

Davey-Wilson I.E.G. and May I.M. (1989), *Development of a knowledge-based system for the selection of groundwater control methods*, Computers and Geotechnics, 7, pp 189-203.

Denby B. and Kizil M.S. (1991), *Application of Expert Systems in Geotechnical Risk Assessment for Surface Coal Mine Design*, in International. Journal of Surface Mining and Reclamation., vol. 5, n. 2, pp 75-82.

Dimmick K., Bhatia S.K. and Hassett J. (1991), *Geotextile Edge Drain Design and Specification by Expert System*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 288-297.

Elton D.L. and Brown D.A. (1991), *Expert System for Driven Pile Selection*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 253-263.

Faure R.M., Mascarelli D., Zelfani M., Charveriat L., Gandar J. and Mosuro O. (1991), *XPENT: An Expert System for Slope Stability*, in Artificial Intelligence and Civil Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 143-147.

Feigenbaum E.A. (1983), *Knowledge Engineering: The Applied Side*, in Intelligent Systems; The Unprecedented Opportunity, (ed. Hayes J.E.), Ellis Horwood Limited, Chichester, England, pp 37-55.

Fookes P.G. and Vaughan P.R. (1986), *A Handbook of Engineering Geomorphology*, Surrey University Press.

Garigliano R. and Bokma A. (1992), *Uncertainty Management Through Source Control: A Heuristic Approach*, in Proceedings of Information Processing and Management of Uncertainty, Springer-Verlag.

Gillette D.R. (1991), *An Expert System for Estimating Soil Strength Parameters*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 276-287.

Hadipriono F.C., Diaz C.F. and Wolfe W.E. (1991), *Toward the Development of a Knowledge Base Expert System for Determining the Causes of Foundation Failures*, Proc. of Conf. on Computational Structures Technology, CIVIL-COMP Press, Herriot-Watt Univ., Edinbourgh.

Hadipriono F.C. and Wolfe W.E. (1991), *Repairability Assessment of Damaged Foundations*, in Artificial Intelligence and Civil Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 137-141.

Halim I.S., Tang W.H. and Garrett J.H.Jr. (1991), *Knowledge-Assisted Interactive Probabilistic Site Characterization*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 264-275.

Hutchinson P.J., Rosenman M.A. and Gero J.S.(1987), *RETWALL: An Expert System for the Selection and Preliminary Design of Earth Retaining Structures*, Knowledge-based systems, vol. 1, n. 1, pp 11-23.

ISSMFE (1988), Technical Committee on Penetration Testing, *International Reference Test Procedures*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, De Ruiter J. (ed.), Balkema A.A., Rotterdam, vol. 1, pp 3-90.

Juang C.H. and Lee D.H. (1989), *Development of an Expert System for Rock Mass Classification*, Civil Engineering Systems, vol.6, pp 147-156.

Konigsberger H.K. and De Bruyn F.W.G.M. (1990), *Prolog from the Beginning*, McGraw-Hill Book Company (UK) Limited, London, England.

Κρικετος Β.Γ. και Παστρας Κ.Σ. (1991), Εγχειριδιο Εισαγωγης στα Εμπειρα Συστηματα, Εκδοθηκε απο την Εταιρεια Αναπτυξης της Ναυτικης Τεχνολογιας Α.Ε., Αθηνα, Ελλαδα.

Maher M.H. and Williams T.P. (1991), *A Hybrid Expert System for Design with Geosynthetics*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 241-252.

Maher M.L. (ed.) (1987) *Expert Systems for Civil Engineers*, American Society of Civil Engineers, New York, U.S.A.

Maher M.L. and Allen R. (1987), *Expert Systems Components*, in Expert Systems for Civil Engineers, (ed. Maher M.L.), American Society of Civil Engineers, New York, U.S.A., pp 3-14.

Manby C.N.D. and Wakeling T.R.M. (1990), *Developments in Soft-Ground Drilling, Sampling and In-Situ Testing*, Trans. Institution of Mining and Metallurgy, Section A: Min. Industry, vol. 99, May-August, pp A91-A97.

Marcellus D.H. (1989), *Expert Systems Programming in Turbo Prolog*, Prentice-Hall,Inc, New Jersey.

Marsland A. (1986), *The Choice of Test Methods in Site Investigation*, Site Investigation Practice: Assessing BS 5930, Geological Society, Engineering Geological Special Publication No. 2, (ed. Hawkins A. B.), pp 289-298.

Meyer S. (1992), *Preliminary Foundation Design Using EDESYN*, Optimisation and Artificial Intelligence in Civil and Structural Engineering, (ed. Topping B.H.V.), Kluwer Academic Publishers, Dordrecht, vol. 2, pp 333-354.

Mi Z. and Jieliang P. (1989), *An Expert System of Predicting and Preventing Surface Settlement Caused by Shield-Driven Tunneling in City*, Proceedings of International Conference'89 on Expert Systems in Engineering Applications, Huazhong University of Science and Technology Press, pp 466-472.

Mitchell J.K. (1988), *New developments in penetration tests and equipment*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 245-262.

Motamed F., Salazar G. and D'Andrea R. (1991), *An Expert System for Preliminary Ground Improvement Selection*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 379-390.

Mullarkey P.W. (1985) *CONE - An Expert System for Interpretation of Geotechnical Characterization Data from Cone Penetrometers*, PhD thesis, Carnegie-Mellon University, Pittsburgh, PA.

Mullarkey P.W. (1986), *A Geotechnical KBS Using Fuzzy Logic*, in Applications of A.I. in Engineering Problems, 1st Int. Conf., Southampton University (eds. Sriram D. and Adey R.), Springer-Verlag, vol. 2, pp 847-859.

Mullarkey P.W. (1987), *Languages and Tools for Building Expert Systems*, in Expert Systems for Civil Engineers, (ed. Maher M.L.), American Society of Civil Engineers, New York, U.S.A., pp 15-34.

Mullarkey P.W. and Fenves S.J. (1986), *Fuzzy logic in a geotechnical knowledge based system : CONE*, Civ. Eng. Syst vol. 3, n. 2, pp 58-81.

Norbury D.R., Child G.H. and Spink T.W. (1986), *A Critical Review of Section 8 (BS5930)-Soil and Rock Description*, Site Investigation Practice: Assessing BS 5930, Geological Society, Engineering Geological Special Publication No. 2, (ed. Hawkins A. B.), pp 331-342.

Norkin D.D. (1985), *Expert System for Geotechnical Site Characterisation*, MSc dissertation, Carnegie-Mellon University, Dept. of Civil Engineering.

Oliphant J. and Blockley D.I. (1989), *Knowledge Based System: Advisor on the Selection of Earth Retaining Structures*, Proc. 4th Int. Conf. A.I. techniques and applications for Civil and Structural Engineering Computing, CIVIL-COMP Press, Edinburgh, pp 253-262.

Orchant C.J., Kulhawy F.H. and Trautmann C.H. (1988), *Reliability-Based Foundation Design for Transmission Line Structures: Critical Evaluation of In-Situ Test Methods*, Report EL-5507, vol. 2, Electric Power Research Institute, Palo Alto, 214 p.

Parikh S.A. and Kameswara Rao N.S.V. (1991), *An Expert System for Civil Engineering Applications*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 413-421.

PDC Prolog Reference Guide (1992), Version 3.30, Prolog Development Center, Copenhagen, Denmark.

PDC Prolog Toolbox (1990), Version 3.20, Prolog Development Center, Copenhagen, Denmark.

PDC Prolog User's Guide (1992), Version 3.30, Prolog Development Center, Copenhagen, Denmark.

Pearse R., Rosenbaum M. and Hammond P. (1986), *The Evaluation of Proposed Road Corridors by the Use of an Expert System*, in Applications of A.I. in Engineering Problems, 1st Int. Conf., Southampton University (eds. Sriram D. and Adey R.), Springer-Verlag, vol. 2, pp 719-730.

PROKAPPA User's Guide (1991), IntelliCorp, Inc. Version 2.0, Public. No: PK2.0-UG-2, October.

Rashad M.M., Yehia N.A.B., Bazaraa A.S. and Dessouki A.I. (1991), *Foundcon : A Conceptual Model for the Integrated Knowledge-Based CAD Systems for Foundation Design*, in Artificial Intelligence and Civil Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 125-135.

Rehak D.R., Christiano P.P. and Norkin D.D. (1985), *SITECHAR: An Expert System Component of a Geotechnical Site Characterization Workbench*, in Applications of knowledge-based systems to engineering analysis and design (ed. Dym C.L.), Am. Soc. Mech. Eng., New York, pp 117-133.

Reintjes P. (1992), *Elegant Technologies*, Proc. of International Conference The Practical Application of Prolog, April, London, vol. 2, Invited Paper.

Righetti G.A. and Cremonini M.G. (1988), *The DAISY Environment and the Expert System GUESS*, in Artificial Intelligence in Engineering: Diagnosis and Learning, (ed. Gero J.S.), Elsevier, Amsterdam.

Robertson P.K. (1985), *In Situ Testing and its Application to Foundation Engineering*, Soil Mechanics Series No. 91, University of British Columbia, Department of Civil Engineering, Vancouver, B.C., 212 p.

Robertson P.K. (1986), *In Situ Testing and its Application to Foundation Engineering*, Canadian Geotechnical Journal, vol. 23, pp 573-594.

Rosenman M.A., Balachandran B.M. and Gero J.S. (1989), *The Place of Expert Systems in Civil Engineering*, in Civil Engineering Systems, Chapman and Hall Ltd, vol. 6, n. 1-2, pp 11-20.

Rowlinson S. (1989), *Knowledge Based Systems: Potential in Design and Management*, Proc. 4th Int. Conf. A.I. techniques and applications for Civil and Structural Engineering Computing, CIVIL-COMP Press, Edinburgh, pp 21-26.

Ruggieri C. and Sancassami M. (1992), *A Set of Prolog Programming Tools*, Proc. of International Conference The Practical Application of Prolog, April, London, vol. 1, Virtual Languages.

Santamarina J.C. and Chameau J.L. (1987), *Expert Systems for Geotechnical Engineers*, Jnl. Computing in Civil Eng., vol. 1, n. 4, pp 241-252.

Shyu G.C. and Hryciw R.D. (1991), *SOLES: A Knowledge-Based Soil Liquefaction Potential Evaluation System*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, n. 27, (eds. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 403-412.

Sieh D., King D. and Gientke F. (1988), *Dam Seepage Analysis Using Artificial Intelligence*, Planning Now for Irrigation and Drainage in the 21st Century, ASCE, pp 417-422.

Siller J.T. (1987), *Expert Systems in Geotechnical Engineering*, in Expert Systems for Civil Engineers: Technology and Applications, (ed. Maher M.L.), ASCE, NY, pp 77-84.

Smith I.G.N. and Oliphant J. (1991), *The Use of a Knowledge-Based System for Civil Engineering Site Investigations*, in Artificial Intelligence and Civil Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 105-112.

Smith I.G.N. and Oliphant J. (1992), *The Use of a Knowledge-Based System for Civil Engineering Site Investigations*, (extended) in Computer Systems in Engineering, (to be published).

Smith R.G. and Baker J.D. (1983), *DIPMETER ADVISOR System*, Proc. 8th Int. Joint Conf. A.I., Karlsruhe, Germany, pp 122-129.

Spink T.W.and Norbury D.R. (1991), *The Engineering Geological Description of Weak Rocks and Overconsolidated Soils*, Proceedings of the 26th Annual Conference of Engineering Geology Group of the Geological Society of London.

Stuckrath L.A. and Grivas D.A. (1990), *A Knowledge-Based System for Bridge Foundation Selection*, OECD Workshop on Knowledge-Based ExpertSystems in Transportation Part 1, in VTT Symposium (Valtion Teknillinen Tutkimuskeskus), Technical Research Center of Finland, Espoo, vol. 1, n. 116, pp 283-297.

Toll D.G. (1990), *Do Geotechnical Engineers need Expert Systems?*, Ground Engineering, vol. 23, n. 23, pp 32-36.

Toll D.G., Moula M. and Vaptismas N. (1991), *Representing the Engineering Description of Soils in Knowledge Based Systems*, AIENG 6, Proc. of 6th Int. Conf. on Application of A.I. in Engineering, University of Oxford, July, pp 113-118.

Toll D.G., Moula M., Oliver A., and Vaptismas N. (1992), *A Knowledge Based Sustem for Interpreting Site Investigation Information*, International Conference on Geotechnics and Computers, Paris, September, pp 607-614.

Tomlinson M.J. (1986), *Foundation Design and Construction*, Fifth Edition, Longman Scientific and Technical, Essex, England.

Turbo PROLOG - Owner's Handbook (1986), Borland International, Scotts Valley, CA.

Vaptismas N. (1992), *A Methodology for the Interpretation of Ground Conditions from Borehole Information*, PhD thesis, University of Durham.

Weltman A.J. and Head J.M. (1983), *Site Investigation Manual*, CIRIA Special Publication 25, Construction Industry Research and Information Association, London.

Wilson J.L, Mikroudis G.K. and Fang H.Y. (1987), *GEOTOX: A Knowledge-Based System for Hazardous Site Evaluation*, in Artificial Intelligence, Computational Mechanics Publications, vol. 2, n. 1, pp 23-32.

Wislocki A.P. and Bentley S.P. (1989), *An Expert System for Landslide Hazard and Risk Assessment*, Proc. 4th Int. Conf. A.I. techniques and applications for Civil and Structural Engineering Computing, CIVIL-COMP Press, Edinburgh, pp 249-252.

Wong K.C., Poulos H.G. and Thorne C.P. (1989), *Site Classification by Expert Systems*, Computers and Geotechnics, 8, pp 133-156.

Wong K.C., Poulos H.G. and Thorne C.P. (1991), *Development of Expert Systems for Pile Foundation Design*, Trans. Inst. of Engineers, vol.CE33, n.2, pp 119-127.

Wroth C.P. (1984), *The Interpretation of In-situ Soil Tests*, Geotechnique 34, n.. 4, pp 449-489.

Yehia N.A.B. and El-Hajj A.H. (1987), *A Knowledge Based Approach for the Design of Spread Footings*, in Application of A.I. techniques to Civil and Structural Engineering (ed. Topping B.H.V.), CIVIL-COMP Press, Edinburgh, pp 119-124.

Zheng H., Mikroudis G.K., Pamukcu S. and Hu Z.X. (1989), *BABE: An Expert System for Structural Design of Bridge Abutments and Piers*, in Computer Utilization in Structural Engineering, Proceedings of the sessions at Structures Congress '89, ASCE, NY, pp 372-381.

APPENDIX A

PROLOG PROGRAM

```
/****************************
*      File KNOWBASE.PRO     *
****************************/
```

/ This file contains the two knowledge bases required to be incorporated in the system, the Ground Knowledge Base and the Tests Knowledge Base. */*

```
domains


list=symbol*
fact=fact(list)
val=val(list, fact)
vallist=val*
att=att(symbol, vallist)
attlist=att*



database - knowledge_base


class(symbol, list, attlist)
modifier(symbol, attlist)



clauses

                                /*  GROUND Hierarchy  */


class(ground, [soil, rock],
        []).
class(soil, [non_organic, organic, man_made],
        [att(ground_type,
                [val([soil], fact([]))])]).
class(rock, [soft_rock, hard_rock],
        [att(ground_type,
                [val([rock], fact([]))]),
         att(uniaxial_compressive_strength,
                [val(["600", "400000"], fact([]))])]).
class(non_organic, [granular, cohesive],
        [att(soil_nature,
                [val([non_organic], fact([]))]),
         att(grain_size,
                [val(["0", "2000"], fact([]))])]).
```

```
class(organic, [organic_granular, organic_cohesive, peat],
        [att(soil_nature,
                [val([organic], fact([]))])]).
class(man_made, [fill, waste],
        [att(soil_nature,
                [val([man_made], fact([]))])]).
class(soft_rock, [],
        [att(rock_name,
                [val([soft_rock], fact([]))]),
         att(uniaxial_compressive_strength,
                [val(["0.6", "12.5"], fact([]))])]).
class(hard_rock, [],
        [att(rock_name,
                [val([hard_rock], fact([]))]),
         att(uniaxial_compressive_strength,
                [val(["12.5", "400"], fact([]))])]).
class(granular, [very_coarse, coarse, granular_fine],
        [att(soil_character,
                [val([granular], fact([]))]),
         att(grain_size,
                [val(["0.002", "2000"], fact([]))])]).
class(cohesive, [fine],
        [att(soil_character,
                [val([cohesive], fact([]))]),
         att(grain_size,
                [val(["0", "0.06"], fact([]))]),
         att(liquid_limit,
                [val(["0", "200"], fact([]))])]).
class(organic_granular, [organic_coarse, organic_granular_fine],
        [att(soil_character,
                [val([organic_granular], fact([]))]),
         att(grain_size,
                [val(["0.002", "2"], fact([]))])]).
class(organic_cohesive, [organic_fine],
        [att(soil_character,
                [val([organic_cohesive], fact([]))]),
         att(grain_size,
                [val(["0", "0.06"], fact([]))]),
         att(liquid_limit,
                [val(["0", "200"], fact([]))])]).
class(peat, [],
        [att(soil_name,
                [val([peat], fact([]))])]).
class(fill, [],
        [att(soil_name,
                [val([fill], fact([]))])]).
class(waste, [],
        [att(soil_name,
                [val([waste], fact([]))])]).
class(very_coarse, [boulders, cobbles],
        [att(coarseness,
                [val([very_coarse], fact([]))]),
         att(grain_size,
                [val(["60", "2000"], fact([]))])]).
```

```
class(coarse, [gravel, sand],
        [att(coarseness,
                [val([coarse], fact([]))]),
         att(grain_size,
                [val(["0.06", "60"], fact([]))])]).
class(granular_fine, [silt],
        [att(coarseness,
                [val([granular_fine], fact([]))]),
         att(grain_size,
                [val(["0.002", "0.06"], fact([]))])]).
class(fine, [silt, clay],
        [att(coarseness,
                [val([fine], fact([]))]),
         att(grain_size,
                [val(["0", "0.06"], fact([]))])]).
class(organic_coarse, [organic_sand],
        [att(coarseness,
                [val([organic_coarse], fact([]))]),
         att(grain_size,
                [val(["0.06", "2"], fact([]))])]).
class(organic_granular_fine, [organic_silt],
        [att(coarseness,
                [val([organic_granular_fine], fact([]))]),
         att(grain_size,
                [val(["0.002", "0.06"], fact([]))])]).
class(organic_fine, [organic_silt, organic_clay],
        [att(coarseness,
                [val([organic_fine], fact([]))]),
         att(grain_size,
                [val(["0", "0.06"], fact([]))])]).
class(boulders, [],
        [att(soil_name,
                [val([boulders], fact([]))]),
         att(grain_size,
                [val(["200", "2000"], fact([]))])]).
class(cobbles, [],
        [att(soil_name,
                [val([cobbles], fact([]))]),
         att(grain_size,
                [val(["60", "200"], fact([]))])]).
class(gravel, [],
        [att(soil_name,
                [val([gravel], fact([]))]),
         att(grain_size,
                [val(["2", "60"], fact([]))])]).
class(sand, [],
        [att(soil_name,
                [val([sand], fact([]))]),
         att(grain_size,
                [val(["0.06", "2"], fact([]))])]).
```

```
class(silt, [],
        [att(soil_name,
                [val([silt], fact([]))]),
         att(grain_size,
                [val(["0.002", "0.06"], fact([]))])]).
class(clay, [],
        [att(soil_name,
                [val([clay], fact([]))]),
         att(grain_size,
                [val(["0", "0.002"], fact([]))])]).
class(organic_sand, [],
        [att(soil_name,
                [val([organic_sand], fact([]))]),
         att(grain_size,
                [val(["0.06", "2"], fact([]))])]).
class(organic_silt, [],
        [att(soil_name,
                [val([organic_silt], fact([]))]),
         att(grain_size,
                [val(["0.002", "0.06"], fact([]))])]).
class(organic_clay, [],
        [att(soil_name,
                [val([organic_clay], fact([]))]),
         att(grain_size,
                [val(["0", "0.002"], fact([]))])]).
```

/ TESTS Hierarchy */*

```
class(tests, [in_situ_tests, large_scale_field_tests, back_analysis_tests,
                        laboratory_tests],
        []).
class(in_situ_tests, [penetration_tests, special_penetrometer_tests,
                        pressuremeter_tests, in_situ_stress_measurement_tests,
                        shear_tests, bearing_tests, in_situ_density_tests,
                        permeability_tests, geophysical_surveying_tests],
        [att(test_category,
                [val([in_situ_tests], fact([]))])]).
class(penetration_tests, [standard_penetration_test, dynamic_probing_test,
                        cone_penetration_test, weight_sounding_test,
                        static_dynamic_penetration_test],
        [att(test_nature,
                [val([penetration_tests], fact([]))]),
         att(test_objective,
                [val([logging_test_method], fact([]))])]).
class(special_penetrometer_tests, [expansion_penetration_tests, seismic_cone_test,
                        lateral_stress_cone_test, density_probe_tests,
                        electrical_conductivity_cone_test,
                        thermal_conductivity_cone_test,
                        acoustic_cone_test, vibratory_cone_test],
        [att(test_nature,
                [val([special_penetrometer_tests], fact([]))])]).
```

```
class(pressuremeter_tests, [menard_type_pressuremeter_test,
                            push_in_pressuremeter_test,
                            self_boring_pressuremeter_test],
        [att(test_nature,
                [val([pressuremeter_tests], fact([]))]),
         att(test_objective,
                [val([specific_test_method], fact([]))])]).
class(in_situ_stress_measurement_tests, [total_stress_cell_test,
                            iowa_stepped_blade_test,
                            hydraulic_fracturing_test,
                            self_boring_ko_meter_test],
        [att(test_nature,
                [val([in_situ_stress_measurement_tests], fact([]))]),
         att(test_objective,
                [val([specific_test_method], fact([]))])]).
class(shear_tests, [vane_test, self_boring_vane_test, borehole_shear_test,
                            in_situ_shear_test],
        [att(test_nature,
                [val([shear_tests], fact([]))]),
         att(test_objective,
                [val([specific_test_method], fact([]))])]).
class(bearing_tests, [plate_loading_tests, screw_plate_test,
                            self_boring_plate_test, pressurized_chamber_test,
                            in_situ_california_bearing_ratio_test],
        [att(test_nature,
                [val([bearing_tests], fact([]))]),
         att(test_objective,
                [val([specific_test_method], fact([]))])]).
class(in_situ_density_tests, [sand_replacement_tests, core_cutter_test,
                            weight_in_water_test, water_replacement_test,
                            rubber_balloon_test, nuclear_tests],
        [att(test_nature,
                [val([in_situ_density_tests], fact([]))]),
         att(test_objective,
                [val([specific_test_method], fact([]))])]).
class(permeability_tests, [borehole_permeability_tests,
                            self_boring_permeameter_test, pumping_tests],
        [att(test_nature,
                [val([permeability_tests], fact([]))]),
         att(test_objective,
                [val([specific_test_method], fact([]))])]).
class(geophysical_surveying_tests, [seismic_tests, resistivity_test,
                            gravimetric_test, magnetic_test],
        [att(test_nature,
                [val([geophysical_surveying_tests], fact([]))]),
         att(test_objective,
                [val([logging_test_method], fact([]))])]).
class(standard_penetration_test, [],
        [att(test_name,
                [val([standard_penetration_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([medium], fact([]))])]).
```

```
class(dynamic_probing_test, [dynamic_probing_light_test,
                             dynamic_probing_medium_test,
                             dynamic_probing_heavy_test,
                             dynamic_probing_superheavy_test],
        [att(test_group,
                [val([dynamic_probing_test], fact([]))])]).
class(cone_penetration_test, [mechanical_penetrometer_friction_test,
                             electrical_cone_penetration_test],
        [att(test_group,
                [val([cone_penetration_test], fact([]))])]).
class(weight_sounding_test, [],
        [att(test_name,
                [val([weight_sounding_test], fact([]))]),
         att(test_frequency,
                [val([], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(static_dynamic_penetration_test, [],
        [att(test_name,
                [val([static_dynamic_penetration_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(expansion_penetration_tests, [flat_plate_dilatometer_test,
                             cone_pressuremeter_test],
        [att(test_group,
                [val([expansion_penetration_tests], fact([]))])]).
class(seismic_cone_test, [],
        [att(test_name,
                [val([seismic_cone_test], fact([]))]),
         att(test_objective,
                [val([combined_test_method], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(lateral_stress_cone_test, [],
        [att(test_name,
                [val([lateral_stress_cone_test], fact([]))]),
         att(test_objective,
                [val([combined_test_method], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(density_probe_tests, [nuclear_density_probe_test,
                             electrical_density_probe_test],
        [att(test_group,
                [val([density_probe_tests], fact([]))]),
         att(test_objective,
                [val([combined_test_method], fact([]))])]).
```

```
class(electrical_conductivity_cone_test, [],
        [att(test_name,
                [val([electrical_conductivity_cone_test], fact([]))]),
         att(test_objective,
                [val([combined_test_method], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(thermal_conductivity_cone_test, [],
        [att(test_name,
                [val([thermal_conductivity_cone_test], fact([]))]),
         att(test_objective,
                [val([combined_test_method], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(acoustic_cone_test, [],
        [att(test_name,
                [val([acoustic_cone_test], fact([]))]),
         att(test_objective,
                [val([logging_test_method], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(vibratory_cone_test, [],
        [att(test_name,
                [val([vibratory_cone_test], fact([]))]),
         att(test_objective,
                [val([combined_test_method], fact([]))]),
         att(test_frequency,
                [val([], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(menard_type_pressuremeter_test, [],
        [att(test_name,
                [val([menard_type_pressuremeter_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([medium], fact([]))])]).
class(push_in_pressuremeter_test, [],
        [att(test_name,
                [val([push_in_pressuremeter], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
```

```
class(self_boring_pressuremeter_test, [],
        [att(test_name,
                [val([self_boring_pressumeter_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([high], fact([]))])]).
class(total_stress_cell_test, [],
        [att(test_name,
                [val([total_stress_cell_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(iowa_stepped_blade_test, [],
        [att(test_name,
                [val([iowa_stepped_blade_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(hydraulic_fracturing_test, [],
        [att(test_name,
                [val([hydraulic_fracturing_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(self_boring_ko_meter_test, [],
        [att(test_name,
                [val([self_boring_ko_meter_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(vane_test, [],
        [att(test_name,
                [val([vane_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([medium], fact([]))])]).
class(self_boring_vane_test, [],
        [att(test_name,
                [val([self_boring_vane_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
```

```
class(borehole_shear_test, [],
        [att(test_name,
                [val([borehole_shear_test], fact([]))]),
         att(test_frequency,
                [val([], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(in_situ_shear_test, [],
        [att(test_name,
                [val([in_situ_shear_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(plate_loading_tests, [],
        [att(test_name,
                [val([plate_loading_tests], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(screw_plate_test, [],
        [att(test_name,
                [val([screw_plate_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(self_boring_plate_test, [],
        [att(test_name,
                [val([self_boring_plate_test], fact([]))]),
         att(test_frequency,
                [val([], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(pressurized_chamber_test, [],
        [att(test_name,
                [val([pressurized_chamber_test], fact([]))]),
         att(test_frequency,
                [val([], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(in_situ_california_bearing_ratio_test, [],
        [att(test_name,
                [val([in_situ_california_bearing_ratio_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(sand_replacement_tests, [small_pouring_cylinder_test,
                               large_pouring_cylinder_test, scoop_test],
        [att(test_group,
                [val([sand_replacement_tests], fact([]))])]).
```

```
class(core_cutter_test, [],
        [att(test_name,
                [val([core_cutter_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(weight_in_water_test, [],
        [att(test_name,
                [val([weight_in_water_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(water_replacement_test, [],
        [att(test_name,
                [val([water_replacement_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(rubber_balloon_test, [],
        [att(test_name,
                [val([rubber_balloon_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(nuclear_tests, [backscatter_test, direct_transmission_test, air_gap_test],
        [att(test_group,
                [val([nuclear_tests], fact([]))])]).
class(borehole_permeability_tests, [variable_head_test, constant_head_test],
        [att(test_group,
                [val([borehole_permeability_tests], fact([]))])]).
class(self_boring_permeameter_test, [],
        [att(test_name,
                [val([self_boring_permeameter_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(pumping_tests, [],
        [att(test_name,
                [val([pumping_tests], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(seismic_tests, [seismic_refraction_test, seismic_reflection_test,
                        seismic_cross_hole_test, seismic_down_hole_test,
                        surface_wave_test],
        [att(test_group,
                [val([seismic_tests], fact([]))])]).
```

```
class(resistivity_test, [],
        [att(test_name,
                [val([resistivity_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([low], fact([]))])]).
class(gravimetric_test, [],
        [att(test_name,
                [val([gravimetric_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(magnetic_test, [],
        [att(test_name,
                [val([magnetic_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(dynamic_probing_light_test, [],
        [att(test_name,
                [val([dynamic_probing_light_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(dynamic_probing_medium_test, [],
        [att(test_name,
                [val([dynamic_probing_medium_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(dynamic_probing_heavy_test, [],
        [att(test_name,
                [val([dynamic_probing_heavy_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(dynamic_probing_superheavy_test, [],
        [att(test_name,
                [val([dynamic_probing_superheavy_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
```

```
class(mechanical_penetrometer_friction_test, [],
        [att(test_name,
                [val([mechanical_penetrometer_friction_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([low], fact([]))])]).
class(electrical_cone_penetration_test, [electrical_penetrometer_friction_test,
                        piezocone_test, piezocone_friction_test],
        [att(test_type,
                [val([electrical_cone_penetration_test], fact([]))])]).
class(flat_plate_dilatometer_test, [],
        [att(test_name,
                [val([flat_plate_dilatometer_test], fact([]))]),
         att(test_objective,
                [val([logging_test_method], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([low], fact([]))])]).
class(cone_pressuremeter_test, [],
        [att(test_name,
                [val([cone_pressuremeter_test], fact([]))]),
         att(test_objective,
                [val([combined_test_method], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(nuclear_density_probe_test, [],
        [att(test_name,
                [val([nuclear_density_probe_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(electrical_density_probe_test, [],
        [att(test_name,
                [val([electrical_density_probe_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(small_pouring_cylinder_test, [],
        [att(test_name,
                [val([small_pouring_cylinder_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
```

```
class(large_pouring_cylinder_test, [],
        [att(test_name,
                [val([large_pouring_cylinder_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(scoop_test, [],
        [att(test_name,
                [val([scoop_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(backscatter_test, [],
        [att(test_name,
                [val([backscatter_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(direct_transmission_test, [],
        [att(test_name,
                [val([direct_transmission_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(air_gap_test, [],
        [att(test_name,
                [val([air_gap_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(variable_head_test, [rising_head_test, falling_head_test],
        [att(test_type,
                [val([variable_head_test], fact([]))])]).
class(constant_head_test, [],
        [att(test_name,
                [val([constant_head_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(seismic_refraction_test, [],
        [att(test_name,
                [val([seismic_refraction_test], fact([]))]),
         att(test_frequency,
                [val([rouitne], fact([]))]),
         att(unit_cost,
                [val([low], fact([]))])]).
```

```
class(seismic_reflection_test, [],
        [att(test_name,
                [val([seismic_reflection_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([low], fact([]))])]).
class(seismic_cross_hole_test, [],
        [att(test_name,
                [val([seismic_cross_hole_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(seismic_down_hole_test, [],
        [att(test_name,
                [val([seismic_down_hole_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(surface_wave_test, [],
        [att(test_name,
                [val([surface_wave_test], fact([]))]),
         att(test_frequency,
                [val([special_purpose], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(electrical_penetrometer_friction_test, [],
        [att(test_name,
                [val([electrical_cone_resistance_friction_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([low], fact([]))])]).
class(piezocone_test, [],
        [att(test_name,
                [val([piezocone_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([medium], fact([]))])]).
class(piezocone_friction_test, [],
        [att(test_name,
                [val([piezocone_friction_test], fact([]))]),
         att(test_frequency,
                [val([less_common], fact([]))]),
         att(unit_cost,
                [val([medium], fact([]))])]).
```

```
class(rising_head_test, [],
        [att(test_name,
                [val([rising_head_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).
class(falling_head_test, [],
        [att(test_name,
                [val([falling_head_test], fact([]))]),
         att(test_frequency,
                [val([routine], fact([]))]),
         att(unit_cost,
                [val([], fact([]))])]).


                    /* Detailed Representation of Dominant Soil Types */


modifier(gravel,
        [att(grain_size,
                [val(["20", "60"], fact([coarse])),
                 val(["6", "20"], fact([medium])),
                 val(["2", "6"], fact([fine]))]),
         att("N_value",
                [val(["0", "4"], fact([very_loose])),
                 val(["4", "10"], fact([loose])),
                 val(["10", "30"], fact([medium_dense])),
                 val(["30", "50"], fact([dense])),
                 val(["50", "100"], fact([very_dense]))]),
         att(coefficient_of_permeability,
                [val(["10e-3", "1"], fact([high_permeability]))]),
         att(secondary_percent,
                [val(["5", "20"], fact([sandy])),
                 val(["5", "15"], fact([silty])),
                 val(["5", "15"], fact([clayey]))])]).
modifier(sand,
        [att(grain_size,
                [val(["0.6", "2"], fact([coarse])),
                 val(["0.2", "0.6"], fact([medium])),
                 val(["0.06", "0.2"], fact([fine]))]),
         att("N_value",
                [val(["0", "4"], fact([very_loose])),
                 val(["4", "10"], fact([loose])),
                 val(["10", "30"], fact([medium_dense])),
                 val(["30", "50"], fact([dense])),
                 val(["50", "100"], fact([very_dense]))]),
         att(coefficient_of_permeability,
                [val(["10e-5", "10e-3"], fact([medium_permeability])),
                 val(["10e-7", "10e-5"], fact([low_permeability]))]),
         att(coefficient_of_volume_compressibility,
                [val(["0", "0.05"], fact([very_low_compressibility]))]),
```

```
        att(secondary_percent,
                [val(["5", "20"], fact([gravelly])),
                 val(["5", "15"], fact([silty])),
                 val(["5", "15"], fact([clayey]))])]).
modifier(silt,
        [att(grain_size,
                [val(["0.02", "0.06"], fact([coarse])),
                 val(["0.006", "0.02"], fact([medium])),
                 val(["0.002", "0.006"], fact([fine]))]),
         att(liquid_limit,
                [val(["0", "35"], fact([low_plasticity])),
                 val(["35", "50"], fact([intermediate_plasticity])),
                 val(["50", "70"], fact([high_plasticity])),
                 val(["70", "90"], fact([very_high_plasticity])),
                 val(["90", "200"], fact([extremely_high_plasticity]))]),
         att("N_value",
                [val(["0", "4"], fact([very_loose])),
                 val(["4", "10"], fact([loose])),
                 val(["10", "30"], fact([medium_dense])),
                 val(["30", "50"], fact([dense])),
                 val(["50", "100"], fact([very_dense]))]),
         att(undrained_shear_strength,
                [val(["0", "20"], fact([very_soft])),
                 val(["20", "40"], fact([soft])),
                 val(["40", "75"], fact([firm])),
                 val(["75", "150"], fact([stiff])),
                 val(["150", "300"], fact([very_stiff]))]),
         att(coefficient_of_permeability,
                [val(["10e-7", "10e-5"], fact([low_permeability])),
                 val(["10e-9", "10e-7"], fact([very_low_permeability]))]),
         att(coefficient_of_volume_compressibility,
                [val(["0", "0.05"], fact([very_low_compressibility]))]),
         att(secondary_percent,
                [val(["35", "65"], fact([gravelly])),
                 val(["35", "65"], fact([sandy]))])]).
modifier(clay,
        [att(liquid_limit,
                [val(["0", "35"], fact([low_plasticity])),
                 val(["35", "50"], fact([intermediate_plasticity])),
                 val(["50", "70"], fact([high_plasticity])),
                 val(["70", "90"], fact([very_high_plasticity])),
                 val(["90", "200"], fact([extremely_high_plasticity]))]),
         att(undrained_shear_strength,
                [val(["0", "20"], fact([very_soft])),
                 val(["20", "40"], fact([soft])),
                 val(["40", "75"], fact([firm])),
                 val(["75", "150"], fact([stiff])),
                 val(["150", "300"], fact([very_stiff]))]),
         att(coefficient_of_permeability,
                [val(["0", "10e-9"], fact([practically_impervious]))]),
```

```
        att(coefficient_of_volume_compressibility,
                [val(["0", "0.05"], fact([very_low_compressibility])),
                 val(["0.05", "0.1"], fact([low_compressibility])),
                 val(["0.1", "0.3"], fact([medium_compressibility])),
                 val(["0.3", "1.5"], fact([high_compressibility]))]),
        att(secondary_percent,
                [val(["35", "65"], fact([gravelly])),
                 val(["35", "65"], fact([sandy]))])]).
modifier(organic_sand,
        [att(grain_size,
                [val(["0.6", "2"], fact([coarse])),
                 val(["0.2", "0.6"], fact([medium])),
                 val(["0.06", "0.2"], fact([fine]))]),
         att("N_value",
                [val(["0", "4"], fact([very_loose])),
                 val(["4", "10"], fact([loose])),
                 val(["10", "30"], fact([medium_dense])),
                 val(["30", "50"], fact([dense])),
                 val(["50", "100"], fact([very_dense]))]),
         att(coefficient_of_permeability,
                [val(["10e-5", "10e-3"], fact([medium_permeability])),
                 val(["10e-7", "10e-5"], fact([low_permeability]))]),
         att(coefficient_of_volume_compressibility,
                [val(["0", "0.05"], fact([very_low_compressibility]))]),
         att(secondary_percent,
                [val(["5", "20"], fact([gravelly])),
                 val(["5", "15"], fact([silty])),
                 val(["5", "15"], fact([clayey]))])]).
modifier(organic_silt,
        [att(grain_size,
                [val(["0.02", "0.06"], fact([coarse])),
                 val(["0.006", "0.02"], fact([medium])),
                 val(["0.002", "0.006"], fact([fine]))]),
         att(liquid_limit,
                [val(["0", "35"], fact([low_plasticity])),
                 val(["35", "50"], fact([intermediate_plasticity])),
                 val(["50", "70"], fact([high_plasticity])),
                 val(["70", "90"], fact([very_high_plasticity])),
                 val(["90", "200"], fact([extremely_high_plasticity]))]),
         att("N_value",
                [val(["0", "4"], fact([very_loose])),
                 val(["4", "10"], fact([loose])),
                 val(["10", "30"], fact([medium_dense])),
                 val(["30", "50"], fact([dense])),
                 val(["50", "100"], fact([very_dense]))]),
         att(undrained_shear_strength,
                [val(["0", "20"], fact([very_soft])),
                 val(["20", "40"], fact([soft])),
                 val(["40", "75"], fact([firm])),
                 val(["75", "150"], fact([stiff])),
                 val(["150", "300"], fact([very_stiff]))]),
         att(coefficient_of_permeability,
                [val(["10e-7", "10e-5"], fact([low_permeability])),
                 val(["10e-9", "10e-7"], fact([very_low_permeability]))])],
```

```
        att(coefficient_of_volume_compressibility,
                [val(["0", "0.05"], fact([very_low_compressibility]))]),
        att(secondary_percent,
                [val(["35", "65"], fact([gravelly])),
                 val(["35", "65"], fact([sandy]))])]).
modifier(organic_clay,
        [att(liquid_limit,
                [val(["0", "35"], fact([low_plasticity])),
                 val(["35", "50"], fact([intermediate_plasticity])),
                 val(["50", "70"], fact([high_plasticity])),
                 val(["70", "90"], fact([very_high_plasticity])),
                 val(["90", "200"], fact([extremely_high_plasticity]))]),
        att(undrained_shear_strength,
                [val(["0", "20"], fact([very_soft])),
                 val(["20", "40"], fact([soft])),
                 val(["40", "75"], fact([firm])),
                 val(["75", "150"], fact([stiff])),
                 val(["150", "300"], fact([very_stiff]))]),
        att(coefficient_of_permeability,
                [val(["0", "10e-9"], fact([practically_impervious]))]),
        att(coefficient_of_volume_compressibility,
                [val(["1.5", "20"], fact([very_high_compressibility]))]),
        att(secondary_percent,
                [val(["35", "65"], fact([gravelly])),
                 val(["35", "65"], fact([sandy]))])]).
modifier(peat,
        [att(coefficient_of_volume_compressibility,
                [val(["1.5", "20"], fact([very_high_compressibility]))])]).
```

/ Detailed Representation of Individual In-situ Testing Methods */*

```
modifier(standard_penetration_test,
        [att(applicability,
                [val([high], fact([sand])),
                 val([medium], fact([soft_rock, gravel, silt, clay])),
                 val([low], fact([peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([soil_type, profile, angle_of_friction, undrained_shear_strength,
                        density, modulus])),
                 val([low], fact([compressibility])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(dynamic_probing_light_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([soft_rock, gravel])),
                 val([none], fact([hard_rock]))]),
```

```
        att(reliability,
                [val([high], fact([profile])),
                 val([medium], fact([])),
                 val([low], fact([angle_of_friction, undrained_shear_strength, density,
                        compressibility])),
                 val([none], fact([soil_type, piezometric_pressure, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(dynamic_probing_medium_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([gravel, sand, silt, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([low], fact([soft_rock])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([profile])),
                 val([low], fact([soil_type, angle_of_friction, undrained_shear_strength, density,
                        compressibility, modulus])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(dynamic_probing_heavy_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([gravel, sand, silt, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([low], fact([soft_rock])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([profile])),
                 val([medium], fact([angle_of_friction])),
                 val([low], fact([soil_type, undrained_shear_strength, density, compressibility,
                        modulus])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(dynamic_probing_superheavy_test,
        [att(applicability,
                [val([high], fact([sand])),
                 val([medium], fact([gravel, silt, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([low], fact([soft_rock])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([profile, density])),
                 val([low], fact([soil_type, angle_of_friction, undrained_shear_strength, modulus,
                        stress_strain_curve])),
                 val([none], fact([piezometric_pressure, compressibility, rate_of_consolidation,
                        permeability, in_situ_stress, stress_history]))])]).
```

```
modifier(mechanical_penetrometer_friction_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay])),
                 val([medium], fact([soft_rock, peat, organic_sand, organic_silt, organic_clay,
                        dense_sand, stiff_clay])),
                 val([low], fact([gravel])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([profile])),
                 val([medium], fact([soil_type, angle_of_friction, undrained_shear_strength,
                        density])),
                 val([low], fact([compressibility, modulus])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(electrical_penetrometer_friction_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([dense_sand, stiff_clay])),
                 val([low], fact([soft_rock, gravel])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([profile])),
                 val([medium], fact([soil_type, angle_of_friction, undrained_shear_strength, density,
                        compressibility, modulus])),
                 val([low], fact([])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(piezocone_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([dense_sand, stiff_clay])),
                 val([low], fact([soft_rock, gravel])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([profile])),
                 val([medium], fact([soil_type, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, density, compressibility, rate_of_consolidation,
                        modulus])),
                 val([low], fact([permeability, stress_history, stress_strain_curve])),
                 val([none], fact([in_situ_stress]))])]).
modifier(piezocone_friction_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([dense_sand, stiff_clay])),
                 val([low], fact([soft_rock, gravel])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([soil_type, profile, piezometric_pressure])),
                 val([medium], fact([angle_of_friction, undrained_shear_strength, density,
                        compressibility, rate_of_consolidation, permeability, modulus,
                        stress_history])),
                 val([low], fact([in_situ_stress, stress_strain_curve])),
                 val([none], fact([]))])]).
```

```
modifier(weight_sounding_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))])]).
modifier(static_dynamic_penetration_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([soft_rock, gravel, sand, silt, clay, peat, organic_sand,
                        organic_silt, organic_clay])),
                 val([low], fact([])),
                 val([none], fact([soft_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([profile, angle_of_friction, undrained_shear_strength,
                        compressibility, modulus])),
                 val([low], fact([soil_type, density])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(flat_plate_dilatometer_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([dense_sand, stiff_clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([undrained_shear_strength, compressibility, modulus,
                        in_situ_stress, stress_history, stress_strain_curve])),
                 val([low], fact([soil_type, profile, angle_of_friction, density])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability]))])]).
modifier(cone_pressuremeter_test,
        [att(applicability,
                [val([high], fact([silt])),
                 val([medium], fact([sand, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([soft_rock, gravel])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([profile, angle_of_friction, undrained_shear_strength,
                        rate_of_consolidation, modulus, stress_strain_curve])),
                 val([low], fact([soil_type, density, compressibility, permeability, in_situ_stress,
                        stress_history])),
                 val([none], fact([piezometric_pressure]))])]).
```

```
modifier(seismic_cone_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([soft_rock, gravel])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([profile, modulus])),
                 val([medium], fact([soil_type])),
                 val([low], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, permeability, in_situ_stress,
                        stress_history, stress_strain_curve])),
                 val([none], fact([]))])]).
modifier(lateral_stress_cone_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([soft_rock, silt, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([low], fact([hard_rock, gravel, sand])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([modulus, in_situ_stress])),
                 val([low], fact([undrained_shear_strength, compressibility, stress_history,
                        stress_strain_curve])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction, density,
                        rate_of_consolidation, permeability]))])]).
modifier(nuclear_density_probe_test,
        [att(applicability,
                [val([high], fact([sand, silt, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([density])),
                 val([medium], fact([angle_of_friction])),
                 val([low], fact([in_situ_stress, stress_strain_curve])),
                 val([none], fact([soil_type, profile, piezometric_pressure, undrained_shear_strength,
                        compressibility, rate_of_consolidation, permeability, modulus,
                        stress_history]))])]).
modifier(electrical_density_probe_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([soft_rock])),
                 val([none], fact([hard_rock, gravel]))]),
         att(reliability,
                [val([high], fact([density])),
                 val([medium], fact([soil_type, profile, angle_of_friction])),
                 val([low], fact([undrained_shear_strength, compressibility, modulus])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_history, stress_strain_curve]))])]).
```

```
modifier(electrical_conductivity_cone_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay])),
                 val([medium], fact([peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([soil_type, density])),
                 val([medium], fact([profile, compressibility, modulus])),
                 val([low], fact([angle_of_friction, undrained_shear_strength, rate_of_consolidation,
                        permeability, in_situ_stress, stress_history, stress_strain_curve])),
                 val([none], fact([piezometric_pressure]))])]).
modifier(thermal_conductivity_cone_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, density, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(acoustic_cone_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([soft_rock])),
                 val([none], fact([hard_rock, gravel]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([soil_type, profile])),
                 val([low], fact([angle_of_friction, undrained_shear_strength, density, compressibility,
                        modulus, stress_history])),
                 val([none], fact([piezometric_pressure, rate_of_consolidation, permeability,
                        in_situ_stress, stress_strain_curve]))])]).
modifier(vibratory_cone_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))])]).
```

```
modifier(menard_type_pressuremeter_test,
        [att(applicability,
                [val([high], fact([soft_rock, clay, dense_sand])),
                 val([medium], fact([hard_rock, gravel, sand, silt, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([angle_of_friction, undrained_shear_strength, modulus,
                        stress_strain_curve])),
                 val([low], fact([soil_type, profile, in_situ_stress, stress_history])),
                 val([none], fact([piezometric_pressure, density, compressibility,
                        rate_of_consolidation, permeability]))])]).
modifier(push_in_pressuremeter_test,
        [att(applicability,
                [val([high], fact([silt, clay])),
                 val([medium], fact([sand, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([undrained_shear_strength, modulus])),
                 val([low], fact([soil_type, profile, angle_of_friction, density, compressibility,
                        rate_of_consolidation, in_situ_stress, stress_history, stress_strain_curve])),
                 val([none], fact([piezometric_pressure, permeability]))])]).
modifier(self_boring_pressuremeter_test,
        [att(applicability,
                [val([high], fact([silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([sand, soft_rock])),
                 val([low], fact([hard_rock, gravel])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([modulus, in_situ_stress])),
                 val([medium], fact([piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, stress_strain_curve])),
                 val([low], fact([soil_type, profile, density, compressibility, rate_of_consolidation,
                        permeability, stress_history])),
                 val([none], fact([]))])]).
modifier(total_stress_cell_test,
        [att(applicability,
                [val([high], fact([clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([silt])),
                 val([none], fact([hard_rock, soft_rock, gravel, sand]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([in_situ_stress, stress_history])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, density, compressibility, rate_of_consolidation,
                        permeability, modulus, stress_strain_curve]))])]).
```

```
modifier(iowa_stepped_blade_test,
        [att(applicability,
                [val([high], fact([silt, clay])),
                 val([medium], fact([sand, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([in_situ_stress, stress_history])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, density, compressibility, rate_of_consolidation,
                        permeability, modulus, stress_strain_curve]))])]).
modifier(hydraulic_fracturing_test,
        [att(applicability,
                [val([high], fact([clay])),
                 val([medium], fact([hard_rock, soft_rock, silt])),
                 val([low], fact([gravel, sand, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([piezometric_pressure])),
                 val([medium], fact([in_situ_stress, stress_history])),
                 val([low], fact([rate_of_consolidation, permeability])),
                 val([none], fact([soil_type, profile, angle_of_friction, undrained_shear_strength,
                        density, compressibility, modulus, stress_strain_curve]))])]).
modifier(self_boring_ko_meter_test,
        [att(applicability,
                [val([high], fact([silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([sand])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([in_situ_stress, stress_history])),
                 val([medium], fact([soil_type, profile])),
                 val([low], fact([])),
                 val([none], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, permeability, modulus,
                        stress_strain_curve]))])]).
modifier(vane_test,
        [att(applicability,
                [val([high], fact([clay])),
                 val([medium], fact([silt, stiff_clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([soft_rock])),
                 val([none], fact([hard_rock, gravel, sand]))]),
         att(reliability,
                [val([high], fact([undrained_shear_strength])),
                 val([medium], fact([])),
                 val([low], fact([profile])),
                 val([none], fact([soil_type, piezometric_pressure, angle_of_friction, density,
                        compressibility, rate_of_consolidation, permeability, modulus, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
```

```
modifier(self_boring_vane_test,
        [att(applicability,
                [val([high], fact([clay])),
                 val([medium], fact([])),
                 val([low], fact([sand, silt, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([undrained_shear_strength])),
                 val([medium], fact([])),
                 val([low], fact([soil_type, profile, stress_history])),
                 val([none], fact([piezometric_pressure, angle_of_friction, density, compressibility,
                        rate_of_consolidation, permeability, modulus, in_situ_stress,
                        stress_strain_curve]))])]).
modifier(borehole_shear_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([hard_rock, soft_rock, sand, silt])),
                 val([low], fact([gravel, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([angle_of_friction])),
                 val([low], fact([soil_type, profile, undrained_shear_strength, modulus,
                        stress_history])),
                 val([none], fact([piezometric_pressure, density, compressibility,
                        rate_of_consolidation, permeability, in_situ_stress, stress_strain_curve]))])]).
modifier(in_situ_shear_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([soft_rock, silt, clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, gravel, sand, peat, organic_sand, organic_silt,
                        organic_clay]))]),
         att(reliability,
                [val([high], fact([angle_of_friction, undrained_shear_strength])),
                 val([medium], fact([])),
                 val([low], fact([rate_of_consolidation, modulus, stress_strain_curve])),
                 val([none], fact([soil_type, profile, piezometric_pressure, density, compressibility,
                        permeability, in_situ_stress, stress_history]))])]).
modifier(plate_loading_tests,
        [att(applicability,
                [val([high], fact([soft_rock, gravel, sand, silt, clay])),
                 val([medium], fact([hard_rock, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([undrained_shear_strength, compressibility, modulus,
                        stress_strain_curve])),
                 val([low], fact([rate_of_consolidation])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction, density,
                        permeability, in_situ_stress, stress_history]))])]).
```

```
modifier(screw_plate_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([modulus])),
                 val([medium], fact([undrained_shear_strength, density, compressibility,
                        stress_history, stress_strain_curve])),
                 val([low], fact([soil_type, profile, angle_of_friction, rate_of_consolidation,
                        permeability, in_situ_stress])),
                 val([none], fact([piezometric_pressure]))])]).
modifier(self_boring_plate_test,
        [att(applicability,
                [val([high], fact([silt, clay])),
                 val([medium], fact([sand, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock, gravel]))]),
         att(reliability,
                [val([high], fact([modulus, stress_history])),
                 val([medium], fact([soil_type, profile, undrained_shear_strength, density,
                        compressibility, in_situ_stress])),
                 val([low], fact([angle_of_friction, rate_of_consolidation, permeability,
                        stress_strain_curve])),
                 val([none], fact([piezometric_pressure]))])]).
modifier(pressurized_chamber_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))])]).
modifier(in_situ_california_bearing_ratio_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([gravel, sand, silt, clay])),
                 val([low], fact([soft_rock, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([undrained_shear_strength, modulus])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction, density,
                        compressibility, rate_of_consolidation, permeability, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
```

```
modifier(small_pouring_cylinder_test,
        [att(applicability,
                [val([high], fact([sand, silt, clay])),
                 val([medium], fact([soft_rock, gravel])),
                 val([low], fact([peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([density])),
                 val([medium], fact([])),
                 val([low], fact([soil_type])),
                 val([none], fact([profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(large_pouring_cylinder_test,
        [att(applicability,
                [val([high], fact([sand, silt])),
                 val([medium], fact([soft_rock, gravel, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([density])),
                 val([medium], fact([])),
                 val([low], fact([soil_type])),
                 val([none], fact([profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(scoop_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([sand, silt, clay])),
                 val([low], fact([soft_rock, gravel, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([hard_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([density])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(core_cutter_test,
        [att(applicability,
                [val([high], fact([clay])),
                 val([medium], fact([peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([soft_rock, silt])),
                 val([none], fact([hard_rock, gravel, sand]))]),
```

```
        att(reliability,
                [val([high], fact([density])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(weight_in_water_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([hard_rock])),
                 val([low], fact([soft_rock])),
                 val([none], fact([gravel, sand, silt, clay, peat, organic_sand, organic_silt,
                        organic_clay]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([density])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(water_replacement_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([hard_rock, soft_rock])),
                 val([low], fact([clay])),
                 val([none], fact([gravel, sand, silt, peat, organic_sand, organic_silt, organic_clay]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([density])),
                 val([low], fact([soil_type])),
                 val([none], fact([profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(rubber_balloon_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([gravel, sand, silt, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([low], fact([])),
                 val([none], fact([hard_rock, soft_rock]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([soil_type, density])),
                 val([low], fact([])),
                 val([none], fact([profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
```

```
modifier(backscatter_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([soft_rock, gravel, sand, silt, clay])),
                 val([low], fact([hard_rock, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([density])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(direct_transmission_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([soft_rock, gravel, sand, silt, clay])),
                 val([low], fact([hard_rock, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([density])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, compressibility, rate_of_consolidation,
                        permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(air_gap_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([])),
                 val([low], fact([hard_rock, soft_rock, gravel, sand, silt, clay, peat, organic_sand,
                        organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([density])),
                 val([low], fact([soil_type])),
                 val([none], fact([]))])]).
modifier(rising_head_test,
        [att(applicability,
                [val([high], fact([gravel, sand, silt])),
                 val([medium], fact([])),
                 val([low], fact([hard_rock, soft_rock, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([permeability])),
                 val([low], fact([piezometric_pressure])),
                 val([none], fact([soil_type, profile, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, modulus, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
```

```
modifier(falling_head_test,
        [att(applicability,
                [val([high], fact([gravel, sand])),
                 val([medium], fact([silt])),
                 val([low], fact([hard_rock, soft_rock, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([permeability])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, piezometric_pressure, angle_of_friction,
                        undrained_shear_strength, density, compressibility, rate_of_consolidation,
                        modulus, in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(constant_head_test,
        [att(applicability,
                [val([high], fact([])),
                 val([medium], fact([gravel, sand, silt])),
                 val([low], fact([hard_rock, soft_rock, clay, peat, organic_sand, organic_silt,
                        organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([permeability])),
                 val([low], fact([piezometric_pressure])),
                 val([none], fact([soil_type, profile, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, modulus, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
modifier(self_boring_permeameter_test,
        [att(applicability,
                [val([high], fact([silt])),
                 val([medium], fact([sand, clay])),
                 val([low], fact([soft_rock, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([hard_rock, gravel]))]),
         att(reliability,
                [val([high], fact([permeability])),
                 val([medium], fact([])),
                 val([low], fact([piezometric_pressure])),
                 val([none], fact([soil_type, profile, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, modulus, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
modifier(pumping_tests,
        [att(applicability,
                [val([high], fact([gravel, sand])),
                 val([medium], fact([hard_rock, soft_rock, silt])),
                 val([low], fact([clay, peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([piezometric_pressure, permeability])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([soil_type, profile, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, modulus, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
```

```
modifier(seismic_refraction_test,
        [att(applicability,
                [val([high], fact([hard_rock, soft_rock, gravel, sand, silt, clay])),
                 val([medium], fact([])),
                 val([low], fact([peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([profile])),
                 val([medium], fact([density])),
                 val([low], fact([soil_type, compressibility, permeability, modulus])),
                 val([none], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        rate_of_consolidation, in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(seismic_reflection_test,
        [att(applicability,
                [val([high], fact([hard_rock, soft_rock, gravel])),
                 val([medium], fact([sand, silt, clay])),
                 val([low], fact([peat, organic_sand, organic_silt, organic_clay])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([profile])),
                 val([medium], fact([density])),
                 val([low], fact([soil_type, compressibility, permeability, modulus])),
                 val([none], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        rate_of_consolidation, in_situ_stress, stress_history, stress_strain_curve]))])]).
modifier(seismic_cross_hole_test,
        [att(applicability,
                [val([high], fact([hard_rock, soft_rock, gravel, sand, silt, clay, peat, organic_sand,
                        organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([modulus])),
                 val([medium], fact([])),
                 val([low], fact([soil_type, profile])),
                 val([none], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, permeability, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
modifier(seismic_down_hole_test,
        [att(applicability,
                [val([high], fact([hard_rock, soft_rock, gravel, sand, silt, clay, peat, organic_sand,
                        organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([modulus])),
                 val([medium], fact([])),
                 val([low], fact([soil_type, profile])),
                 val([none], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, permeability, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
```

```
modifier(surface_wave_test,
        [att(applicability,
                [val([high], fact([hard_rock, soft_rock, gravel, sand, silt, clay, peat, organic_sand,
                        organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([modulus])),
                 val([medium], fact([])),
                 val([low], fact([soil_type, profile])),
                 val([none], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        density, compressibility, rate_of_consolidation, permeability, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
modifier(resistivity_test,
        [att(applicability,
                [val([high], fact([gravel, sand, silt, clay])),
                 val([medium], fact([soft_rock, peat, organic_sand, organic_silt, organic_clay])),
                 val([low], fact([hard_rock])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([profile])),
                 val([low], fact([soil_type, piezometric_pressure, density])),
                 val([none], fact([angle_of_friction, undrained_shear_strength, compressibility,
                        rate_of_consolidation, permeability, modulus, in_situ_stress, stress_history,
                        stress_strain_curve]))])]).
modifier(gravimetric_test,
        [att(applicability,
                [val([high], fact([hard_rock, soft_rock, gravel, sand, silt, clay, peat, organic_sand,
                        organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
         att(reliability,
                [val([high], fact([])),
                 val([medium], fact([profile, density])),
                 val([low], fact([soil_type])),
                 val([none], fact([piezometric_pressure, angle_of_friction, undrained_shear_strength,
                        compressibility, rate_of_consolidation, permeability, modulus, in_situ_stress,
                        stress_history, stress_strain_curve]))])]).
modifier(magnetic_test,
        [att(applicability,
                [val([high], fact([hard_rock, soft_rock, gravel, sand, silt, clay, peat, organic_sand,
                        organic_silt, organic_clay])),
                 val([medium], fact([])),
                 val([low], fact([])),
                 val([none], fact([]))]),
```

```
att(reliability,
        [val([high], fact([])),
         val([medium], fact([profile])),
         val([low], fact([soil_type, undrained_shear_strength, density])),
         val([none], fact([piezometric_pressure, angle_of_friction, compressibility,
                rate_of_consolidation, permeability, modulus, in_situ_stress, stress_history,
                stress_strain_curve]))])]).
```

```
/*************************
*    File GENERIC.PRO    *
*************************/
```

/ This file contains the generic rules, concerning list processing, used by the rules of the main program. */*

```
domains


vallists=vallist*
reallist=real*
reallists=reallist*
lists=list*


predicates


append(attlist, attlist, attlist)
append(list, list, list)
append(list, stringlist, stringlist)
append(lists, lists, lists)
append(reallist, reallist, reallist)
append(stringlist, stringlist, stringlist)
append(vallist, vallist, vallist)
append(vallists, vallists, vallists)
delete_item(integer, integerlist, integerlist)
delete_item(symbol, list, list)
delete_item(symbol, stringlist, stringlist)
delete_item(vallist, vallists, vallists)
delete_list(list, list, list)
delete_list(list, stringlist, stringlist)
delete_list(vallists, vallists, vallists)
first(list, symbol)
first(stringlist, symbol)
last(list, symbol)
last(stringlist, string)
max_number(reallist, real)
member(att, attlist)
member(real, reallist)
member(string, stringlist)
member(symbol, list)
member(vallist, vallists)
members(att, attlist)
members(symbol, list)
min_number(reallist, real)
remove_duplicates(list, list, list)
```

```
remove_duplicates(list, stringlist, stringlist)
remove_duplicates(reallist, reallist, reallist)
remove_duplicates(stringlist, stringlist, stringlist)
remove_duplicates(vallists, vallists, vallists)
reverse(attlist, attlist)
reverse(list, list)
reverse(list, stringlist)
reverse(reallist, reallist)
reverse(stringlist, stringlist)
reverse(vallist, vallist)
simplify_lists(lists, list, list)
simplify_lists(reallists, reallist, reallist)
simplify_lists(vallists, vallist, vallist)
split_list(symbol, stringlist, stringlist, stringlist)
split_list(symbol, list, list, list)



clauses


members(Name, [Name|_]).
members(Name, [_|Tail]) :-
        members(Name, Tail).


member(Name, [Name|_]):-!.
member(Name, [_|Tail]) :-
        member(Name, Tail).


reverse([], []).
reverse([Head|Tail], List):-
        reverse(Tail, Result),
        append(Result, [Head], List).


append([], List, List).
append([X|L1], List2, [X|L3]):-
        append(L1, List2, L3).


remove_duplicates([], List2, List2):-!.
remove_duplicates(List, List1, List2):-
        List=[H|Tail],
        member(H, Tail), !,
        remove_duplicates(Tail, List1, List2).
remove_duplicates(List, List1, List2):-
        List=[H|Tail],
        not(member(H, Tail)),
        append([H], List1, TempList),
        remove_duplicates(Tail, Templist, List2).
```

```
split_list(Name, List, L_front, L_back):-
        append(L_front, L_back, List),
        first(L_back, Name).


first([First|_], First).


last([Last], Last).
last([X|Rest], Last):-
        last(Rest, Last).


simplify_lists([], List, List).
simplify_lists(Lists, Old_list, List):-
        Lists=[Head|Tail],
        append(Head, Old_list, Temp_list),
        simplify_lists(Tail, Temp_list, List).


max_number([X], X).
max_number([X|Tail], X):-
        max_number(Tail, M),
        X>M.
max_number([X|Tail], M):-
        max_number(Tail, M), X<=M.


min_number([X], X).
min_number([X|Tail], X):-
        min_number(Tail, M),
        X<M.
min_number([X|Tail], M):-
        min_number(Tail, M), X>=M.


delete_item(Item, [], []).
delete_item(Item, [Item|Tail], List2):-!,
        delete_item(Item, Tail, List2).
delete_item(Item, [Head|Tail], [Head|Rest]):-
        not(Item=Head),
        delete_item(Item, Tail, Rest).


delete_list([], List, List).
delete_list([H|T], List1, List):-
        delete_item(H, List1, Temp_list),
        delete_list(T, Temp_list, List).
```

```
/**********************
*    File INFINT.PRO    *
**********************/
```

/ This file contains the Extended Inference Mechanism, the advisory rule developed to assist in the selection of appropriate in-situ tests and the rules required for the development of the user interface. */*

```
code=4100


include "\\pdcpro\\toolbox\\ui\\longmenu.pro"
include "\\pdcpro\\toolbox\\ui\\status.pro"
include "\\pdcpro\\toolbox\\ui\\menu.pro"
include "\\pdcpro\\toolbox\\ui\\lineinp.pro"
include "\\pdcproph\\knowbase.pro"
include "\\pdcproph\\generic.pro"


domains

name=symbol
names=name*
vallistss=vallists*


predicates

case(symbol, symbol)
change_value(att, attlist, attlist, attlist)
check_attributes_left(list)
check_integer(list, integer, integer)
check_option(integer)
check_parameter(symbol, attlist, symbol)
check_selection(integer, string, stringlist)
check_soils_left(list)
check_val_list(vallist)
condition(symbol, symbol, list)
continue(char, symbol, vallist)
convert_input(string, list)
discover_member(symbol, symbol)
discover_members(symbol, list)
find_all_ancestors(symbol, list, lists)
find_all_attrib_names(symbol, symbol, stringlist)
find_all_general_range(symbol, reallist, reallist, reallist)
find_all_mod_attributes(string, stringlist, list)
find_all_mod_attributes(symbol, list, list)
```

find_all_mod_attributes(symbol, stringlist, stringlist)
find_all_names_factors(symbol, list, list, lists)
find_all_num_value_attr(symbol, symbol, reallist)
find_all_roots(list)
find_all_roots(stringlist)
find_all_sym_values(string, stringlist)
find_all_test_attributes(list)
find_all_test_attributes(stringlist)
find_ancestors(symbol, list, list)
find_attrib_name(symbol, symbol, list, symbol)
find_attribute_and_value(symbol, symbol, attlist, attlist)
find_attribute_data(symbol, list, list)
find_factors(attlist, list, list)
find_factors(attlist, stringlist, stringlist)
find_instances(list, list, list, list)
find_modifiers(symbol, symbol, list, lists)
find_num_values(vallist, reallist, reallist)
find_objects_and_modifiers(symbol, list, symbol, list)
find_root(symbol)
find_root_tree(string, stringlist, stringlist, stringlist)
find_root_tree(string, stringlist, list, list)
find_root_tree(symbol, list, list, list)
find_sym_values(vallist, list, list)
find_sym_values(vallist, stringlist, stringlist)
find_test_attributes(list, list)
find_test_attrs(symbol, list, list)
find_unique_attribute_data(symbol, list, list)
find_unique_attribute_data(symbol, stringlist, stringlist)
find_vallist(symbol, symbol, symbol, vallist, vallist)
find_vallists(symbol, symbol, symbol)
get_add_value(symbol, list, list, list, vallists, vallists)
get_all_attributes(symbol, symbol, attlist, attlist, attlist)
get_all_fact_list(string, string, stringlist)
get_all_fact_list(symbol, symbol, list)
get_all_names_with_factors(stringlist, symbol, stringlist)
get_attlist(symbol, attlist)
get_attrib_value(symbol, attlist, attlist, attlist)
get_attribute_data(vallist, list, list, list, list)
get_attribute_names(attlist, list, list)
get_attribute_names(attlist, stringlist, stringlist)
get_fact(vallist, list, list)
get_fact_attribute_list(symbol, list)
get_fact_attribute_list(symbol, stringlist)
get_fact_list(symbol, symbol, list, symbol)
get_factor(vallist, list, list)
get_factors(attlist, symbol, list, lists)
get_general_range(symbol, reallist, reallist)
get_members(symbol, list)
get_mod_attributes(string, stringlist, list)
get_mod_attributes(symbol, list, list)
get_mod_f(vallist, list, list)
get_modified_soil(symbol, list, list, list)
get_modified_value(symbol, lists, symbol, list, list, list, list)
get_name_factor(attlist, symbol, list, symbol, list)

get_names_values(vallist, list, list, list, list, list)
get_names_with_factors(symbol, symbol, stringlist)
get_order(list, list)
get_parents(symbol)
get_root_tree(string, stringlist, list, list)
get_root_tree(string, stringlist, stringlist, stringlist)
get_root_tree(symbol, list, list, list)
get_soil_value(vallist, symbol, list, list, list, list)
get_sym_values(symbol, list)
get_val_list(attlist, symbol, vallist)
give_value(symbol, list, symbol, list, list)
investigate(symbol, symbol, symbol, list, list, symbol, list, list, list, list, list, vallists)
match_choice(stringlist, integer, symbol)
match_choices(stringlist, integerlist, list, list)
modified_soil(symbol, list)
modified_soil_names(list, lists, lists)
num_matches(list, list)
num_value_attr(symbol, symbol, reallist, reallist)
output_modifiers(lists)
output_whole_range_modifiers(lists)
set_attribute(att, attlist, attlist)
set_attributes(attlist, attlist, attlist)
situation(symbol)
sort_test_name(symbol, symbol, symbol)
state(symbol, list)
sym_matches(list, list)
sym_value_attr(symbol, symbol, list)
sym_value_attr(symbol, symbol, stringlist)
user_interface
write_add_attr(list, vallists)
write_add_attributes(list, list, vallists)
write_app(list, list)
write_applicability(list, list, list)
write_attlist(attlist)
write_children(list, integer)
write_fact_list(list)
write_factor(list)
write_factor_list(list)
write_list(list)
write_lists(lists)
write_mod_app(list, list)
write_mod_applicability(list, list)
write_names_factors(list, lists)
write_non_app(list)
write_non_attr(list)
write_soil_names(list, symbol)
write_title(vallist)
write_v_list(vallist)
write_val_list(vallist)
write_vallist(vallist)
write_vallists(vallists)
write_values(val)

```
clauses

/* The clauses below describe the rules that form the Extended Inference Mechanism. */


get_all_attributes(Root, Root, Old_attlist, Class_attlist, Mod_attlist):-
        class(Root, _, Att_list),
        set_attributes(Att_list, Old_attlist, New_attlist),
        reverse(New_attlist, Class_attlist),
        modifier(Root, Mod_attlist), !.
get_all_attributes(Root, Root, Old_attlist, Class_attlist, []):-
        class(Root, _, Att_list),
        set_attributes(Att_list, Old_attlist, New_attlist),
        reverse(New_attlist, Class_attlist).
get_all_attributes(Root, Root, Att_list, Att_list, []):-
        not(class(Root, _, _)).
get_all_attributes(Name, Root, Old_attlist, Attlist, Mod_attlist):-
        class(Root, List, Att_list),
        set_attributes(Att_list, Old_attlist, Temp_attlist),
        members(Member, List),
        get_all_attributes(Name, Member, Temp_attlist, Attlist, Mod_attlist).

set_attributes([], Att_list, Att_list).
set_attributes(Att_list, Old_attlist, New_attlist):-
        Att_list=[att(Attribute, Val_list)|Tail],
        set_attribute(att(Attribute, Val_list), Old_attlist, Temp_attlist),
        set_attributes(Tail, Temp_attlist, New_attlist).

set_attribute(att(Attribute, Val_list), Old_attlist, New_attlist):-
        not(member(att(Attribute, _), Old_attlist)),
        append([att(Attribute, Val_list)], Old_attlist, New_attlist).
set_attribute(att(Attribute, Val_list), Old_attlist, New_attlist):-
        member(att(Attribute, _), Old_attlist),
        change_value (att(Attribute, Val_list), Old_attlist, [], New_attlist).

change_value(_, [], Att_list, Att_list).
change_value(att(Attribute, Val_list), Old_attlist, Vals, New_attlist):-
        Old_attlist=[att(Attrib_1, _ ) |Tail],
        Attribute=Attrib_1,
        Attlist=[att(Attrib_1, Val_list)|Vals],
        change_value(att(Attribute, Val_list), Tail, Attlist, New_attlist).
change_value(att(Attribute, Val_list), Old_attlist, Vals, New_attlist):-
        Old_attlist=[att(Attrib_1, Val_list_1) |Tail],
        not(Attribute=Attrib_1),
        Attlist=[att(Attrib_1, Val_list_1) |Vals],
        change_value(att(Attribute, Val_list), Tail, Attlist, New_attlist).
```

```
find_vallists(Name, Root, Attribute):-
        write(" Processing knowledge..."), nl,
        findall(Class_vallist, find_vallist(Name, Root, Attribute, Class_vallist, Mod_vallist),
                Class_val_lists),
        simplify_lists(Class_val_lists, [], Class_vallists),
        check_val_list(Class_vallists),
        Mod_vallist=[], !,
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write("  There is no available knowledge for the attribute ", Attribute, " for the ", Name), nl.
find_vallists(Name, Root, Attribute):-
        findall(Class_vallist, find_vallist(Name, Root, Attribute, Class_vallist, Mod_vallist),
                Class_val_lists),
        Mod_vallist=[],
        simplify_lists(Class_val_lists, [], Class_val_list),
        not(check_val_list(Class_val_list)), !,
        remove_duplicates(Class_val_lists, [], Class_vallists),
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write("  The attribute ", Attribute, " has "),
        write_vallists(Class_vallists).
find_vallists(Name, Root, Attribute):-
        findall(Class_vallist, find_vallist(Name, Root, Attribute, Class_vallist, Mod_vallist),
                Class_val_lists),
        simplify_lists(Class_val_lists, [], Class_vallists),
        not(Mod_vallist=[]),
        check_val_list(Class_vallists), !,
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write("  The attribute ", Attribute, " has "),
        write_v_list(Mod_vallist).
find_vallists(Name, Root, Attribute):-
        findall(Class_vallist, find_vallist(Name, Root, Attribute, Class_vallist, Mod_vallist),
                Class_val_lists),
        simplify_lists(Class_val_lists, [], Class_val_list),
        not(check_val_list(Class_val_list)),
        not(Mod_vallist=[]),
        remove_duplicates(Class_val_lists, [], Class_value_list),
        delete_list([[]], Class_value_list, Class_vallists),
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write("  The attribute ", Attribute, " has "),
        write_vallists(Class_vallists),
        write(" Would you like to see a more detailed representation? (y/n)"), nl,
        readchar(X), nl,
        continue(X, Attribute, Mod_vallist).

find_vallist(Name, Root, Attribute, Class_vallist, Mod_vallist):-
        get_all_attributes(Name, Root, [], Class_attlist, Mod_attlist),
        get_val_list(Class_attlist, Attribute, Class_vallist),
        get_val_list(Mod_attlist, Attribute, Mod_vallist),
        not(Class_vallist=[]),
        not(Mod_vallist=[]).
```

```
find_vallist(Name, Root, Attribute, Class_vallist, Mod_vallist):-
        get_all_attributes(Name, Root, [], Class_attlist, Mod_attlist),
        get_val_list(Class_attlist, Attribute, Class_vallist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist),
        not(member(Attribute, Mod_attrlist)),
        Mod_vallist=[].
find_vallist(Name, Root, Attribute, Class_vallist, Mod_vallist):-
        get_all_attributes(Name, Root, [], Class_attlist, Mod_attlist),
        get_attribute_names(Class_attlist, [], Class_attrlist),
        get_val_list(Mod_attlist, Attribute, Mod_vallist),
        not(member(Attribute, Class_attrlist)),
        Class_vallist=[].

check_val_list(Class_val_list):-
        Class_val_list=[].
check_val_list(Class_val_list):-
        Class_val_list=[val([], fact([]))].

continue(X, Attribute, Mod_vallist):-
        X='y', !,
        write("  In a more detailed representation scheme, the attribute ", Attribute, " has "), nl,
        write_v_list(Mod_vallist).
continue('n', _, _).

get_val_list([att(Attribute, Val_list)|_], Attribute, Val_list):-!.
get_val_list([_|Tail], Attribute, Val_list):-
        get_val_list(Tail, Attribute, Val_list).

get_attribute_names([], Attrlist, Attrlist).
get_attribute_names([att(Attribute, _)|Tail], Oldlist, Attrlist):-
        append([Attribute], Oldlist, Templist),
        get_attribute_names(Tail, Templist, Attrlist).

write_vallists([Head|Tail]):-
        Tail=[],
        write_v_list(Head).
write_vallists([Head|Tail]):-
        not(Tail=[]),
        not(Tail=[[]]),
        write_v_list(Head),
        write(" Alternatively, it could have "),
        write_vallists(Tail).

write_v_list([H|[]]):-
        write_values(H).
write_v_list(Val_list):-
        Val_list=[_|T],
        not(T=[]),
        write_title(Val_list),
        write_vallist(Val_list).
```

```
write_values(val([Vmin, Vmax], fact(F))):-
        not(F=[]), !,
        write("the following range of values:"), nl,
        write('\t', "vmin= ", Vmin), nl,
        write('\t', "vmax= ", Vmax), nl,
        write("and the modifier is: "), nl,
        write_list(F).
write_values(val([V], fact(F))):-
        not(F=[]), !,
        write("the following value:"), nl,
        write('\t', "value= ", V), nl,
        write("and the factor is: "), nl,
        write_list(F).
write_values(val([Vmin, Vmax], fact([]))):-!,
        write("the following range of values:"), nl,
        write('\t', "vmin= ", Vmin), nl,
        write('\t', "vmax= ", Vmax), nl, nl.
write_values(val([V], fact([]))):-
        write("the following value:"), nl,
        write('\t', "value= ", V), nl, nl.

write_title([val([_, _], fact(_))|_]):-
        write("the following ranges of values according to the modifier:"), nl,
        writef("\t%10s %10s %30s", vmin, vmax, modifier), nl.
write_title([val([_], fact(_))|_]):-
        write("the following values according to the modifier:"), nl,
        writef("\t%20s %30s", value, modifier), nl.

write_vallist([]).
write_vallist([val([Vmin, Vmax], fact(F))|Rest]):-
        not(F=[]),
        F=[H|T],
        writef("\t%10 %10 %30", Vmin, Vmax, H), nl,
        write_factor(T),
        write_vallist(Rest).
write_vallist([val([Vmin, Vmax], fact([]))|Rest]):-
        writef("\t%10 %10 %30", Vmin, Vmax, "No modifiers specified"), nl,
        write_vallist(Rest).
write_vallist([val([Value], fact(F))|Rest]):-
        not(F=[]),
        F=[H|T],
        writef("\t%20 %30 ", Value, H), nl,
        write_factor(T),
        write_vallist(Rest).
write_vallist([val([Value], fact([]))|Rest]):-
        writef("\t%20 %30", Value, "No modifiers specified"), nl,
        write_vallist(Rest).

write_factor([]).
write_factor([A|Tail]):-
        writef("\t%50", A), nl,
        write_factor(Tail).
```

```
find_all_ancestors(Name, Old_list, Ancestor_lists):-
        write(" Processing knowledge..."), nl,
        findall(Ancestor_list, find_ancestors(Name, Old_list, Ancestor_list), Ancestor_lists),
        not(Ancestor_lists=[[]]), !,
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write("   ", Name, " has the following ancestor(s): "), nl, nl,
        write_lists(Ancestor_lists), nl.
find_all_ancestors(Name, Old_list, Ancestor_lists):-
        findall(Ancestor_list, find_ancestors(Name, Old_list, Ancestor_list), Ancestor_lists),
        Ancestor_lists=[[]],
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write("   ", Name, " is the root of the hierarchy.").

find_ancestors(Name, Ancestor_list, Ancestor_list):-
        findall(List, class(_, List, _), Lists),
        simplify_lists(Lists, [], Simp_list),
        not(member(Name, Simp_list)).
find_ancestors(Name, Old_list, Ancestor_list):-
        class(Parent, List, _),
        member(Name, List),
        append([Parent], Old_list, Temp_list),
        find_ancestors(Parent, Temp_list, Ancestor_list).

write_lists(Ancestor_lists):-
        Ancestor_lists=[Ancestor|Tail],
        Tail=[], !,
        write_list(Ancestor), nl.
write_lists(Ancestor_lists):-
        Ancestor_lists=[Ancestor|Tail],
        write_list(Ancestor), nl,
        not(Tail=[]),
        write("   An alternative solution is: "), nl, nl,
        write_lists(Tail), nl.

write_list([]).
write_list([H|Rest]):-
        write("              ", H), nl,
        write_list(Rest).



get_members(Category, Names):-
        discover_members(Category, Names),
        not(Names=[_|[]]), !,
        write(" The members of the category ", Category, " are: "), nl, nl,
        write_children(Names, 1).
get_members(Category, Names):-
        discover_members(Category, Names),
        write(" The category ", Category, " has one member which is:"), nl, nl,
        write_children(Names, 1).
```

```
discover_members(Category, Names):-
        findall(Name, discover_member(Category, Name), Nameslist),
        remove_duplicates(Nameslist, [], Namelist),
        reverse(Namelist, Names).

discover_member(Name, Name):-
        class(Name, [], _), !.
discover_member(Category, Name):-
        class(Category, List, _), !,
        members(Member, List),
        discover_member(Member, Name).

write_children([], _):-!.
write_children(Soil_names, N):-
        Soil_names=[H|T],
        writef(" %30" , H), nl,
        TempN=N+1,
        check_integer(T, TempN, NewN),
        write_children(T, NewN).

check_integer(T, TempN, NewN):-
        TempN>=15, nl,
        not(T=[]), !,
        write(" Press any key to see the rest..."), nl, nl,
        readchar(_),
        NewN=1.
check_integer(T, TempN, NewN):-
        TempN>=15, nl,
        T=[], !,
        NewN=TempN.
check_integer(T, TempN, NewN):-
        TempN<15,
        NewN=TempN.



find_attribute_and_value(Name, Factor, Old_attlist, All_attlist):-
        modifier(Name, Attlist),
        get_attrib_value(Factor, Attlist, Old_attlist, New_attlist),
        reverse(New_attlist, All_attlist),
        write("   For the modifier ", Factor),
        write_attlist(All_attlist).

get_attrib_value(_, [], All_list, All_list).
get_attrib_value(Factor, List, Old_list, All_list):-
        List=[att(Attribute, Val_list)|Tail],
        Val_list=[val(_, fact(Factors))|Rest],
        not(Rest=[]),
        not(member(Factor, Factors)),
        get_attrib_value(Factor, [att(Attribute, Rest)|Tail], Old_list, All_list).
```

```
get_attrib_value(Factor, List, Old_list, All_list):-
        List=[att(Attribute, Val_list)|Tail],
        Val_list=[val(_, fact(Factors))|Rest],
        Rest=[],
        not(member(Factor, Factors)),
        get_attrib_value(Factor, Tail, Old_list, All_list).
get_attrib_value(Factor, List, Old_list, All_list):-
        List=[att(Attribute, Val_list)|Tail],
        Val_list=[val(Value, fact(Factors))|_],
        member(Factor, Factors),
        append([att(Attribute, [val(Value, fact(Factors))])], Old_list, New_list),
        get_attrib_value(Factor, Tail, New_list, All_list).

write_attlist([]).
write_attlist([att(Attribute, Val_list)|Tail]):-
        write(" the attribute ", Attribute), nl,
        write_val_list(Val_list),
        write_attlist(Tail).
write_attlist([att(Attribute, Val_list)|Tail]):-
        Val_list=[val([], fact(_))],
        write('\t', "has no values"), nl,
        write_attlist(Tail).

write_val_list([]).
write_val_list([val([Vmin, Vmax], fact(_))|Rest]):-
        write("    takes the following range of values:"), nl,
        write('\t', '\t', "Vmin= ", Vmin), nl,
        write('\t', '\t', "Vmax= ", Vmax), nl,
        write_val_list(Rest).
write_val_list([val([Value], fact(_))|Rest]):-
        write("    takes the following value:"), nl,
        write('\t', '\t', "Value= ", Value), nl,
        write_val_list(Rest).



find_modifiers(Name, Attribute, Value_list, Factors):-
        modifier(Name, Attlist),
        get_factors(Attlist, Attribute, Value_list, Factors).

get_factors(Attlist, Attribute, Value_list, Factors):-
        get_val_list(Attlist, Attribute, Val_list),
        not(Val_list=[_|[]]),
        findall(Factor, get_factor(Val_list, Value_list, Factor), Factors),
        output_modifiers(Factors).
get_factors(Attlist, Attribute, Value_list, Factors):-
        get_val_list(Attlist, Attribute, Val_list),
        Val_list=[_|[]],
        findall(Factor, get_factor(Val_list, Value_list, Factor), Factors),
        output_whole_range_modifiers(Factors).
```

```
get_factors(Attlist, Attribute, Value_list, Modified_factors):-
        get_val_list(Attlist, Attribute, Val_list),
        findall(Factor, get_factor(Val_list, Value_list, Factor), Factor_list),
        Factor_list=[],
        findall(Modified_factor, get_mod_f(Val_list, Value_list, Modified_factor), Modified_factors),
        not(Modified_factors=[]),
        write(" Corresponding modifier(s): "), nl,
        write_lists(Modified_factors), nl.

get_factor([val(Values, fact(Factor))|_], Value_list, Factor):-
        num_matches(Values, Value_list).
get_factor([val(Values, fact(Fact))|_], Value_list, Factor):-
        not(num_matches(Values, Value_list)),
        sym_matches(Values, Value_list),
        Fact=Factor.
get_factor([_|Rest], Value_list, Factor):-
        get_factor(Rest, Value_list, Factor).

get_mod_f(Val_list, [V1, V2], Factor):-
        get_order([V1, V2], [Lo, Hi]),
        get_fact(Val_list, [Lo], [Factor1]),
        get_fact(Val_list, [Hi], [Factor2]),
        not(Factor1=Factor2),
        concat(Factor1, "_to_", Temp_factor),
        concat(Temp_factor, Factor2, Fact),
        Factor=[Fact].

get_order([V1, V2], [Lo, Hi]):-
        str_real(V1, V1r),
        str_real(V2, V2r),
        V1r<V2r, !,
        V1=Lo,
        V2=Hi.
get_order([V1, V2], [Lo, Hi]):-
        V1=Hi,
        V2=Lo.

num_matches([Min, Max], [V1, V2]):-
        get_order([V1, V2], [Lo, Hi]),
        str_real(Min, Vmin),
        str_real(Max, Vmax),
        str_real(Lo, Vlo),
        str_real(Hi, Vhi),
        Vlo>=Vmin,
        Vhi<=Vmax, !.
num_matches([Min, Max], [V]):-
        str_real(Min, Vmin),
        str_real(Max, Vmax),
        str_real(V, Vreal),
        Vreal>=Vmin,
        Vreal<=Vmax.
```

```
sym_matches(_, []):-!.
sym_matches(Values, [H|T]):-
        member(H, Values),
        sym_matches(Values, T).

get_fact([val([Min, Max], fact(Factor1))|_], [E], Factor1):-
        num_matches([Min, Max], [E]).
get_fact([_|Rest], [E], Factor1):-
        get_fact(Rest, [E], Factor1).

output_modifiers(Factors):-
        Factors=[[]], !,
        write("  There are no modifiers specified for the chosen value."), nl.
output_modifiers(Factors):-
        not(Factors=[]),
        write("  Corresponding modifier(s): "), nl, nl,
        write_lists(Factors), nl.

output_whole_range_modifiers(Factors):-
        Factors=[[]], !,
        write("  There are no modifiers specified for the chosen value."), nl.
output_whole_range_modifiers(Factors):-
        not(Factors=[]),
        write("  Corresponding modifier(s): "), nl, nl,
        write_lists(Factors), nl, nl,
        write("  The above modifier(s) applies to the whole range of values."), nl.



find_all_names_factors(Attribute, Value_list, Names, Factors):-
        write(" Processing knowledge..."), nl,
        findall(Name, find_objects_and_modifiers(Attribute, Value_list, Name, _), Names),
        findall(Factor, find_objects_and_modifiers(Attribute, Value_list, _, Factor), Factors),
        not(Names=[]), !,
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write_names_factors(Names, Factors).
find_all_names_factors(Attribute, [V1, V2], Names, Factors):-
        get_order([V1, V2], [Vmin, Vmax]),
        findall(Name, find_objects_and_modifiers(Attribute, [Vmin], Name, _), Names_min),
        findall(Factor, find_objects_and_modifiers(Attribute, [Vmin], _, Factor), Factors_min),
        findall(Name, find_objects_and_modifiers(Attribute, [Vmax], Name, _), Names_max),
        findall(Factor, find_objects_and_modifiers(Attribute, [Vmax], _, Factor), Factors_max),
        append(Names_min, Names_max, Names),
        append(Factors_min, Factors_max, Factors),
        not(Names=[]), !,
        write(" Press any key to see results..."), nl, nl,
        readchar(_),
        write("  The input range of values does not correspond to a single object!!"), nl, nl,
        write(" Press any key to get answer(s) for the lower range..."), nl, nl,
        readchar(_),
        write("  The lower range (", Vmin, ") corresponds to: "), nl,
        write_names_factors(Names_min, Factors_min), nl, nl,
        write(" Press any key to get answer(s) for the upper range..."), nl, nl,
```

```
        readchar(_),
        write("  The upper range (", Vmax, ") corresponds to: "), nl,
        write_names_factors(Names_max, Factors_max), nl.
find_all_names_factors(Attribute, Value_list, Names, Factors):-
        findall(Name, find_objects_and_modifiers(Attribute, Value_list, Name, _), Names),
        findall(Factor, find_objects_and_modifiers(Attribute, Value_list, _, Factor), Factors),
        Names=[],
        fail.

find_objects_and_modifiers(Attribute, Value_list, Name, Factor):-
        modifier(Name, Mod_attlist),
        get_attlist(Attribute, Mod_attlist),
        get_name_factor(Mod_attlist, Attribute, Value_list, Name, Factor).
find_objects_and_modifiers(Attribute, Value_list, Name, Factor):-
        class(Name, [], Class_attlist),
        not(modifier(Name, _)),
        get_attlist(Attribute, Class_attlist),
        get_name_factor(Class_attlist, Attribute, Value_list, Name, Factor).
find_objects_and_modifiers(Attribute, Value_list, Name, Factor):-
        class(Name, [], Class_attlist),
        modifier(Name, Mod_attlist),
        not(get_attlist(Attribute, Mod_attlist)),
        get_attlist(Attribute, Class_attlist),
        get_name_factor(Class_attlist, Attribute, Value_list, Name, Factor).

get_attlist(Attribute, [att(Attribute, _)|_]):-!.
get_attlist(Attribute, [_|Tail]):-
        get_attlist(Attribute, Tail).

get_name_factor(Attlist, Attribute, Value_list, Name, Factor):-
        get_val_list(Attlist, Attribute, Val_list),
        get_factor(Val_list, Value_list, Factor).
get_name_factor(Attlist, Attribute, Value_list, Name, Factor):-
        get_val_list(Attlist, Attribute, Val_list),
        findall(Fact, get_factor(Val_list, Value_list, Fact), Fact_list),
        Fact_list=[],
        get_mod_f(Val_list, Value_list, Factor).

write_names_factors(Names, Factors):-
        Names=[Name|Tail],
        Tail=[], !,
        write("  Object:  "),
        write(Name), nl,
        Factors=[Factor_list|Rest],
        write_fact_list(Factor_list).
write_names_factors(Names, Factors):-
        Names=[Name|Tail],
        not(Tail=[]),
        Factors=[Factor_list|Rest],
        write("  Object:  "),
        write(Name), nl,
        write_fact_list(Factor_list),
        write(" Press any key to see alternative solutions..."), nl, nl,
        readchar(_),
```

```
        write("  Alternatively, "), nl,
        write_names_factors(Tail, Rest).

write_fact_list(Factor_list):-
        Factor_list=[], !,
        write("   Corresponding modifiers:   No modifiers are defined "), nl, nl.
write_fact_list(Factor_list):-
        Factor_list=[Factor|Remaining],
        Remaining=[], !,
        write("   Corresponding modifier:   ", Factor), nl, nl.
write_fact_list(Factor_list):-
        Factor_list=[_|Remaining],
        not(Remaining=[]),
        write("   Corresponding modifiers: "), nl,
        write_factor_list(Factor_list), nl.

write_factor_list([]):-!.
write_factor_list([H|T]):-
        write("                          ", H), nl,
        write_factor_list(T).
```

/ The clauses below describe the advisory rule (rule investigate) developed to provide assistance in the selection of appropriate in-sistu tests. */*

```
investigate(Soil_category, Parameter, Accuracy, Add_attributes, Soil_names, Test_name, Names,
                Values, Modified_names, Modified_values, Available_attributes, Available_vallists):-
        discover_members(Soil_category, Soil_instances),
        write_soil_names(Soil_instances, Soil_category), nl,
        modified_soil_names(Soil_instances, [], Modified_soil_instances),
        reverse(Soil_instances, Soil_names),
        findall(Method_name, sort_test_name(Parameter, Method_name, Accuracy), Method_names),
        repeat,
        sort_test_name(Parameter, Test_name, Accuracy),
        write(" Processing knowledge..."), nl,
        give_value(Test_name, Soil_names, applicability, Names, Values),
        get_modified_value(Test_name, Modified_soil_instances, applicability, [], Modified_names,
                [], Modified_values),
        get_add_value(Test_name, Add_attributes, [], Available_attributes, [], Available_vallists),
        write(" Press any key to continue..."), nl, nl,
        readchar(_),
        write("   Test name: ", Test_name), nl,
        write_applicability(Soil_names, Names, Values),
        write_mod_applicability(Modified_names, Modified_values), nl,
        write_add_attributes(Add_attributes, Available_attributes, Available_vallists), nl,
        last(Method_names, Test),
        Test_name=Test, !.
```

```
modified_soil_names([], Soil_instances_modifiers, Soil_instances_modifiers).
modified_soil_names(Soil_instances, Old_list, Soil_instances_modifiers):-
        Soil_instances=[Soil_instance|Tail],
        modified_soil(Soil_instance, Soil_instance_modifiers),
        append([Soil_instance_modifiers], Old_list, Temp_list),
        modified_soil_names(Tail, Temp_list, Soil_instances_modifiers).

modified_soil(Soil_name, Modified_soil_list):-
        modifier(Soil_name, Attlist), !,
        find_factors(Attlist, [], Factor_list),
        get_modified_soil(Soil_name, Factor_list, [], Modified_soil_list).
modified_soil(Soil_name, Modified_soil_list):-
        not(modifier(Soil_name, _)),
        Modified_soil_list=[].

get_modified_soil(_, [], Modified_soil_list, Modified_soil_list).
get_modified_soil(Soil_name, [Factor|Tail], Old_list, Modified_soil_list):-
        concat(Factor, "_", Halfstring),
        concat(Halfstring, Soil_name, Wholestring),
        Modified_soil=[Wholestring],
        append(Modified_soil, Old_list, Temp_list),
        get_modified_soil(Soil_name, Tail, Temp_list, Modified_soil_list).

sort_test_name(Parameter, Test_name, Accuracy):-
        modifier(Test_name, Attlist),
        check_parameter(Parameter, Attlist, Accuracy).

check_parameter(Parameter, Attlist, Accuracy):-
        Attlist=[att(reliability, [val([Accuracy], fact(Factors))|_])|_],
        member(Parameter, Factors).
check_parameter(Parameter, [att(Attribute, [_|Rest])|Tail], Accuracy):-
        check_parameter(Parameter, [att(Attribute, Rest)|Tail], Accuracy).
check_parameter(Parameter, [att(_, [])|Tail], Accuracy):-
        check_parameter(Parameter, Tail, Accuracy).

give_value(Test_name, Soil_names, applicability, Names, Values):-
        modifier(Test_name, Attlist), !,
        get_val_list(Attlist, applicability, Val_list),
        Val_list=[val(Valuelist, fact(Parameter))|Rest],
        get_names_values(Val_list, Soil_names, [], Names, [], Values).

get_names_values(_, [], Names, Names, Values, Values).
get_names_values(Val_list, [Soil_name|Tail], Old_name, Names, Old_value, Values):-
        get_soil_value(Val_list, Soil_name, Old_name, Temp_names, Old_value, Temp_values),
        get_names_values(Val_list, Tail, Temp_names, Names, Temp_values, Values).

get_soil_value([], Soil_name, Temp_name, Temp_name, Temp_value, Temp_value).
get_soil_value(Val_list, Soil_name, Old_name, Temp_name, Old_value, Temp_value):-
        Val_list=[val(Valuelist, fact(Parameter))|_],
        member(Soil_name, Parameter),
        append([Soil_name], Old_name, Temp_name),
        append(Valuelist, Old_value, Temp_value).
```

```
get_soil_value(Val_list, Soil_name, Old_name, Temp_name, Old_value, Temp_value):-
        Val_list=[val(_, fact(Parameter))|Rest],
        not(member(Soil_name, Parameter)),
        get_soil_value(Rest, Soil_name, Old_name, Temp_name, Old_value, Temp_value).

get_add_value(_, [], Attrlist, Attrlist, Vallist, Vallist).
get_add_value(Test_name, [Attribute|Tail], Old_attrlist, Attrlist, Old_vallist, Vallist):-
        get_all_attributes(Test_name, tests, [], Class_attlist, Mod_attlist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist),
        member(Attribute, Mod_attrlist),
        get_val_list(Mod_attlist, Attribute, Mod_vallist),
        append([Attribute], Old_attrlist, Temp_attrlist),
        append([Mod_vallist], Old_vallist, Temp_vallist),
        get_add_value(Test_name, Tail, Temp_attrlist, Attrlist, Temp_vallist, Vallist), !.
get_add_value(Test_name, [Attribute|Tail], Old_attrlist, Attrlist, Old_vallist, Vallist):-
        get_all_attributes(Test_name, tests, [], Class_attlist, Mod_attlist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist),
        not(member(Attribute, Mod_attrlist)),
        get_attribute_names(Class_attlist, [], Class_attrlist),
        member(Attribute, Class_attrlist),
        get_val_list(Class_attlist, Attribute, Class_vallist),
        append([Attribute], Old_attrlist, Temp_attrlist),
        append([Class_vallist], Old_vallist, Temp_vallist),
        get_add_value(Test_name, Tail, Temp_attrlist, Attrlist, Temp_vallist, Vallist), !.
get_add_value(Test_name, [Attribute|Tail], Old_attrlist, Attrlist, Old_vallist, Vallist):-
        get_add_value(Test_name, Tail, Old_attrlist, Attrlist, Old_vallist, Vallist).

get_modified_value(_, [], _, Modified_names, Modified_names, Modified_values, Modified_values).
get_modified_value(Test_name, [Soil_name_modifiers|Tail], applicability, Old_names,
                Modified_names, Old_values, Modified_values):-
        not(Soil_name_modifiers=[]),
        give_value(Test_name, Soil_name_modifiers, applicability, Mod_name_list, Mod_value_list),
        append(Mod_name_list, Old_names, Temp_names),
        append(Mod_value_list, Old_values, Temp_values),
        get_modified_value(Test_name, Tail, applicability, Temp_names, Modified_names,
                Temp_values, Modified_values).

get_modified_value(Test_name, [Soil_name_modifiers|Tail], applicability, Old_names,
                Modified_names, Old_values, Modified_values):-
        Soil_name_modifiers=[],
        get_modified_value(Test_name, Tail, applicability, Old_names, Modified_names, Old_values,
                Modified_values).

write_soil_names(Soil_names, Soil_category):-
        not(Soil_names=[_|[]]), !,
        write(" The members of the category ", Soil_category, " are: "), nl, nl,
        write_children(Soil_names, 1).
write_soil_names(Soil_names, Soil_category):-
        Soil_names=[H|[]],
        not(class(H, [], _)), !,
        write(" The category ", Soil_category, " has one member which is:"), nl, nl,
        write_children(Soil_names, 1).
write_soil_names(_, _).
```

```
write_applicability(Soil_names, Soil_type, Applicability_value):-
        write("     The applicability of this test in "), nl,
        write_app(Soil_type, Applicability_value),
        delete_list(Soil_type, Soil_names, Soils_left),
        check_soils_left(Soils_left).

write_app([], []).
write_app([Soil|R1], [Applicability|R2]):-
        write("                    ", Soil, " is ", Applicability), nl,
        write_app(R1, R2).

check_soils_left(Soils_left):-
        not(Soils_left=[]), !,
        write_non_app(Soils_left).
check_soils_left([]).

write_non_app([]).
write_non_app([Soil_left|R]):-
        write("                    ", Soil_left, " is  unspecified"), nl,
        write_non_app(R).

write_mod_applicability([], []):-!.
write_mod_applicability(Modified_soil_type, Modified_applicability_value):-
        write("     It should be noted though that the applicability in "), nl,
        write_mod_app(Modified_soil_type, Modified_applicability_value).

write_mod_app([], []).
write_mod_app([Mod_soil|R1], [Mod_Applicability|R2]):-
        write("                    ", Mod_Soil, " is ", Mod_Applicability), nl,
        write_mod_app(R1, R2).

write_add_attributes([], _, _):-!.
write_add_attributes(Add_attributes, Available_add_attribute, Available_add_vallist):-
        not(Available_add_attribute=[]), !,
        write(" Press any key to continue..."), nl, nl,
        readchar(_),
        write("    Additional attributes under consideration: "), nl,
        write_add_attr(Available_add_attribute, Available_add_vallist),
        delete_list(Available_add_attribute, Add_attributes, Attributes_left),
        check_attributes_left(Attributes_left).
write_add_attributes(Add_attributes, Available_add_attribute, Available_add_vallist):-
        delete_list(Available_add_attribute, Add_attributes, Attributes_left),
        check_attributes_left(Attributes_left).

write_add_attr([], []):-!.
write_add_attr([Attribute|R1], [Vallist|R2]):-
        Vallist=[val([], fact([]))], !,
        write("     No values have been specified for the attribute ", Attribute), nl,
        write_add_attr(R1, R2).
```

```
write_add_attr([Attribute|R1], [Vallist|R2]):-
        write("     The attribute ", Attribute, " has "),
        Vallist=[H|T],
        T=[],
        write_values(H),
        write_add_attr(R1, R2).

check_attributes_left([]):-!.
check_attributes_left(Attributes_left):-
        write_non_attr(Attributes_left).

write_non_attr([]).
write_non_attr([Attribute_left|T]):-
        write("     The attribute ", Attribute_left, " is not defined for this test"), nl,
        write_non_attr(T).



/* The clauses below describe additional search rules required by the user interface. */


find_all_roots(Roots):-
        findall(Root, find_root(Root), Roots).

find_root(Root):-
        class(Root, List, Attlist),
        not(List=[]),
        Attlist=[].



find_root_tree(Root, Roots, List, Root_tree):-
        first(Roots, Root),
        Roots=[Root, Next|_],
        split_list(Next, List, Root_tree, _).
find_root_tree(Root, Roots, List, Root_tree):-
        last(Roots, Root),
        split_list(Root, List, _, Root_tree).
find_root_tree(Root, Roots, List, Root_tree):-
        get_root_tree(Root, Roots, List, Root_tree).

get_root_tree(Root, Roots, List, Root_tree):-
        Roots=[_, Root2, Root3|Rest],
        Root=Root2,
        split_list(Root2, List, _, Lb),
        split_list(Root3, Lb, Root_tree, _).
get_root_tree(Root, Roots, List, Root_tree):-
        Roots=[_, Root2, Root3|Rest],
        Tail=[Root2, Root3|Rest],
        get_root_tree(Root, Tail, List, Root_tree).



find_all_attrib_names(Name, Root, Attributes):-
```

```
        findall(Attribute, find_attrib_name(Name, Root, [], Attribute), Attrs),
        remove_duplicates(Attrs, [], Attributes).

find_attrib_name(Name, Root, Oldlist, Attribute):-
        get_all_attributes(Name, Root, [], Class_attlist, Mod_attlist),
        get_attribute_names(Class_attlist, Oldlist, Class_attrlist),
        get_attribute_names(Mod_attlist, Oldlist, Mod_attrlist),
        append(Class_attrlist, Mod_attrlist, Attrlist),
        members(Attribute, Attrlist).



get_parents(Parents):-
        class(X, List, _),
        not(List=[]),
        Parents=X.



get_all_names_with_factors(Names, Root, Roots):-
        findall(F_name, get_names_with_factors(F_name, Root, Roots), F_names),
        remove_duplicates(F_names, [], Names_list),
        reverse(Names_list, Names).
get_names_with_factors(F_name, Root, Roots):-
        findall(X, class(X, _, _), Names),
        find_root_tree(Root, Roots, Names, Root_tree),
        members(Name, Root_tree),
        class(Name, _, List),
        not(List=[]),
        get_all_attributes(Name, Root, [], Class_attlist, Mod_attlist),
        append(Class_attlist, Mod_attlist, Attlist),
        find_factors(Attlist, [], Fact_list),
        not(Fact_list=[]),
        Name=F_name.

find_factors([], List2, List2).
find_factors(A_list, List1, List2):-
        A_list=[att(Attribute, Vallist)|Tail],
        Vallist=[val(_, fact(Factors))|Rest],
        not(Rest=[]),
        append(Factors, List1, Templist),
        find_factors([att(Attribute, Rest)|Tail], Templist, List2).
find_factors(A_list, List1, List2):-
        A_list=[att(Attribute, Vallist)|Tail],
        Vallist=[val(_, fact(Factors))|Rest],
        Rest=[],
        append(Factors, List1, Templist),
        find_factors(Tail, Templist, List2).



get_all_fact_list(Name, Root, Factors):-
        findall(Factor, get_fact_list(Name, Root, [], Factor), Facts),
        remove_duplicates(Facts, [], Factors).
```

```
get_fact_list(Name, Root, Oldlist, Factor):-
        get_all_attributes(Name, Root, [], Class_attlist, Mod_attlist),
        append(Class_attlist, Mod_attlist, Attlist),
        find_factors(Attlist, Oldlist, Fact_list),
        members(Factor, Fact_list).



get_fact_attribute_list(Name, Mod_attrlist):-
        modifier(Name, Mod_attlist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist).



find_all_mod_attributes(Root, Roots, Mod_attributes_list):-
        findall(Mod_attrlist, get_mod_attributes(Root, Roots, Mod_attrlist), Mod_attrlists),
        simplify_lists(Mod_attrlists, [], Mod_attr_list),
        remove_duplicates(Mod_attr_list, [], Mod_attributes_list).

get_mod_attributes(Root, Roots, Mod_attrlist):-
        findall(X, class(X, _, _), List),
        find_root_tree(Root, Roots, List, Root_tree),
        findall(Y, class(Y, [], _), Names),
        find_instances(Names, Root_tree, [], Instances_list),
        members(Instance, Instances_list),
        modifier(Instance, Mod_attlist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist).

find_instances([], _, Instances_list, Instances_list).
find_instances([Name|Rest], Root_tree, Old_list, Instances_list):-
        member(Name, Root_tree),
        append([Name], Old_list, Temp_list),
        find_instances(Rest, Root_tree, Temp_list, Instances_list).
find_instances([Name|Rest], Root_tree, Old_list, Instances_list):-
        not(member(Name, Root_tree)),
        find_instances(Rest, Root_tree, Old_list, Instances_list).



find_unique_attribute_data(Attribute, Values, Factors):-
        findall(Value_list, find_attribute_data(Attribute, Value_list, _), Values_lists),
        findall(Factor_list, find_attribute_data(Attribute, _, Factor_list), Factors_lists),
        simplify_lists(Values_lists, [], Values_list),
        simplify_lists(Factors_lists, [], Factors_list),
        remove_duplicates(Values_list, [], Values),
        remove_duplicates(Factors_list, [], Factors).

find_attribute_data(Attribute, Value_list, Factor_list):-
        modifier(Name, Attlist),
        get_val_list(Attlist, Attribute, Vallist),
        get_attribute_data(Vallist, [], Value_list, [], Factor_list).

get_attribute_data([], Value_list, Value_list, Factor_list, Factor_list).
```

```
get_attribute_data([val(Value, fact(Factor))|Rest], Old_value, Value_list, Old_factor, Factor_list):-
        append(Value, Old_value, Temp_value),
        append(Factor, Old_factor, Temp_factor),
        get_attribute_data(Rest, Temp_value, Value_list, Temp_factor, Factor_list).



find_all_test_attributes(Attributes):-
        findall(Class_attrlist, find_test_attributes(Class_attrlist, _), Class_attrs),
        findall(Mod_attrlist, find_test_attributes(_, Mod_attrlist), Mod_attrs),
        simplify_lists(Class_attrs, [], Class_attribs),
        simplify_lists(Mod_attrs, [], Mod_attribs),
        remove_duplicates(Class_attribs, [], Class_attributes),
        remove_duplicates(Mod_attribs, [], Mod_attributes),
        append(Class_attributes, Mod_attributes, Attributes).

find_test_attributes(Class_attrlist, Mod_attrlist):-
        findall(X, class(X, _, _), List),
        find_all_roots(Roots),
        find_root_tree(tests, Roots, List, Root_tree),
        members(Name, Root_tree),
        find_test_attrs(Name, Class_attrlist, Mod_attrlist).

find_test_attrs(Name, Class_attrlist, Mod_attrlist):-
        modifier(Name, Mod_attlist), !,
        class(Name, _, Class_attlist),
        get_attribute_names(Class_attlist, [], Class_attrlist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist).
find_test_attrs(Name, Class_attrlist, Mod_attrlist):-
        class(Name, _, Class_attlist),
        get_attribute_names(Class_attlist, [], Class_attrlist),
        Mod_attrlist=[].



find_all_general_range(Attribute, Old_list, Old_range, General_range):-
        findall(Range1, get_general_range(Attribute, Range1, _), Ranges1),
        findall(Range2, get_general_range(Attribute, _, Range2), Ranges2),
        simplify_lists(Ranges1, [], Ranges1_list),
        simplify_lists(Ranges2, [], Ranges2_list),
        append(Ranges1_list, Old_list, Temp_list),
        append(Ranges2_list, Temp_list, Ranges),
        remove_duplicates(Ranges, [], Range_list),
        min_number(Range_list, Min),
        max_number(Range_list, Max),
        append([Max], Old_range, Temp_range),
        append([Min], Temp_range, General_range).

get_general_range(Attribute, Range1, Range2):-
        class(X, [], Class_attlist),
        get_attribute_names(Class_attlist, [], Class_attrlist),
        member(Attribute, Class_attrlist),
        num_value_attr(X, Attribute, [], Range1),
        modifier(Name, Mod_attlist),
```

```
        get_attribute_names(Mod_attlist, [], Mod_attrlist),
        member(Attribute, Mod_attrlist),
        num_value_attr(Name, Attribute, [], Range2).
get_general_range(Attribute, [], Range2):-
        modifier(Name, Mod_attlist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist),
        member(Attribute, Mod_attrlist),
        num_value_attr(Name, Attribute, [], Range2).



convert_input(Input, Values):-
        fronttoken(Input, Input, ""),
        Values=[Input].
convert_input(Input, Values):-
        fronttoken(Input, Vmin, Rest),
        fronttoken(Rest, _, Vmax),
        Values=[Vmin, Vmax].



find_all_num_value_attr(Name, Attribute, Ranges):-
        findall(Range, num_value_attr(Name, Attribute, [], Range), Ranges_list),
        simplify_lists(Ranges_list, [], Range_list),
        remove_duplicates(Range_list, [], Rangelist),
        reverse(Rangelist, Ranges).

num_value_attr(Name, Attribute, Old_range, Range):-
        modifier(Name, Attlist),
        get_val_list(Attlist, Attribute, Val_list),
        find_num_values(Val_list, [], Value_list),
        remove_duplicates(Value_list, [], Values),
        min_number(Values, Vmin),
        max_number(Values, Vmax),
        append([Vmax], Old_range, Temp_range),
        append([Vmin], Temp_range, Range).
num_value_attr(Name, Attribute, Old_range, Range):-
        class(Name, _, Attlist),
        get_val_list(Attlist, Attribute, Val_list),
        find_num_values(Val_list, [], Value_list),
        remove_duplicates(Value_list, [], Values),
        min_number(Values, Vmin),
        max_number(Values, Vmax),
        append([Vmax], Old_range, Temp_range),
        append([Vmin], Temp_range, Range).

find_num_values([], Values, Values).
find_num_values([val([V1, V2], _)|Rest], Old_values, Values):-
        str_real(V1, V1r),
        str_real(V2, V2r),
        append([V1r], Old_values, Temp_values),
        append([V2r], Temp_values, New_values),
        find_num_values(Rest, New_values, Values).
```

```
find_all_sym_values(Attribute, Value_list):-
        findall(Values, get_sym_values(Attribute, Values), Values_lists),
        simplify_lists(Values_lists, [], Values_list),
        remove_duplicates(Values_list, [], Valuelist),
        reverse(Valuelist, Value_list).

get_sym_values(Attribute, Values):-
        modifier(Name, Mod_attlist),
        get_attribute_names(Mod_attlist, [], Mod_attrlist),
        member(Attribute, Mod_attrlist),
        sym_value_attr(Name, Attribute, Values).

sym_value_attr(Name, Attribute, Values):-
        modifier(Name, Attlist),
        get_val_list(Attlist, Attribute, Val_list),
        find_sym_values(Val_list, [], Values).

find_sym_values([], Temp_values, Values):-
        reverse(Temp_values, Values).
find_sym_values([val([Value], _)|Rest], Old_values, Values):-
        not(str_real(Value, _)),
        append([Value], Old_values, Temp_values),
        find_sym_values(Rest, Temp_values, Values).



/* The clauses below describe the rules required to develop the user interface of the system. */


match_choices(_, [], New_list, Return_list):-
        reverse(New_list, Return_list).
match_choices(List1, [Head|R], Old_list, Return_list):-
        delete_item(0, R, Rest),
        First=Head-1,
        match_choice(List1, First, Item),
        append([Item], Old_list, Temp_list),
        match_choices(List1, Rest, Temp_list, Return_list).

match_choice([Item|_], 0, Item).
match_choice([_|Tail], Length, Item):-
        Length1=Length-1,
        match_choice(Tail, Length1, Item).



user_interface:-
        L=["Query Knowledge Bases", "Assist Selection of In-Situ Tests"],
        makestatus(112, " Arrow keys:Inspect items  Enter:Select  Esc:Return to previous menu or
                exit"),
        longmenu_repeat(1, 3, 2, 7, 7, L, "options", 1, Option),
        check_option(Option),
        fail.
user_interface.
```

```
check_option(Option):-
        Option=1,
        removestatus,
        Action_list=["get attributes", "find ancestors", "discover members", "find attribute and value",
                "find modifiers", "find objects and modifiers"],
        makestatus(112, " Arrow keys:Inspect items  Enter:Select  Esc:Return to previous menu or
                exit"),
        longmenu_repeat(4, 2, 6, 7, 7, Action_list, "actions", 1, Choice),
        find_all_roots(Roots),
        longmenu_repeat(11, 3, 5, 7, 7, Roots, "knowledge bases", 1, Choice1),
        Selection1=Choice1-1,
        match_choice(Roots, Selection1, Root),
        check_selection(Choice, Root, Roots).

check_option(Option):-
        Option=2,
        removestatus,
        findall(X, class(X, _, _), Object_list),
        split_list(tests, Object_list, Ground_objects, _),
        makestatus(112, " Arrow keys:Inspect items  Enter:Select  Esc:Return to previous menu or
                exit"),
        longmenu_repeat(1, 43, 5, 7, 7, Ground_objects, "ground conditions", 1, Choice1),
        Selection1=Choice1-1,
        match_choice(Ground_objects, Selection1, Soil_category),
        find_unique_attribute_data(reliability, Values, Factors),
        longmenu_repeat(7, 5, 6, 7, 7, Factors, "geotechnical information", 1, Choice2),
        Selection2=Choice2-1,
        match_choice(Factors, Selection2, Parameter),
        longmenu_repeat(9, 35, 4, 7, 7, Values, "reliability", 1, Choice3),
        Selection3=Choice3-1,
        match_choice(Values, Selection3, Reliability),
        find_all_test_attributes(Attribute_list),
        delete_list([test_name, applicability, reliability], Attribute_list, Additional_attributes),
        makestatus(112, " Multiple selection allowed.  F10:End selections  Esc:No selections"),
        longmenu_mult(16, 15, 5, 7, 7, Additional_attributes, "additional attributes to be considered",
                [0], Choices),
        match_choices(Additional_attributes, Choices, [], Selected_attributes),
        makewindow(2, 79, 7, "Answer", 0, 1, 24, 78),
        removestatus,
        makestatus(112, ""),
        investigate(Soil_category, Parameter, Reliability, Selected_attributes, _, _, _, _, _, _, _, _),
        removestatus,
        makestatus(112, "press any key to return to the previous menu"),
        readchar(_),
        removestatus,
        removewindow.
```

```
check_selection(Choice, Root, Roots):-
        Choice=1,
        findall(X, class(X, _, _), List),
        find_root_tree(Root, Roots, List, Root_tree),
        Root_tree=[RootlRest],
        concat(Root, " tree", Label),
        longmenu_repeat(6, 30, 7, 7, 7, Rest, Label, 1, Choice2),
        Selection2=Choice2-1,
        match_choice(Rest, Selection2, Object),
        find_all_attrib_names(Object, Root, Attribute_list),
        longmenu_repeat(14, 52, 7, 7, 7, Attribute_list, "attributes", 1, Choice3),
        Selection3=Choice3-1,
        match_choice(Attribute_list, Selection3, Attribute),
        makewindow(2, 79, 7, "Answer", 0, 1, 24, 79),
        makestatus(112, ""),
        find_vallists(Object, Root, Attribute), nl, nl,
        removestatus,
        makestatus(112, "press any key to return to the previous menu"),
        readchar(_),
        removestatus,
        removewindow.

check_selection(Choice, Root, Roots):-
        Choice=2,
        findall(X, class(X, _, _), List),
        find_root_tree(Root, Roots, List, Root_tree),
        concat(Root, " tree", Label),
        longmenu_repeat(6, 30, 15, 7, 7, Root_tree, Label, 1, Choice2),
        Selection2=Choice2-1,
        match_choice(Root_tree, Selection2, Object),
        makewindow(2, 79, 7, "Answer", 1, 1, 23, 78),
        makestatus(112, ""),
        find_all_ancestors(Object, [], _),
        removestatus,
        makestatus(112, "press any key to return to the previous menu"),
        readchar(_),
        removestatus,
        removewindow.

check_selection(Choice, Root, Roots):-
        Choice=3,
        findall(X, get_parents(X), List),
        find_root_tree(Root, Roots, List, Root_tree),
        longmenu_repeat(6, 30, 15, 7, 7, Root_tree, "classes", 1, Choice2),
        Selection2=Choice2-1,
        match_choice(Root_tree, Selection2, Object),
        makewindow(2, 79, 7, "Answer", 1, 1, 23, 78),
        makestatus(112, ""),
        get_members(Object, _),
        removestatus,
        makestatus(112, "press any key to return to the previous menu"),
        readchar(_),
        removestatus,
        removewindow.
```

```
check_selection(Choice, Root , Roots):-
        Choice=4,
        removestatus,
        makestatus(112, " Please wait..."),
        get_all_names_with_factors(Names, Root, Roots),
        removestatus,
        makestatus(112, " Arrow keys:Inspect items  Enter:Select  Esc:Return to previous menu or
                exit"),
        longmenu_repeat(6, 30, 7, 7, 7, Names, "instances having modifiers", 1, Choice2),
        Selection2=Choice2-1,
        match_choice(Names, Selection2, Name),
        get_all_fact_list(Name, Root, Factors),
        longmenu_repeat(14, 45, 7, 7, 7, Factors, "modifiers", 0, Choice3),
        Selection3=Choice3-1,
        match_choice(Factors, Selection3, Factor),
        makewindow(2, 79, 7, "Answer", 1, 1, 23, 78),
        makestatus(112, "press any key to return to the previous menu"),
        find_attribute_and_value(Name, Factor, [], _),
        readchar(_),
        removestatus,
        removewindow.

check_selection(Choice, Root, Roots):-
        Choice=5,
        removestatus,
        makestatus(112, " Please wait..."),
        get_all_names_with_factors(Names, Root, Roots),
        removestatus,
        makestatus(112, " Arrow keys:Inspect items  Enter:Select  Esc:Return to previous menu or
                exit"),
        longmenu_repeat(6, 30, 7, 7, 7, Names, "instances having modifiers", 1, Choice2),
        Selection2=Choice2-1,
        match_choice(Names, Selection2, Name),
        get_fact_attribute_list(Name, Mod_attrlist),
        longmenu_repeat(14, 45, 5, 7, 7, Mod_attrlist, "attributes", 0, Choice3),
        Selection3=Choice3-1,
        match_choice(Mod_attrlist, Selection3, Attribute),
        case(Name, Attribute).

check_selection(Choice, Root, Roots):-
        Choice=6,
        find_all_mod_attributes(Root, Roots, Attributes),
        longmenu_repeat(6, 35, 4, 7, 7, Attributes, "attributes defined with modifiers", 1, Choice2),
        Selection2=Choice2-1,
        match_choice(Attributes, Selection2, Attribute),
        situation(Attribute).
```

```
case(Name, Attribute):-
        find_all_num_value_attr(Name, Attribute, [Vminr, Vmaxr]), !,
        str_real(Vmin, Vminr),
        str_real(Vmax, Vmaxr),
        concat("Enter value(s) (", Vmin, String1),
        concat(String1, ", ", String2),
        concat(String2, Vmax, String3),
        concat(String3, ") : ", String4),
        tempstatus(112, " Type in a value or a range of values (V1, V2)"),
        lineinput_repeat(20, 25, 50, 7, 7, String4, "", Input),
        convert_input(Input, Values),
        makewindow(2, 79, 7, "Answer", 1, 1, 23, 79),
        makestatus(112, "press any key to return to the previous menu"),
        condition(Name, Attribute, Values).
case(Name, Attribute):-
        sym_value_attr(Name, Attribute, Values),
        makestatus(112, " Arrow keys:Inspect items  Enter:Select  Esc:Return to previous menu or
                exit"),
        longmenu_repeat(18, 55, 4, 7, 7, Values, "Values", 1, Selection2),
        Choice3=Selection2-1,
        match_choice(Values, Choice3, Value),
        makewindow(2, 79, 7, "Answer", 1, 1, 23, 78),
        makestatus(112, "press any key to return to the previous menu"),
        find_modifiers(Name, Attribute, [Value], _),
        readchar(_),
        removestatus,
        removewindow.

condition(Name, Attribute, Values):-
        find_modifiers(Name, Attribute, Values, _), !,
        readchar(_),
        removestatus,
        removewindow.
condition(Name, Attribute, Values):-
        write("          **********"), nl,
        write("          *    Error    *"), nl,
        write("          **********"), nl, nl, nl,
        write("  Your input is incorrect!! Try again."), nl,
        readchar(_),
        removestatus,
        removewindow.

situation(Attribute):-
        makestatus(112, " Please wait..."),
        find_all_general_range(Attribute, [], [], [Vminr, Vmaxr]), !,
        str_real(Vmin, Vminr),
        str_real(Vmax, Vmaxr),
        concat("Enter value(s) (", Vmin, String1),
        concat(String1, ", ", String2),
        concat(String2, Vmax, String3),
        concat(String3, ") : ", String4),
        removestatus,
        tempstatus(112, " Type in a value or a range of values (V1, V2)"),
        lineinput_repeat(14, 25, 50, 7, 7, String4, "" , Input),
```

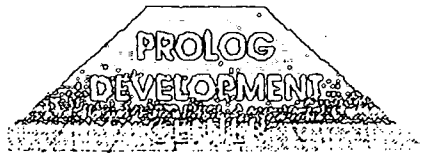
```
        convert_input(Input, Values),
        makewindow(2, 79, 7, "Answer", 1, 1, 23, 78),
        makestatus(112, ""),
        state(Attribute, Values).
situation(Attribute):-
        find_all_sym_values(Attribute, Values),
        removestatus,
        longmenu_repeat(11, 45, 4, 7, 7, Values, "Values", 1, Selection2),
        Choice3=Selection2-1,
        match_choice(Values, Choice3, Value),
        makewindow(2, 79, 7, "Answer", 1, 1, 23, 78),
        makestatus(112, ""),
        find_all_names_factors(Attribute, [Value], _, _),
        removestatus,
        makestatus(112, "press any key to return to the previous menu"),
        readchar(_),
        removestatus,
        removewindow.

state(Attribute, Values):-
        find_all_names_factors(Attribute, Values, _, _), !,
        removestatus,
        makestatus(112, "press any key to return to the previous menu"),
        readchar(_),
        removestatus,
        removewindow.
state(Attribute, Values):-
        removestatus,
        write("          **********"), nl,
        write("          *    Error    *"), nl,
        write("          **********"), nl, nl, nl,
        write("  Your input is incorrect!! Try again."), nl,
        makestatus(112, "press any key to return to the previous menu"),
        readchar(_),
        removestatus,
        removewindow.


GOAL
textmode(R, C), R1=R-1,
makewindow(1, 79, 0, "test", 0, 0, R1, C),
user_interface,
changestatus("End of execution").
```

APPENDIX B

PDC PROLOG FAX



Technical Support

From: Prolog Development Center A/S
H.J. HolstVej 5a
DK 2605 Broendby
Denmark
Phone +45 36 72 10 22
Facsimile +45 36 72 02 69

To: Marina Moula
School of Engineering and
Computer Science
University of Durham,
South Road,
Durham DH1 3LE, England
Phone: 091/3742000 ext. 4235
Fax: + 44 91 374 2550
E-mail: marina.moula@durham.ac.uk

Received (date):09-11-92
Sent (date):12-11-92
Service Request No.: 11008

Question
Bug detected

Answer

We are sorry that the bug has causes you problems. We have detected that there is a bug, but it is not fixed yet.

You can avoid the bug by a little change in your file GRTESKB.PRO

1 Declare a dabase predicate:

```
predicates
class(symbol,list,attlist)
modifier(symbol,attlist)
database - modif        % Suggested by PDC
modif(symbol,attlist)   % Suggested by PDC
clauses
```

2) change the modifier predicate to:

```
modifier(organic_sand,X):-
        modifier(sand,X).
modifier(organic_silt,X):-
        modifier(silt,X).
modifier(organic_clay,X):-
        modifier(clay,X).
modifier(A,B):-modif(A,B).      % Suggested by PDC
```

3) Rename the rest of the modifier clauses to:

```
modif(rising_head_test,
        [att(applicability,
           [val([high],fact([])),
            val([medium],fact([])),
            val([low],fact([])),
            val([none],fact([]))]),
         att(reliability,
           [val([high],fact([])),
            val([medium],fact([])),
            val([low],fact([])),
            val([none],fact([]))])]).
modif(falling_head_test,
        [att(applicability,
           [val([high],fact([])),
            val([medium],fact([])),
```

Extra questions

> 1) At the moment all the facts required by the program are included in the
> GRTESKB.PRO file. Would it be more appropriate to use internal databases for
> their storage although they represent static knowledge, or not?

Yes it is normally more efficient to declare static facts as internal database predicates.

The Service Request (SR) number given above should be used in all further correspondance about this matter to PDC Technical Support. PDC Technical Support uses the SR number for proper tracking and computing of your correspondances. Without this number, we cannot properly service your request.

Best Regards
Leo Jensen

APPENDIX C

DIFFERENT VERSIONS OF THE IN-SITU TESTS HIERARCHY

IN-SITU TESTS (First Version)

Borehole Tests

- Permeability Tests
  - Open Borehole Tests
    - Variable Head Test
      - Rising Head Test
      - Falling Head Test
    - Constant Head Test
  - Constant Head Test from Piezometers
- Standard Penetration Test (SPT)
- Vane Test
- Pressuremeter Tests
  - Ménard-type Pressuremeter Test
  - Camcometer Test
  - Stressprobe Pressuremeter Test
- Plate Tests
  - Large Diameter Borehole Plate Test
  - Small Diameter Borehole Plate Test

Probing Tests

- Vane Test
- Penetration Tests
  - Static Cone Penetration Test (CPT)
  - Dynamic Cone Penetration Test (DCP)
  - Static-Dynamic Penetration Test

Non-Borehole Field tests

- Pumping Tests
- In-situ Stress Measurements
  - Hydraulic Pressure Cells
  - Hydraulic Fracturing

- Bearing Tests
  - Vertical Loading Test
  - Lateral and Inclined Loading Tests
  - Pressurized Chamber Test
  - In-situ California Bearing Ratio Test (CBR)
- In-situ Shear Test
- In-situ Density Tests
  - Sand Replacement Tests
    - Small Pouring Cylinder
    - Large Pouring Cylinder
    - Scoop Test
  - Core Cutter Test
  - Weight in Water Test
  - Water Replacement Test
  - Rubber Balloon Test
  - Nuclear Tests

Geophysical Surveying

- Seismic Tests
  - Seismic Refraction Test
  - Seismic Reflection Test
- Resistivity Test
- Gravimetric Test
- Magnetic Test

IN-SITU TESTS (Fourth Version)

Borehole Tests

Pre-bored Tests

- Permeability Tests
  - Open Borehole Tests
    - Variable Head Test
      - Rising Head Test
      - Falling Head Test
    - Constant Head Test
  - Constant Head Test from Piezometers
- Standard Penetration Test (SPT)
- Vane Test
- Borehole Shear Test
- Pressuremeter Tests
- Ménard-type Pressuremeter Test
- Push-in Pressuremeter Test
- Plate Tests
  - Large Diameter Borehole Plate Test
  - Small Diameter Borehole Plate Test
- Screw Plate Test (Field Compressometer Test)

Self-boring Tests

- Pressuremeter Tests
  - Self-boring Pressuremeter Test
- In-situ Stress Measurements
  - Ko meter Test
- Self-boring Permeameter Test
- Self-boring Vane Test
- Plate Tests
  - Self-boring Plate Test

Probing Tests

- Vane Test

- Penetration Tests
  - Dynamic Cone Penetration Test (DCP)
  - Static Cone Penetration Test (CPT)
    - Mechanical Cone Penetration Test
      - Mechanical Cone Resistance Test
      - Mechanical Cone Resistance Friction Test
    - Electrical Cone Penetration Test
      - Electrical Cone Resistance Test
      - Electrical Cone Resistance Friction Test
      - Piezocone Test
      - Piezocone Friction Test
  - Static-Dynamic Penetration Test
- Flat Plate Dilatometer Test
- In-situ Stress Measurements
  - Total Stress Cell Test (Earth Pressure Cell)
  - Ko Stepped Blade Test (Iowa Stepped Blade)
- Special Penetrometer Probes
  - Cone Pressuremeter Test (Pressio-Penetrometer)
  - Nuclear Density Probe Test
  - Electrical Density Probe Test
  - Electrical Conductivity Cone Test
  - Thermal Conductivity Cone Test
  - Acoustic Cone Test

Non-Borehole Field Tests

- Pumping Tests
- In-situ Stress Measurements
  - Hydraulic Fracturing
- Bearing Tests
  - Vertical Loading Test
  - Lateral and Inclined Loading Tests
  - Pressurized Chamber Test
  - In-situ California Bearing Ratio Test (CBR)
- In-situ Shear Test

- In-situ Density Tests
  - Sand Replacement Tests
    - Small Pouring Cylinder Test
    - Large Pouring Cylinder Test
    - Scoop Test
  - Core Cutter Test
  - Weight in Water Test
  - Water Replacement Test
  - Rubber Balloon Test
  - Nuclear Tests
    - Backscatter Test
    - Direct Transmission Test
    - Air Gap Test

Geophysical Surveying

- Seismic Tests
  - Seismic Refraction Test
  - Seismic Reflection Test
  - Seismic Cross-Hole Test
  - Seismic Down-Hole Test
  - Surface Wave Test
- Resistivity Test
- Gravimetric Test
- Magnetic Test

IN-SITU TESTS (Eighth Version-Final) (test category)

Penetration Tests (test nature)

- Standard Penetration Test (SPT) (test name)
- Dynamic Probing Test (DP) (test group)
  - Dynamic Probing Light Test (DPL) (test name)
  - Dynamic Probing Medium Test (DPM) (test name)
  - Dynamic Probing Heavy Test (DPH) (test name)
  - Dynamic Probing Superheavy Test (DPSH) (test name)
- Cone Penetration Test (CPT) (test group)
  - Mechanical Penetrometer Friction Test (test name)
  - Electrical Cone Penetration Tests (test type)
    - Electrical Penetrometer Friction Test (test name)
    - Piezocone Test (CPTU) (test name)
    - Piezocone Friction Test (test name)
- Weight Sounding Test (WST) (test name)
- Static-Dynamic Penetration Test (test name)

Special Penetrometer Tests (test nature)

- Expansion Penetration Tests (test group)
  - Flat Plate Dilatometer Test (DMT) (test name)
  - Cone Pressuremeter Test (test name)
- Lateral Stress Cone Test (LSSCP) (test name)
- Seismic Cone Test (SCPT) (test name)
- Vibratory Cone Test (CPTV) (test name)
- Density Probe Tests (test group)
  - Nuclear Density Probe Test (NCDT) (test name)
  - Electrical Density Probe Test (test name)
- Electrical Conductivity Cone Test (test name)
- Thermal Conductivity Cone Test (test name)
- Acoustic Cone Test (ACPT) (test name)

Pressuremeter Tests (test nature)

- Ménard-type Pressuremeter Test (PMT) (test name)
- Push-in Pressuremeter Test (PIP) (test name)
- Self-boring Pressuremeter Test (SBP) (test name)

In-situ Stress Measurement Tests (test nature)

- Total Stress Cell Test (TSC) (test name)
- Iowa Stepped Blade Test (ISB) (test name)
- Hydraulic Fracturing Test (HFT) (test name)
- Self-boring Ko meter Test (test name)

Shear Tests (test nature)

- Vane Test (test name)
- Self-boring Vane Test (test name)
- Borehole Shear Test (BST) (test name)
- In-situ Shear Test (test name)

Bearing Tests (test nature)

- Plate Loading Tests (PLT) (test name)
- Screw Plate Test (SPLT)) (test name)
- Self-boring Plate Test (test name)
- Pressurized Chamber Test (test name)
- In-situ California Bearing Ratio Test (CBR) (test name)

In-situ Density Tests (test nature)

- Sand Replacement Tests (test group)
  - Small Pouring Cylinder Test (test name)
  - Large Pouring Cylinder Test (test name)
  - Scoop Test (test name)
- Core Cutter Test (test name)
- Weight in Water Test (test name)
- Water Replacement Test (test name)
- Rubber Balloon Test (test name)
- Nuclear Tests (test group)
  - Backscatter Test (test name)
  - Direct Transmission Test (test name)
  - Air Gap Test (test name)

- **Permeability Tests** (test nature)
  - Borehole Tests (test group)
    - Variable Head Test (test type)
      - Rising Head Test (test name)
      - Falling Head Test (test name)
    - Constant Head Test (test name)
  - Self-boring Permeameter Test (test name)
  - Pumping Tests (test name)

- **Geophysical Surveying Tests** (test nature)
  - Seismic Tests (test group)
    - Seismic Refraction Test (test name)
    - Seismic Reflection Test (test name)
    - Seismic Cross-Hole Test (SCT) (test name)
    - Seismic Down-Hole Test (SDS) (test name)
    - Surface Wave Test (test name)
  - Resistivity Test (test name)
  - Gravimetric Test (test name)
  - Magnetic Test (test name)

APPENDIX D

DESCRIPTION OF IN-SITU TESTS

Brief Description of In-situ Tests

PENETRATION TESTS

Several penetration testing methods have been developed and are used at present all over the world (Broms and Flodin, 1988)[1] . The interpretation of the results of penetration tests is mainly empirical. As Meigh (1989) remarks, an empirical approach can only be successful if the test procedures are standardised to a large degree. Recommended standards on the test methods were put forward by the International Society of Soil Mechanics and Foundation Engineering Subcommittee on the penetration test for use in Europe (ISSMFE, 1977). The subcommittee on the standardization of Penetration testing in Europe recommended four standard penetration testing methods:

- Cone Penetration Test (CPT)
- Dynamic Probing Test (DP)
- Standard Penetration Test (SPT)
- Weight Sounding Test (WST)

A draft international reference test procedure for penetration testing, heading towards the finalization of the standardization of penetration testing, was published in 1988 (ISSMFE, 1988). It is based on the recommended standards of the European Subcommittee but includes recent developments, such as the piezocone.

The test hierarchy presented in section 4.2, based on the recommendations given in the European and the International standards, includes the penetration tests mentioned earlier, as well as an additional penetration testing method, as is included in the British Standards (BS 5930, 1981), called:

- Static-Dynamic Penetration Test

A brief description of these tests is given below.

---

[1] Note: Appendix D has a separate reference list.

Cone Penetration Test (CPT)

The Cone Penetration Test consists of pushing into the soil, at a sufficiently slow rate, a series of cylindrical rods with a cone at the base, and measuring continuously, or at selected depth intervals, the penetration resistance, q_c, of the cone and if required the total penetration resistance and/or the friction resistance, f_s, on a friction sleeve.

The Cone Penetration Test includes what has been variously called the Static Penetration Test, the Quasi Static Penetration Test and the Dutch Sounding Test.

Cone penetration tests are performed in order to obtain data on one or more of the following subjects:

1) the stratigraphy of the layers, and their homogeneity over the site
2) the depth to firm layers; the location of cavities, voids and other discontinuities
3) soil identification
4) mechanical soil characteristics
5) driveability and bearing capacity of piles

The cone penetrometers can be divided into three categories according to the system of measurement:

1) Electric Penetrometer, which uses electrical devices such as strain gauges and vibrating wires, built into the tip
2) Mechanical Penetrometer, which uses a set of inner rods to operate the penetrometer tip
3) Hydraulic and Pneumatic Penetrometer, which uses hydraulic or pneumatic devices built into the tip.

The above information, as well as other technical specifications, are presented in the international reference test procedure (De Beer *et al*, 1988) covering the Cone Penetration Test.

As the Hydraulic and Pneumatic Penetrometers are less common (Meigh, 1987), they are not included in the in-situ tests hierarchy. The Mechanical and Electric Penetrometers can generally be further divided into those for measurement of cone resistance only and those for measurement of both cone resistance and local side friction (Meigh, 1987). However, it has become common practice to use penetrometers with a friction sleeve, which are referred to as friction cones or friction penetrometers (De Ruiter, 1982). Hence, only friction penetrometers are considered.

The Piezocone Test (cone penetration test with pore-pressure measurement - CPTU) has evolved from the standard Electric Cone Test. It consists of a cone into which - or in the immediate vicinity of which - a porous filter has been inserted to measure, by means of a pore-pressure sensor, the pore-water pressure present at the interface between the penetrometer tip and the soil during penetration. This pore-water pressure includes the excess pore-water pressure (positive or negative) arising from the penetration of the cone and the push rods into the ground. In addition, the equilibrium piezometric profile can be determined during a stop in penetration (Manby and Wakeling, 1990; Robertson and Campanella, 1983b). Hence, the direct correlation between cone resistance, local side friction (when available) and pore pressure can be studied.

The international reference test procedure (De Beer *et al*, 1988) includes the Piezocone Test without standardizing any details such as the location and size of filter or the stiffness of the measuring system, as these areas are still under research. The Piezocone Test can be subdivided into those which provides friction measurements and to those with no friction sleeve available. Both of them are included in the test tree. The need for a friction sleeve though, when pore pressure data are available, was questioned by some of the members of the committee on penetration testing (De Beer *et al*, 1988).

The Piezocone Test opens the way for an effective stress analysis of the cone resistance and for an improved determination of soil parameters from CPT data (De Ruiter, 1982). However, it is in an early stage of development and its applications should be used with caution (Meigh, 1987).

Standard Penetration Test (SPT)

The Standard Penetration Test is the most widely used in-situ soil test worldwide. The test determines the resistance of soils in a borehole to the penetration of a tubular steel sampler, and obtains a disturbed sample for identification (Decourt *et al*, 1988). It is performed by dropping a hammer weighing 63.5 kg onto a drive head (screwed to the top of the drive rods) from a height of 760 mm (free fall). The number of blows, N, required to achieve a penetration of 300 mm, after its penetration under gravity and below a seating drive of 150 mm, is regarded as the penetration resistance, or N-value. Decourt *et al* (1988) presented an international reference test procedure.

The main purpose of the test is to obtain an indication of the consistency of sands and gravels in terms with relative density, D_r, of granular soils (BS 5930, 1981; Weltman and Head, 1983). This interpretation of the penetration resistance still suffers a lot of critisism (Lunne *et al*, 1990). It is also used to obtain an indication of the consistency of silts, clays and weak rocks in terms of undrained shear strength (BS 5930, 1981; Weltman and Head, 1983). The penetration value can be related to other soil characteristics in general use, such as angle of friction of granular soils and deformability. Engineering applications of N-values include determination of settlement of granular soils, bearing capacity of shallow and deep foundations, estimation of liquefaction potential and compaction control (Orchant *et al*, 1988; Robertson, 1985, 1986; Lunne *et al*, 1990).

Dynamic Probing Test (DP)

Dynamic Probing Test (or Dynamic Penetration Test) is probably the oldest penetration method for soil exploration in the field of foundation engineering (Broms and Flodin, 1988). The test consists of determining a driving resistance profile for a solid cone-shaped probe being driven into the soil by means of regular blows from a hammer of mass M, dropped freely through a constant distance H, on to an anvil at the top of the rods connected to the cone (Nixon, 1989). The number of blows required to drive the penetrometer a defined distance is regarded as the penetration resistance.

Four procedures are recommended by the ISSMFE Subcommittee (Stefanoff *et al*, 1988), classified according to the mass of the hammer used:

- *Dynamic Probing Light (DPL)*, corresponding to a hammer mass of 10 kg. The hammer should fall freely from a height of 0.5 m. The investigation depth usually is not larger than about 8 m and the number of blows should be recorded every 0.1 m (N_{10}).
- *Dynamic Probing Medium (DPM)*, corresponding to a hammer mass of 30 kg. The hammer should fall freely from a height of 0.5 m. The investigation depth usually is not larger than about 20 to 25 m and the number of blows should be recorded every 0.1 m (N_{10}).
- *Dynamic Probing Heavy (DPH)*, corresponding to a hammer mass of 50 kg. The hammer should fall freely from a height of 0.5 m. The investigation depth usually is not larger than about 25 m and the number of blows should be recorded every 0.1 m (N_{10}).
- *Dynamic Probing Superheavy (DPSH)*, corresponding to a hammer mass of 63.5 kg. The hammer should fall freely from a height of 0.75 m. The investigation depth can be larger than 25 m and the number of blows should be recorded every 0.2 m (N_{20}).

The results of dynamic probing testing (Stefanoff *et al*, 1988), can be used mainly qualitatively for general assessment of layering and types of subsoil and/or quantitatively to estimate engineering parameters of cohesionless and cohesive soils, such as relative density, shear strength and compressibilty. Some correlations also exist for the estimation of bearing capacity of deep and shallow foundations. Applications are generally restricted to estimating pile length and for compaction control. The Dynamic Probing Test is mainly used in cohesionless soils. Additional research is required in order to get better correlations with soil properties and other testing methods (Nixon, 1989; Scarff, 1989; Card *et al*, 1990)

Weight Sounding Test (WST)

The Weight Sounding Test originated in Sweden and became the most common penetration method in the Scandinavian countries (Bergdahl *et al*, 1988; Broms and Flodin, 1988). According to Meigh (1989)

it has never been used in UK. The weight penetrometer consists of a screw-shaped point, rods, weights and a handle. The penetrometer is used as a static penetrometer in soft soils when the penetration resistance is less than 1 kN and it is rotated when the resistance exceeds 1 kN. The point is penetrated into the ground by the application of weights added in stages to maintain a constant rate of penetration, and when it will not penetrate further under a weight of 1 kN, it is rotated. The number of half-turns every 0.2 m of penetration is recorded (N_{WST}). Due to the rotation of the screw-shaped point it can penetrate even stiff clays and dense sands.

The Weight Sounding Test is primarily used to obtain a continuous profile and an indication of the layer sequence, and of the lateral extent of different soil layers. The results can also be used to get an indication of the relative density and angle of friction of cohesionless soils, as well as the shear strength of cohesive soils. The degree of compaction can also be investigated. The bearing capacity of friction piles and spread footings in cohesionless soils and the settlement of spread footings and rafts can be determined as well. The above are discussed by Bergdahl *et al* (1988) and Broms and Flodin (1988). Comparisons of the weight sounding penetration resistance with other penetration resistances have also been carried out (Bergdahl *et al*, 1988; Bergdahl and Ottosson, 1988; Broms and Flodin, 1988; Pitts, 1990).

Static-Dynamic Penetration Test

The Static-Dynamic Penetration Test combines the Standard Penetration testing method and the Cone Penetration testing method (BS 5930, 1981, Weltman and Head, 1983). The equipment used is the Dutch Cone Penetrometer. The penetrometer is driven directly into the ground and the number of hammer blows is recorded for each 75 mm of penetration (dynamic part of the test). A static test is carried out at intervals of 300 mm. The test is used for non-cohesive soils, particularly those with thin coarse or dense layers.

SPECIAL PENETROMETER TESTS

The electric cone penetrometer permits the incorporation of a variety of sensors, of which the data can be recorded simultaneously with cone resistance and local side friction. A number of recent developments have been reviewed by De Ruiter (1982), Meigh (1987), Manby and Wakeling (1990) and Robertson(1986). A comprehensive report of these devices has been presented by Mitchell (1988). Jamiolkowski and Robertson (1989) provide relevant references for a number of them. Some of them are briefly discussed below.

Flat Plate Dilatometer Test (DMT)

The Flat Plate Dilatometer Test (or Marchetti Dilatometer Test) is considered to be a penetration tool that performs a lateral expansion test. It consists of a stainless steel blade containing, on one face, a thin flat circular expandable stainless steel membrane which is flush with the surrounding flat surface of the blade (Marchetti, 1980). The blade is pushed into the ground usually using a penetrometer rig. At 20 cm depth intervals, the membrane is inflated by gas pressure. According to Marchetti (1980) two measurements are taken at each test level: a) the pressure required to just begin to move the membrane (reading A) and b) the pressure required to move its centre 1 mm into the soil (reading B). Campanella *et al* (1985) suggested a modification to the test procedure, namely to record at each test level the closing pressure (C reading) at which the membrane recontacts the plane of the blade, in addition to the A and B readings. As this reading has only recently been introduced, its use has not been fully investigated yet although it is claimed that it could be used to estimate pore water pressures (Lutenegger, 1988).

These measurements are used to calculate three index parameters: material index, horizontal stress index and dilatometer modulus. Soil profiling and identification as well as soil parameters such as undrained shear strength of clays, friction angle of sands, density, overconsolidation ratio, lateral earth pressure coefficient and stiffness can be derived from empirical correlations with dilatometer's index parameters (Robertson, 1985; Jamiolkowski *et al*, 1985; Robertson, 1986; Orchant *et al*, 1988; Manby

and Wakeling, 1990; Lunne *et al*, 1990). Luttenberg (1988) describes the current state-of-practice of the test.

A dilatometer for offshore use has been developed at the Norwegian Geotechnical Institute that incoprorates a pore pressure element (Mitchell, 1988; Lunne *et al*, 1990). A similar device has been described by Campanella *et al* (1985).

Cone Pressuremeter Test

The Cone Pressuremeter is a penetration tool with a lateral expansion. The expansion tests are performed after stopping the penetration at selected intervals This type of instrument has significant future potential by combining the good logging capabilities of the CPTU and the good modulus measurements of the pressuremeter. A number of systems have been developed or are under development worldwide (Mitchell, 1988).

A device called Pressio-Penetrometer has been developed by Laboratoires des Ponts et Chaussées in Paris (Amar *et al*, 1982). It consists of three modules, a penetrometer cone, a piezometer and a pressuremeter cell, which are of 89 mm diameter. A friction sleeve can also be fitted.

Another device, which combines a piezometer, friction, bearing cone with a small size pressuremeter element is discussed by Campanella *et al* (1985) and Robertson (1985,1986). The pressure expansion test performed using the pressuremeter element is referred to as a Full- Displacement Pressuremeter Test since the cone produces a full-displacement installation technique (Hughes and Robertson, 1985).

Lateral Stress Cone Test (LSSCP)

The Lateral Stress Cone consists of an electronic cone, the friction sleeve of which is instrumented with a lateral stress sensing element to measure the normal stress acting on the sleeve (Robertson, 1986).

Research is in progress to determine the relationship between the measured value of horizontal stress and its initial value as a function of the initial relative density (Mitchell, 1988).

The development of the Lateral Stress Cone is discussed by Robertson (1986) and Mitchell (1988). Jamiolkowski and Robertson (1989) provide references to relevant published work.

Seismic Cone Test (SCPT)

The Seismic Cone Test provides an economic means of determining shear and compression wave velocities and hence permit the direct determination of dynamic shear modulus, G_{max}. Manby and Wakeling (1990) discuss some of the systems that have been developed and are now in use commercially.

Campanella *et al* (1985) and Robertson (1985, 1986) describe a system developed in North America. This device combines a piezometer, friction, bearing cone with a set of miniature seismometers built into the cone. The bearing, friction and pore pressure measurements are used to log the stratigraphy of a site during penetration and a downhole seismic technique is performed during pauses in the penetration to provide a profile of the in-situ shear wave velocity, V_S and hence the in-situ dynamic shear modulus, G_{max}.

Baldi *et al* (1988), describe a crosshole seismic piezocone penetration test in which the wave velocities between two penetrometers (one with a source and one with a receiver) are measured. Hepton (1988), reports downhole seismic testing using a seismic piezocone and a seismic flat dilatometer.

Vibratory Cone Test (CPTV)

The Vibratory Cone consists of a friction cone penetrometer equipped with an electrical vibrator and is intended as a quick way for evaluating the susceptibility of cohesionless deposits to liquefaction by defining a parameter D, which describes the relationship between the penetration resistance without

vibration and the penetration resistance with vibration (Mitchell, 1988). More research is required to establish quantitative correlations between D and liquefaction potential (Mitchell, 1988; Lunne *et al*, 1990).

Relevant references are given by Jamiolkowski and Robertson (1989).

Nuclear Density Probe Test (NCDT)

The Nuclear Density Probe consists of a cone penetrometer into which a nuclear source and detector are incorporated. The Nuclear Density Probe Test enables the measurement of bulk density of the penetrated geological materials using a gamma ray back scatter technique, with a radioactive source near the point of the probe and a detector mounted a short distance above it, separated by a radiation shield (De Ruiter, 1982; Meigh, 1987; Van Den Berg, 1987; Mitchell, 1988; Lunne *et al*, 1990; Manbu and Wakeling, 1990). Porosity and saturation can also be measured if both gamma and neutron rays are used (Nieuwenhuis and Smits, 1982; Mitchell, 1988; Sully and Echezuria, 1988).

Further details on nuclear density probes and results obtained by their use can be obtained from Ledoux *et al* (1982), Nieuwenhuis and Smits (1982) and Sully and Echezuria (1988).

Electrical Density Probe Test

The Electrical Density Probe Test (or Electrical Resistivity Probe Test) is used for the assessment of the porosity or density. Meigh (1987), Van Den Berg (1987), Mitchell (1988), Lunne *et al* (1990) and Manbu and Wakeling (1990) refer briefly to a two probe system for use in saturated sands. The first probe, the soil probe, consists of a cone penetrometer into which four electrodes are fitted above the friction sleeve. This device measures the electrical resistivity of the soil volume (soil plus water). The second probe, the water probe, contains a measuring cell which is sucked full of water at selected depths, and the resistivity of the water is determined. The readings are generally taken at 0.2 m

intervals of depth. The ratio of the porewater resistivity to that of the saturated soil is related to the porosity (and so to the in-situ density), by calibration tests performed in the laboratory.

Applications of the Electrical Density Probe Test are described by Kermabon *et al* (1969), Nelissen (1988). Woeller *et al* (1991) describe a similar system, which measures the bulk resistivity of the soil volume and the conductivity of the pore water for use in groundwater contaminant studies.

Electrical Conductivity Cone Test

The Electrical Conductivity Cone Test measures the electrical conductivity of the ground. Mitchell (1988) and Lunne *et al* (1990) refer to an electrical conductivity probe that consists of a standard electric friction cone with electrodes fitted into an insulating body behind the friction sleeve. The test can be used to detect salt water-fresh water boundaries and to locate contaminated groundwater (Mitchell, 1988; Manby and Wakeling, 1990).

Thermal Conductivity Cone Test

The Thermal Conductivity Cone Test enables the measurement of soil temperatures and change in temperature caused by the penetration process, by incorporating a temperature sensor (thermocouples or thermisters) in the penetrometer. It is then possible for the thermal conductivity of the ground to be computed from measurements of increase in temperature against time for a constant rate of heat input to the heating element (De Ruiter, 1982; Mitchell, 1988; Manby and Wakeling, 1990; Lunne *et al*, 1990).

Acoustic Cone Test (ACPT)

The Acoustic Cone Test is still at a development stage but seems to be a promising, supplementary in-situ testing method for site characterization. Results obtained so far suggest that the acoustic response can provide useful information of the soil type and profile conditions (De Ruiter, 1982; Meigh, 1987).

Tringale and Mitchell (1982) describe a friction cone penetrometer with a microphone located in the cone and an accompanying data acquisition system recently developed to receive, monitor, and record the acoustic response generated by soil particles interacting with the penetrometer as it moves through the soil.

Other acoustic cone penetrometer devices are discussed by Mitchell (1988).

PRESSUREMETER TESTS

The principle of pressuremeter testing is the expansion of a long cylindrical membrane installed in the ground in order to measure a relationship between pressure and deformation for the soil. According to the method of insertion, three categories of test can be distinguished (Mair and Wood, 1987):

a) Ménard- type Presuremeter Test (PMT)

b) Self-boring Pressuremeter Test (SBP)

c) Push-in Pressuremeter Test (PIP)

Comprehensive reviews of pressuremeters have been provided by Baguelin, Jézéquel and Shields (1978) and Mair and Wood (1987).

Ménard-type Pressuremeter Test (PMT)

The Ménard-type Pressuremeter Test consists of a long cylindrical probe covered with a rubber membrane and connected to a loading system and a measurement console. The device is lowered into a pre-formed hole and the test is performed by injecting fluid under pressure into the probe which causes expansion of the membrane into the soil. The volume injected as a function of the pressure applied is measured which enables the strength and, mainly, the deformation characteristcs of the ground to be investigated (Mair and Wood, 1987; Orchant *et al*, 1988).

However, it is believed that the Ménard-type Pressuremeter Test should not be considered as means of obtaining fundamental soil properties, but as a testing method whose results should be used in direct empirical design models for deep and shallow foundations (Baguelin *et al*, 1978).

Because of lack of standardization, several varieties of pressuremeter are in current use, but they all function on the same principle, as described above (Orchant *et al*, 1990; Mair and Wood, 1987; Robertson 1985, 1986).

Self-boring Pressuremeter Test (SBP)

The Self-boring Pressuremeter has been developed both in the UK (Camkometer) and France (PAFSOR) in order to overcome the problem of soil disturbance created by the insertion of the Ménard-type Pressuremeters in pre-formed holes (Mair and Wood, 1987). The Self-boring Pressuremeter consists of a part similar to the Ménard-type Pressuremeter and a small rotating boring tool incorporated at the tip of the apparatus. The soil cuttings from the rotating action are slurried out to the surface via a double string of rods. Load cells and transducers enable independent measurements of horizontal stress and strain, equilibrium pore pressure and excess pore pressure (Lunne *et al*, 1990).

The Self-boring Pressuremeter Test provides effective stress and deformation parameters when pore water stress measurements are made, as well as a reasonable estimate of the in-situ horizontal stress (Orchant *et al*, 1988). An estimate of the coefficient of horizontal consolidation can be obtained from pore pressure measurements (Mair and Wood, 1987). It should be noted though that there is limited experience of these interpretation methods (Mair and Wood, 1987).

Push-in Pressuremeter Test (PIP)

The Push-in Pressuremeter, in which the device is pushed into the ground below the base of a borehole, has been mainly developed for offshore use (Weltmann and Head, 1983; Mair and Wood, 1987; Lunne *et al*, 1990). The device consists of a pressuremeter head, a spacer, a pressure developer and a control

unit. The pressuremeter head consists of a hollow cylinder with an unrestricted passage through the instrument similar to a sampling tube, enabling the extruded soil to slide into the head and finally into the spacer, aiming to minimise the disturbance caused by the penetration action to the surrounding soils. At the end of each test, a disturbed sample can be recovered. The membrane is inflated with oil delivered under pressure by an electrical pump within the pressure developer and both volume increase and pressure applied are monitored continuously. Therefore, the strength and the deformation characteristics of the ground can be obtained. The test is not suitable for estimating the in-situ horizontal stress (Mair and Wood, 1987). Huang and Haefele (1988), present a similar push-in pressuremeter developed for on-shore use.

An alternative approach to the hollow push-in pressuremeter, discussed by Hughes and Robertson (1985), is a closed-ended push-in pressuremeter, called a Full-Displacement Pressuremeter. Data from self-boring and full-displacement pressuremeter tests in sand are also presented. Although this test could be considered as a separate pressuremeter test, in this study its principle is demonstrated in the Cone Pressuremeter Test described above, that incorporates a small diameter full-displacement pressuremeter with a cone penetrometer.

Comparisons are presented by Huang and Haefele (1988) between test data obtained by the push-in pressuremeter and a pre-bored, a self-boring, and a full displacement pressuremeter.

IN-SITU STRESS MEASUREMENT TESTS

Total Stress Cell Test (TSC)

The Total Stress Cell (or Earth Pressure Cell) consists of pushing into the ground a spade-shaped, thin cell and measuring the in-situ horizontal stress and stress changes. The principle of the test is that the disturbance created by the insertion of the cell is allowed to dissipate with time, and the stresses surrounding the pressure cell creep back to equilibrium (BS 5930, 1981; Ohya *et al*, 1983).

Massarch (1975), presented a hydraulically operated total stress cell which permitted succesful measurements of total lateral stress in soft clays. Tedd and Charles (1981), presented the application of the technique to stiff clays. Lack of experience exists with the push-in spade-like total stress cell in sands (Jamiolkowski *et al*, 1985). Lunne *et al* (1990), refer to a small total stress cell, that can be used to measure both vertical and horizontal stresses from a borehole.

Iowa Stepped Blade Test (ISB)

Handy *et al* (1982) presented the development and testing of the Iowa Stepped Blade Test (or K_o Stepped Blade Test). The concept of the test is that the disturbance caused by the insertion of any device into the ground is unavoidable and that it varies as a function of the thickness of the device. The test consists of pushing into the ground a series of pressure sensing membranes, each fixed to a blade of increasing thickness. The pressure recorded on each total stress cell when positioned at the depth of interest, is plotted versus the corresponding blade thickness. The plot is extrapolated to zero thickness to give an estimate of the total lateral stress for the undisturbed state. In its present form the device incorporates four total stress cells of different thicknesses (Mitchell, 1988; Lunne *et al*, 1990), instead of three as reported by Handy *et al* (1982).

This device is an extension of the spade-shaped Total Stress Cell instrument, with the difference that, according to Handy *et al*, (1982), it is not necessary in this case to wait for the equilibrium pressure to be established in order to evaluate the in-situ lateral earth pressure (Jamiolkowski *et al*, 1985).

Hydraulic Fracturing Test (HFT)

The principle of hydraulic fracturing is described by Bjerrum *et al*, (1972). The Hydraulic Fracturing Test (Lunne *et al*, 1990; Jamiolkowski *et al*, 1985; BS 5930, 1981) is usually performed by the use of piezometers in soils and consists of gradually increasing the water pressure in a piezometer monitoring the outflow rate for a few minutes at each pressure step, until a pressure is reached at which a large increase in the flow rate occurs. This means that a crack has formed in the soil around the piezometer

(perpendicular to the direction of the minor principal stress). When the pressure is reduced incrementally, the width of the crack gradually decreases. The pressure at which the crack just closes is assumed to correspond to the total in-situ horizontal stress. Hydraulic fracturing is also used in rocks.

Self-boring Ko meter Test

The Self-boring Ko meter is a device in which total pressure cells are installed in the sides of a square or hexagonal self-boring probe in order to enable measurements of the total horizontal earth pressure in two or three dimensions (Baguelin *et al*, 1978). A Self-boring Ko meter, known as a Self-boring Lateral Stress Cell, has been developed in UK. This device is also associated with the name Camkometer, as is the Self-boring Pressuremeter developed in UK (Mair and Wood, 1987).

SHEAR TESTS

Vane Test

The Vane Test (BS 5930, 1981; Weltman and Head, 1983; Van Den Berg, 1987; Orchant *et al*, 1988) consists of placing a four bladed vane in the undisturbed soil and rotating it from the surface. The torque required to cause a cylindrical surface to be sheared by the vane is measured. The vane is connected to the surface by steel torque rods. The test can be performed either at the bottom of a borehole or to a limited depth by direct penetration using purpose-designed equipment.

The measured torque can be related to the undrained shear strength of the soil. The test can be extended to measure the remoulded shear strength of the soil; hence, the Vane Test can also be used to investigate the sensitivity of clays (Orchant *et al*, 1988; Lunne *et al*, 1990). The test is suitable for very soft to stiff intact saturated cohesive soils (BS 1377, 1990). The interpretation of the results of the Vane Test is discussed by Wroth (1984) and Lunne *et al* (1990) among others.

Lunne *et al* (1990) compare the requirements of the American, British and Norwegian national standards for in-situ vane shear testing.

Self-boring Vane Test

In the Self-boring Vane Test (Baguelin *et al*, 1978), a length of the sides of the probe is fitted with blades and this cylindrical part rotates on command once the probe is in position. The number and height (projection) of the blades can be varied from one test to another.

Borehole Shear Test (BST)

The Borehole Shear Test is a test in which the shear resistance of the soil is determined in the borehole by pressing two ridged plates horizontally against the borehole sides under a controlled pressure (normal stress) and then pulling upwards on the shearing device at a constant rate until the maximum force is reached (which can be converted to a maximum shear stress). The test is repeated at the same location by increasing the pressure on the plates and again by pulling on the shearing device and hence measuring the coresponding shear stress. Shear strength parameters c and ϕ are determined by plotting shear stress versus normal stress and drawing the Mohr-Coulomb failure envelope (Lamrechte and Rixner, 1981).

The current BST equipment is lightweight and portable, requires no external power to operate it, and a complete test can usually be accomplished in about an hour (Lutenegger and Hallberg, 1981).

In-situ Shear Test

The principle of this In-situ Shear Test (BS 5930, 1981; Weltman and Head, 1983) is similar to that of the laboratory shear box test. A reinforced open box is moved laterally by a jacking system while a normal stress is applied to the top by jacking from a fixed point, subjecting a sample of ground to direct shear. The test is generally designed to measure the peak shear strength of the in-situ material as a function of the stress normal to the sheared plane. Rates of shear vary according to whether total or

effective parameters are required. More than one test is generally required to obtain a realistic design value. Indication of the residual shear strength may be obtained by reversal and/or re-shear.

A detailed study of the results of undrained direct simple shear tests is presented by Wroth (1984). Marsland (1990) examines the determination of effective strength parameters of stiff fissured clays using large in-situ shear boxes.

BEARING TESTS

Plate Loading Tests (PLT)

The Plate Loading Tests (BS 5930, 1981; Weltman and Head, 1983; Robertson, 1985, 1986; Van Den Berg, 1987) involve measuring the penetration of a rigid plate into a soil or weak rock caused by an applied load. The plate is usually loaded through a column formed by a steel tube; the load is applied to the column by means of a hydraulic jack operating against the resistance of kentledge, tension piles or ground anchors. The penetration of the plate under load is generally transmitted to dial gauges at the surface by means of a settlement measurement rod that is located within the steel tube by which the load is applied. The test can be carried out in shallow pits or trenches or at depth in the bottom of a borehole, pit or adit. The diameter of the plate can vary according to the depth at which the test is performed, the dimensions of the load in the real structure and on the grain size of the material to be tested (Van Den Berg, 1987).

The test is used to determine the deformation characteristics of the material beneath the loading plate, as well as the shear strength characteristics if the test is continued to failure. The test is usually carried out either under a series of maintained loads (allowing consolidation before a further load increment is applied) or at a constant rate of penetration depending on whether the drained or undrained strength and deformation characteristics are required (BS 5930, 1981). To determine the variation of ground properties with depth, it will generally be necessary to carry out a series of plate tests at different

depths. The interpretation of Plate Loading Tests in order to obtain deformation parameters is discussed by Jamiolkowski *et al* (1985).

Screw Plate Test (SPLT)

The Screw Plate Test (or Field Compressometer Test) (Weltman and Head, 1983; Robertson 1985,1986; Orchant *et al*, 1990; Massarsch, 1986), is a recent variation of the conventional Plate Loading Test. The test consists of the measurement of the load versus settlement and settlement versus time behaviour of a helical plate screwed into the natural soil with a minimum of disturbance at any desired depth in conjuction with a prebored hole (Mitchell and Kay, 1985). The plate is loaded in a similar manner to the Plate Loading Test. The test can be performed with either load or displacement control. The Screw Plate Test is used when it is required to perform tests at depth, since it is faster and less expensive than Plate Loading Tests (Jamiolkowski *et al*, 1985).

The Screw Plate Test has been utilised for the measurement of the in-situ deformability characteristics for both cohesive and cohesionless soils and the undrained shear strength of cohesive soils. Parameters for drained conditions can also be obtained (Kay and Parry, 1982). Test procedures and interpretation of the results have been described by Selvadurai *et al* (1980), Kay and Parry (1982), Kay and Avalle (1982) and Selvadurai (1986) amongst others. Jamiolkowski *et al* (1985) discuss the interpretation of the Screw Plate Test for the determination of deformation parameters.

Self-boring Plate Test

Mori (1983) presents a self-boring instrument used for borehole loading tests that minimises the disturbance of soil caused by installation. The instrument consists of a cylinder closed with a plate at its lower end. A couple of blades scrape the soil beneath the loading plate when the cylinder is rotated and provide a clean and smooth surface. Cuttings are forced into the space above the loading plate through an opening between the plate and the blade. The plates are retracted into the loading plate when the plate reaches the desired depth.

The Self-Boring Plate Test is mainly applied to obtain design parameters for deep foundations resting on dense sandy soils or stiff cohesive soils. Mori (1983) presents results of the test for dense sandy soils.

Pressurized Chamber Test

The Pressurized Chamber Test (BS 5930, 1981), is carried out in an underground excavation or length of tunnel and consists in charging a chamber with water under various pressures in order to obtain the deformation moduli of the surrounding soil. The test is usually used in projects involving tunnels carrying water under pressure. It is necessary to know the drainage conditions which apply during the test in order to know whether the modulus obtained is drained, partially drained or undrained.

In-situ California Bearing Ratio Test (CBR)

The In-situ California Bearing Ratio Test (BS 5930, 1981; Weltman and Head, 1983; Van Den Berg, 1987) consists of pushing a cylindrical plunger into the soil at a given rate and comparing the relationship between force and penetration into the soil to that of a standard material in order to obtain the California Bearing Ratio (CBR) value of the penetrated soil. The test is an empirical method in which design curves are used to estimate road pavement thickness appropriate to the CBR of the soil.

IN-SITU DENSITY TESTS

Most of the available methods depend on the removal of a representative sample of soil from the site and then determining its mass and the volume it occupied before being removed. The tests based on this principle that are briefly examined below, are the *Sand Replacement Tests*, the *Core Cutter Test*, the *Weight in Water Test*, the *Water Replacement Test*, and the *Rubber Ballon Test*. The variations between these methods lie in different procedures used for measuring the volume, according to the nature of the soil being tested. In addition, *Nuclear Tests* are described that use gamma rays for the determination of the in-situ density of soil.

The tests determine bulk density. All methods are suitable for shallow depth investigations. Nuclear probes have now been developed which can be lowered down boreholes for deeper investigations.

Sand Replacement Tests

In the Sand Replacement Tests, dried graded sand is poured into the void from which the soil sample is taken to determine its volume (Weltman and Head, 1983). BS 5930 (1981) refer to three variations on the sand replacement method:

- Small Pouring Cylinder Test
- Large Pouring Cylinder Test
- Scoop Test

The first, employing a small pouring cylinder, is used for fine and medium grained soils. The second, using a large pouring cylinder, is suitable for fine, medium and coarse grained soils. Both methods are described in BS 1377 (1990). The third may be used for fine, medium, coarse grained soils but it is essentially cruder than the first two and yields less reliable results; hence its use should be restricted to situations where no pouring cylinder is available (BS 5930, 1981). This method is not included in the revised BS 1377 (1990).

Core Cutter Test

In the Core Cutter Test a cylindrical cutter is driven into the soil and the known internal volume of the cylinder is completely filled. The method, described in BS 1377 (1990) is restricted to cohesive soils where a core may be cut and the sample does not fall out.

Water Replacement Test

In the Water Replacement Test the density of natural or compacted coarse-grained soils is measured by using a circular density ring on the ground surface and a flexible plastic sheet to retain water to determine the volume of an excavated hole (BS 1377, 1990).

Weight in Water Test

The Weight in Water Test is applicable to any soil where representative samples occur in discrete lumps which will not disintegrade during handling and submersion in water (BS 5930, 1981). This method is not included in the revised BS 1377 (1990).

Rubber Balloon Test

The Rubber Balloon Test is a water replacement method with an inflated rubber membrane retaining the liquid required to measure the volume of the test hole (BS 5930, 1981). The method is described in ASTM D 2167 (1966).

Nuclear Tests

The Nuclear Tests (BS 1377, 1990; ASTM D2922, 1971) determine the density of soils through the use of a nuclear gauge by the attenuation of gamma rays, where the gamma source or gamma detector (or both) are placed at or near the surface. The rate at which the gamma rays arrive from the gamma source through the material being tested to the gamma detector is determined. The relationship between the nuclear-count rate and material density is determined by correlation tests of materials of known average densities.

Three methods are examined, depending on the test geometry used:

- Backscatter Test
- Direct Transmission Test
- Air Gap Test

In the Backscatter Test both the source and the detector are placed on the material under test. Some gauges include a nuclear moisture measuring system allowing the determination of in-situ dry density and moisture content. The method is described in the British Standard (BS 1377, 1990), the American Standard (ASTM D 2922, 1971) and the Australian Standard (AS 1289.E8.2, 1984).

The Direct Transmission Test, which is also described in the British Standard, the American Standard and the Australian Standard (AS 1289.E8.1, 1984), requires that either the gamma source or the detector shall be housed in a probe for inserting in the material to be tested. Facilitiy for the determination of dry density and moisture content could also be provided by a gauge operating in the direct transmission mode (BS 1377, 1990; AS 1289.E8.1, 1984).

In the Air Gap Test the gauge will be supported by cradle or spacers at the optimum air gap, so both the gamma source and the detector are at optimum height above the material being tested. This method, described only in the American Standard (ASTM D 2922, 1971), requires taking one or more readings in the backscatter position and the air gap position.

PERMEABILITY TESTS

Borehole Tests

The determination of in-situ permeability by tests in boreholes involves the application of a hydraulic pressure head difference between water in the borehole and that in the ground to measure the resulting flow. For more accurate measurements a piezometer is installed by surrounding it with a granular filter to prevent erosion of the ground. According to whether the pressure in the borehole is kept constant or not it is possible to distinguish the following types of test:

- Variable Head Test
- Constant Head Test.

The Variable Head Test (BS 5930, 1981; Weltman and Head, 1983; Van Den Berg, 1987) can be further subdivided into:

- Rising Head Test
- Falling Head Test.

In the Rising Head Test (or Outflow Test), the pressure in the borehole may be decreased by pumping water out of it, whereas in the Falling Head Test (or Inflow Test), the pressure in the borehole may be increased by introducing water into it. The head in the borehole is then allowed to equalise with that in the ground, the actual head being measured at intervals of time from the beginning of the test. These tests are suitable in medium and coarse grained soils.

The Constant Head Test (BS 5930, 1981; Weltman and Head, 1983; Van Den Berg, 1987) is usually conducted as an inflow test in which the rate of flow of water into the ground is adjusted until a constant head is achieved. The rate of flow required to maintain the constant water level is measured. In compressible soils such as silt or clay, a piezometer is usually installed. The Constant Head Tests are likely to give more accurate results than Variable Head Tests but they are more complicated to perform. They are used when the rise or fall of water is too rapid for accurate timing (Weltman and Head, 1983).

Self-boring Permeameter Test

In this test the self-boring technique is used for the installation of the piezometer with minimal disturbance of the ground allowing at the same time the perfomance of the test without delay. The Self-boring Permeameter Test (Bageulin *et al*, 1974; Jézéquel and Mieussens, 1975) consists of the self-boring part, the filtering part and the ward cells. The filtering part consists of a porous cylinder placed in a direct line behind the cutting edge. The ward cells, placed on either side of the filtering part, consist of rubber membranes that dilate under pressure of water or gas and serve two purposes: i) to hold in place the de-aerating cylinder around the piezometer, until the unit arrives at the water table and ii) during the permeability test, to prevent leakage of water between the permeameter and the soil.

A constant head test is performed and the coefficients of permeability and consolidation are measured. The horizontal earth pressure coefficient, and therefore K_0 can also be measured using hydraulic fracturing (Bageulin *et al*, 1978; Jézéquel and Mieussens, 1975).

Pumping Tests

A large scale Pumping Test (BS 5930, 1981; Weltman and Head, 1983) is the best, but most expensive method, presently available to estimate the permeability in a relative pervious deposit ($k>10^{-4}$ cm/sec) (Jamiolkowski *et al*, 1985). In principle, a pumping test consists of pumping at a known constant rate from a well and observing the drawdown effect on ground water levels at some distance away from the pumped well. The test procedure is to bore a pumping well to the full depth of the aquifer to be tested and install two lines of observation wells (four in minimum) perpendicular to each other and radially in plan from it. The analysis of the results is discussed in detail in the British Standards (BS 5930, 1981).

GEOPHYSICAL SURVEYING TESTS

The Geophysical Surveying Tests are based on determining variations in a physical property of rock or soil, such as velocity of shock waves (seismic methods), electrical conductivity (resistivity method), variations in density (gravimetric method) or magnetic susceptibility (magnetic method) (BS 5930, 1981). When conducting a geophysical survey, subsurface conditions are examined indirectly by interpreting the contrast in physical properties between different materials and their relationship with engineering parameters. These methods are complementary to direct methods of subsurface exploration.

Seismic Methods

The Seismic Methods involve the sudden release of energy by the use of an explosive charge in the ground or from impacting or vibrating the ground in order to generate seismic shock waves to propagate through the soil and the measurement of the velocities of the waves. These methods rely on the differences in the velocity of the generated waves through different geological or man-made materials. In general two types of waves are generated by a seismic disturbance, body waves (compressional and shear) and surface waves (Rayleigh and Love). The main seismic methods are discussed briefly below.

- **Seismic Refraction Test**

The Seismic Refraction Test (Clayton *et al*, 1982) is one of the most frequently used geophysical techniques which consists of producing seismic body waves, either from a small explosive charge or from a mechanical source and accurately measuring the time required for them to travel from a source to vibration detectors (geophones) at varying known distances away (BS 5930, 1981). The technique is suitable for investigating shallow depths (Weltman and Head, 1983).

Seismic velocities have been correlated to material type (Orchant *et al*, 1988; Bell *et al*, 1990) and have been used to determine the dynamic shear modulus (Woods, 1978). The greatest use of this technique is in the determination of rockhead level (BS 5930, 1981). Applications of the Seismic Refraction Test are discussed by Lee and De Freitas (1990), and McDowell (1990) among others.

- **Seismic Reflection Test**

The Seismic Reflection Test (Clayton *et al*, 1982) involves the generation of seismic body waves at or near the surface and the reception of the energy reflected back to the geophones from acoustic impedance contrasts at depth. The acoustic impedance is the product of seismic velocity and density (Orchant *et al*, 1988). The Seismic Reflection Test, only recently used for land-based investigations to shallow depth, is mainly used for accurate profiling of geological structures (Clayton *et al*, 1982).

- **Seismic Cross-Hole Test (SCS)**

The Seismic Cross-Hole Test (Woods, 1978; Clayton, 1982) consists of generating a source of seismic energy in or at the bottom of one borehole and measuring the time required for that energy (body waves) to travel to the detector placed in another borehole by the most direct route.

From the borehole spacing and travel time the velocity of the seismic wave is computed, and it is then used to compute the shear modulus. The technique is considered by many engineers to be the most

reliable field method for obtaining the shear modulus. Anderson *et al*, (1978), Arango *et al*, (1978), McDowell (1990), Pinches and Thompson (1990) discuss results obtained from cross-hole tests.

- **Seismic Down-Hole Test (SDS)**

The Seismic Down-Hole Test involves lowering one or more geophones into a borehole and clamping them at preselected depths in predetermined orientations. An impulse is generated at the surface of the ground near the top of the borehole and the times required for the body waves to travel between the surface and down-hole receivers is measured (Woods, 1978). Results obtained from Down-Hole Tests are discussed by Arango *et al*, (1978), McDowell (1990) and, Pinches and Thompson (1990).

- **Surface Wave Test**

The Surface Wave Test employ Rayleigh and Love waves (surface waves) for the determination of shear modulus of near surface soils (Woods, 1978; Lunne *et al*, 1990). Using an electro-magnetic or some other harmonic vibrator, a steady state R-wave can be generated and the output of a geophone moved along the surface on a radius from the vibrator is compared to a reference or input signal and in-phase points are identified. A plot of distance from source versus number of waves can be used to determine the average wavelength for the R-wave from which the shear wave velocity can be calculated. It has been shown that steady-state Love waves can be used to determine shear wave velocities for a soil profile with a low velocity layer on top of a high velocity layer; Woods (1978) comments that he knows of no large scale applications of this technique for engineering purposes.

Resistivity Test

The Resistivity Test (or Electrical Resistivity Test) (Clayton *et al*, 1982; Weltman and Head, 1983; BS 5930, 1981; Orchant *et al*, 1988), used for investigating simpler geological problems, rely on measuring subsurface variations of electrical current flow revealed by transmitting direct or alternating current into the subsurface by two electrodes (current electrodes). Another pair of electrodes (potential electrodes) measures the voltage in the soil generated by this current flow.

The Electrical Resistivity Test is commonly used to map lateral and vertical changes in geological or man-made materials. Lateral changes in resistivity are detected by using a fixed electrode spacing (appropriate to the depth of interest) and moving the whole electrode array along a traverse between each resistivity measurement (Profiling method). Vertical changes are measured by progressively moving the electrodes outwards with respect to a fixed central point, increasing each time the depth of penetration (Electrical Sounding method).

The method may also be used to determine the depth to the water table and to identify buried features. The analysis of the results is done by curve matching using standard curves for various soil layer configurations. Results obtained from the Electrical Resistivity Test are discussed by Frost and Dumble (1986), Barker *et al*, (1990) amongst others.

Gravimetric Test

The Gravimetric Test (Clayton *et al*, 1982; BS 5930, 1981) involves measuring lateral changes in the earth's gravitational field. Such variations are associated with near surface changes in density; therefore they may be related to changes in soil or rock type. In ground investigation, gravity methods are limited to locating large faults and the extent of large buried channels.

Magnetic Test

The Magnetic Test (Clayton, 1982, BS 5930, 1981) is based on the measurement of local variations in the earth's magnetic field. Such variations are associated with differences in magnetic susceptibility (the degree to which a body is magnetised) of rocks and soils or the presence of magnetised bodies. Magnetic techniques are used to locate localised subsurface features of engineering interest such as abandoned mine shafts, sink holes and buried services.

References

Amar S., Baguelin F. and Jézéquel J.F. (1982), *Pressio-Penetrometer for Geotechnical Surveys on Land and Offshore*, in Proceedings of the 2nd European Symposium on Penetration Testing, ESOPT II, Amsterdam, May, Balkema A.A., Rotterdam, vol. 2, pp 419-423.

Anderson D.G., Espana C. and McLamore V.R. (1978), *Estimating In Situ Shear Moduli at Competent Sites*, Earthquake Engineering and Soil Dynamics, Specialty Conference, Pasadena, CA, June, pp 181-197.

Arango I., Moriwaki Y. and Brown F. (1978), *In-situ and Laboratory Shear Velocity and Modulus*, Earthquake Engineering and Soil Dynamics, Specialty Conference, Pasadena, CA, June, pp 198-212.

AS 1289.E8.1 (1984), *Determination of Field Moisture Content and Field Dry Density of a Soil - Method Using a Nuclear Surface Moisture-Density Gauge - Direct Transmission Mode*, Methods of Testing soils for Engineering Purposes, Part E - Soil Compaction and Density Tests, Australian Standard, pp E8.1-1-E8.1-2.

AS 1289.E8.2 (1984), *Determination of Field Moisture Content and Field Dry Density of a Soil - Method Using a Nuclear Surface Moisture-Density Gauge - Backscatter Mode*, Methods of Testing soils for Engineering Purposes, Part E - Soil Compaction and Density Tests, Australian Standard, pp E8.2-1-E8.2-2.

ASTM D2167-66 (1966), *Density of Soil in Place by the Rubber-Balloon Method*, American Society for Testing Materials, pp 267-270.

ASTM D2922-71 (1971), *Density of Soil and Soil-Aggregate in Place by Nuclear Methods (Shallow Depth)*, American Society for Testing Materials, pp 357-364.

Bageulin F., Jézéquel J.F. and Shields D.H. (1978), *The Pressuremeter and Foundation Engineering*, Series on Rock and Soil Mechanics, vol. 2, (1974/77), n. 4, Trans Tech Publications, 617 p.

Bageulin F., and Jézéquel J.F.and Le Méhauté A. (1974), *Le Perméamètre Autoforeur*, Canadian Geotechnical Journal, vol.11, pp 624-628.

Baldi B., Bruzzi D., Superbo S., Battaglio M. and Jamiolkowski M. (1988), *Seismic Cone in Po River Sand*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, De Ruiter J. (ed.), Balkema A.A., Rotterdam, March, vol. 2, pp 643-650.

Barker R.D., Lerner D.N. and Rodrigeuz-Estrada H.V. (1990), *Resistivity Sounding for a Landfill Investigation at Bray, Berkshire*, Field Testing in Engineering Geology, (eds. Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R.), Geological Society Engineering Geology Special Publication No 6, pp 287-294.

Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R. (eds) (1990), *Field Testing in Engineering Geology*, Session 2: Pressuremeter Testing in Soils, Geological Society Engineering Geology Special Publication No 6, pp 23-63.

Bergdahl U. and Ottosson E. (1988), *Soil Characteristics from Penetration Test Results: A Comparison Between Various Investigation Methods in Non-Cohesive Soils*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 399-405.

Bjerrum L., Nash J.K.T.L.,Kennard R.M. and Gibson R.E. (1972), *Hydraulic Fracturing in Field Permeability Testing*, Géotechnique, vol. 22, n. 2, pp 319-332.

British Standard 1377 (1990), *In-situ Tests*, British Standard Methods of Test for Soils for Civil Engineering Purposes, Part 9, British Standards Institution, London.

British Standard 5930 (1981), *Code of Practice for Site Investigations*, British Standards Institution, London.

Broms B.B. and Flodin N. (1988), *History of Soil Penetration Testing*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 157-220.

Campanella R.G., Robertson P.K., Gillespie D.G. and Grieg J. (1985), *Recent Developments in In-situ Testing of Soils*, in Proceedings of the 11th International Conference on Soil Mechanics and Foundation Engineering (ISSMFE), San Francisco, Balkema A.A., Rotterdam, vol.2, pp 849-854.

Card G.B., Roche D.P. and Herbert S.M. (1990), *Applications of Continuous Dynamic Probing in Ground Investigation*, (eds. Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R.), Geological Society Engineering Geology Special Publication No 6, pp 129-135.

Clayton C.R.I., Simons N.E. and Matthews M.C. (1982), *Site Investigation*, Granada Publishing Ltd, p 424.

De Beer E.E., Goelen E., Heynen W.J. and Joustra K. (1988), *Cone Penetration Test (CPT): International Reference Test Procedure*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 27-51.

De Ruiter J. (1982), *The Static Cone Penetration Test*, State of the Art Report, in Proceedings of the 2nd European Symposium on Penetration Testing, ESOPT II, Amsterdam, May, Balkema A.A., Rotterdam, pp 389-405.

Decourt L., Muromachi T., Nixon I.K., Schmertmann J.H., Thorburn S. and Zolkov E. (1988), *Standard Penetration Test (SPT): International Reference Test Procedure*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, March, vol. 1, pp 3-26.

Frost F.B. and Dumble J.P. (1986), *The Application of Ground Conductivity and Offset Wenner Resistivity Soundings to Optimise the Investigation of a 300ha Site in the West Midlands*, Site Investigation Practice: Assessing BS 5930, Geological Society, Engineering Geological Special Publication No. 2, (ed. Hawkins A. B.), pp 241-246.

Handy R. L., Remmes B., Moldt S., Lutteneger A.J. and Trott G. (1983), *In-situ Stress Determination by Iowa Stepped Blade*, Journal of Geotechnical Division, ASCE, vol. 109, GT11, pp 1405-1422.

Hepton P. (1989), *Shear Wave Velocity Measurements During Penetration Testing*, Proceedings of the Geotechnology Conference on Penetration Testing in the UK, Birmingham, Institution of Civil Engineers, Thomas Telford, London, pp 275-278.

Huang A. -B. and Haefele K.C. (1988), *A Push-In Pressuremeter/Sampler*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, March, vol. 1, pp 533-538.

Hughes J.M.O. and Robertson P.K. (1985), *Full-Displacement Pressuremeter testing in Sand*, Canadian Geotechnical Journal, vol. 22, n. 3, pp 298-307.

ISSMFE, (1977), International Society for Soil Mechanics and Foundation Engineering, *Report of the Subcommittee on Standardization of Penetration Testing in Europe*, in Proceedings of 9th International Conference on Soil Mechanics and Foundation Engineering, Tokyo, vol. 3, appendix 5, pp 95-152.

ISSMFE, (1988), Technical Committee on Penetration Testing, *International Reference Test Procedures*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 3-90.

Jamiolkowski M., Ladd C.C., Germaine J.T. and Lancellotta R. (1985), *New Developments in Field and Laboratory Testing of Soils*, Theme Lecture, in Proceedings of the 11th International Conference on Soil Mechanics and Foundation Engineering (ISSMFE), San Francisco, Balkema A.A., Rotterdam, vol.1, pp 57-153.

Jamiolkowski M. and Robertson P.K. (1989), *Future Trends for Penetration Testing*, Closing Address, Proceedings of the Geotechnology Conference on Penetration Testing in the UK, Birmingham, Institution of Civil Engineers, Thomas Telford, London, pp 321-342.

Jézéquel J.F.and Mieussens C. (1975), *In Situ Measurement of Coefficients of Permeability and Consolidation in Fine Soils*, Proceedings of the Conference on In Situ Measurement of Soil Properties, American Society of Civil Engineers, Raleigh, June, vol.1, pp 208-224.

Kay J.N. and Avalle D.L. (1982), *Application of Screw Plate to Stiff Clays*, Journal of Geotechnical Engineering Division, January, ASCE, vol. 108. n. GT1, pp 145-154.

Kay J.N. and Parry R.H.G. (1982), *Screw Plate Tests in a Stiff Clay*, Ground Engineering, September, pp 22-30.

Kermabon A., Gehin C. and Blavier P. (1969), *A Deep-Sea Electrical Resistivity Probe for Measuring Porosity and Density of Unconsolidated Sediments*, Geophysics, vol. 34, n. 4, August, pp 554-571.

Lambrechte J.R. and Rixner J.J. (1981), *Comparison of Shear Strength Values Derived from Laboratory Triaxial, Borehole Shear and Cone Penetration Tests*, Laboratory Shear Strength of Soil, ASTM STP 740, (eds. Yong R.N and Townsend F.C.), American Society for Testing and Materials, pp 551-565.

Ledoux J.L., Menard J. and Soulard P. (1982), *The Penetro-Gammadensimeter*, in Proceedings of the 2nd European Symposium on Penetration Testing, ESOPT II, Amsterdam, May, Balkema A.A., Rotterdam, pp 679-682.

Lee S.G. and De Freitas M.H. (1990), *Seismic Refraction Surveys for Predicting the Intensity and Depth of Weathering and Fracturing in Granitic Masses*, Field Testing in Engineering Geology, (eds. Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R.), Geological Society Engineering Geology Special Publication No 6, pp 240-256.

Lunne T., Lacasse S., Rad N.S. and Décourt L. (1990), *SPT, CPT, Pressuremeter Testing and Recent Developments on In Situ Testing of Soils*, Norwegian Geotechnical Institute, General Report, Publication NR. 179, Oslo.

Lutenegger A. J. (1988), *Current Status of the Marchetti Dilatometer Test*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, March, vol. 1, pp 137-156.

Lutenegger A.J. and Hallberg G.R. (1981), *Borehole Shear Test in Geotechnical Investigations*, Laboratory Shear Strength of Soil, ASTM STP 740, (eds. Yong R.N and Townsend F.C.), American Society for Testing and Materials, pp 566-578.

Mair R.J. and Wood D.M. (1987), *Pressuremeter Testing: Methods and Interpretation*, CIRIA Ground Engineering Report: In-situ Testing, Butterworths, London, 160 p.

Manby C.N.D. and Wakeling T.R.M. (1990), *Developments in Soft-Ground Drilling, Sampling and In-situ Testing*, Trans. Institution of Mining and Metallurgy, Section A: Min. Industry, vol. 99, May-August, pp A91-A97.

Marchetti S. (1980), *In Situ Tests by Flat Dilatometer*, Journal of the Geotechnical Engineering Division, ASCE, vol. 106, n. GT3, March, pp 299-320.

Marsland A. (1990), *Measurements of Effective Strength Parameters of Stiff Fissured Clays using Large In Situ Shear Boxes*, (eds. Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R.), Geological Society Engineering Geology Special Publication No 6, pp 217-228.

Massarsch K.R. (1975), *New Method for Measurement of Lateral Earth Pressure in Cohesive Soils*, Canadian Geotechnical Journal, vol. 12, n. 2, pp 142-146.

Massarsch K.R. (1986), *Field Exploration and Instrumentation Methods*, Recent Developments in Laboratory and Field Tests and Analysis of Geotechnical Problems, (eds. Balasubramaniam A.S., Chandra S. and Bergado D.T.), A.A. Balkema, Boston, pp 211-222.

McDowell P.W. (1990), *The Determination of the Dynamic Elastic Moduli of Rock Masses by Geophysical Methods*, Field Testing in Engineering Geology, (eds. Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R.), Geological Society Engineering Geology Special Publication No 6, pp 267-274.

Meigh A.C. (1987), *Cone Penetration Testing: methods and interpretation*, CIRIA Ground Engineering Report: In-situ Testing, Butterworths, London, 141 p.

Meigh A.C. (1989), *Keynote Address*, Proceedings of the Geotechnology Conference on Penetration Testing in the UK, Birmingham, Institution of Civil Engineers, London, pp 1-8.

Mitchell J.K. (1988), *New developments in penetration tests and equipment*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 245-262.

Mitchell P.W. and Kay J.N. (1985), *Screw Plate and Cone Penetrometer as a Field Testing System*, in Proceedings of the 11th International Conference on Soil Mechanics and Foundation Engineering (ISSMFE), San Francisco, Balkema A.A., Rotterdam, vol.2, pp 913-915.

Mori H. (1983), *In-situ Plate Loading Test for Dense Sandy Soils Using a Self-Boring Instrument*, Symposium International, Soil and Rock Investigations by In-situ Testing, Paris, vol. 2, pp 353-357.

Nelissen H.A.M. (1988), *The Applications of the CPT and the Electrical Density Probe During the Construction of the Eastern Scheldt Storm Surge Barrier*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, March, vol. 2, pp 881-885.

Nieuwenhuis J.K. and Smits F.P. (1982), *The Development of a Nuclear Density Probe in a Cone Penetrometer*, in Proceedings of the 2nd European Symposium on Penetration Testing, ESOPT II, Amsterdam, May, Balkema A.A., Rotterdam, pp 745-749.

Nixon, I.K. (1989), *Review Paper on Dynamic Probing*, Proceedings of the Geotechnology Conference on Penetration Testing in the UK, Birmingham, Institution of Civil Engineers, London, pp 105-111.

Ohya S., Imai T. and Nagura M. (1986), *Recent Developments in Pressuremeter Testing, In-situ Stress Measurement and S-Wave Velocity Measurement*, Recent Developments in Laboratory and Field Tests and Analysis of Geotechnical Problems, (eds. Balasubramaniam A.S., Chandra S. and Bergado D.T.), A.A. Balkema, Boston, pp 189-210.

Orchant C.J., Kulhawy F.H. and Trautmann C.H. (1988), *Reliability-Based Foundation Design for Transmission Line Structures: Critical Evaluation of In-situ Test Methods*, Report EL-5507, vol. 2, Electric Power Research Institute, Palo Alto, 214 p.

Pinches G.M. and Thompson R.P. (1990), *Crosshole and Downhole Seismic Surveys in the UK Trias and Lias*, Field Testing in Engineering Geology, (eds. Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R.), Geological Society Engineering Geology Special Publication No 6, pp 299-308.

Pitts J. (1990), *The Use of Swedish Ram Sounding and Weight Sounding in Residual Soils and Weathered Rocks*, Field Testing in Engineering Geology, (eds. Bell F.G., Cripps J.C., Culshaw M.G. and Coffey J.R.), Geological Society Engineering Geology Special Publication No 6, pp 161-171.

Robertson P.K. (1985), *In Situ Testing and its Application to Foundation Engineering*, Soil Mechanics Series No. 91, University of British Columbia, Department of Civil Engineering, Vancouver, B.C., 212 p.

Robertson P.K. (1986), *In Situ Testing and its Application to Foundation Engineering*, Canadian Geotechnical Journal, vol. 23, pp 573-594.

Scarff R.D. (1989), *Factors Governing the Use of Continuous Dynamic Probing in UK Ground Investigation*, Proceedings of the Geotechnology Conference on Penetration Testing in the UK, Birmingham, Institution of Civil Engineers, London, pp 129-132.

Selvadurai A.P.S., Bauer G.E. and Nicholas T.J.(1980), *Screw Plate Testing of a Soft Clay*, Canadian Geotechnical Journal, November, vol. 17, n. 4, pp 465-472.

Selvadurai A.P.S. (1986), *The Use of Auger Type-Devices for the In-situ Testing of Soft Sensitive Clays*, Recent Developments in Laboratory and Field Tests and Analysis of Geotechnical Problems, (eds. Balasubramaniam A.S., Chandra S. and Bergado D.T.), A.A. Balkema, Boston, pp 223-230.

Stefanoff G., Sanglerat G., Bergdahl U. and Melzer K.J. (1988), *Dynamic Probing (DP): International Reference Test Procedure*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 53-70.

Stefanoff G., Sanglerat G., Bergdahl U. and Melzer K.J. (1988), *Weight Sounding Test (WPT): International Reference Test Procedure*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, vol. 1, pp 71-90.

Sully J.P. and Echezuria H.J. (1988), *In Situ Density Measurement with Nuclear Cone Penetrometer*, in Proceedings of 1st International Symposium on Penetration Testing (ISOPT-1), Orlando, (ed. De Ruiter J.), Balkema A.A., Rotterdam, March, vol. 2, pp 1001-1005.

Tedd P. and Charles J.A. (1981), *In-situ Measurement of Horizontal Stress in Overconsolidated Clay using Push-in Spade-Shaped Pressure Cells*, Géotechnique, vol. 31, n. 4, pp 554-558.

Tringale P.T. and Mitchell J.K. (1982), *An Acoustic Cone Penetrometer for Site Investigations*, in Proceedings of the 2nd European Symposium on Penetration Testing, ESOPT II, Amsterdam, May, Balkema A.A., Rotterdam, vol. 2, pp 909-914.

Van Den Berg H.J. (1987), *In-situ Testing of Soils*, Ground Engineer's Reference Book, (ed. Bell F.G.), Butterworths, London, pp 25/1-25/23.

Weltman A.J. and Head J.M. (1983), *Site Investigation Manual*, CIRIA Special Publication 25, Construction Industry Research and Information Association, London.

Woeller D.J., Weemees I., Kokan M., Jolly G. and Robertson P.K. (1991), *Penetration Testing for Groundwater Contaminants*, Proceedings of the Geotechnical Engineering Congress, in Geotechnical Special Publication, vol. 1, no. 27, (ed. McLean F.G., Campbell DW. A. and Harris D.W.), ASCE, Boulder, Colorado, pp 76-87.

Woods R.D. (1978), *Measurement of Dynamic Soil Properties*, State of the Art Report, Earthquake Engineering and Soil Dynamics, Specialty Conference, Pasadena, CA, June, pp 91-178.

Wroth C.P. (1984), *The Interpretation of In-situ Soil Tests*, Geotechnique 34, No. 4, pp 449-489.

APPENDIX E

QUESTIONNAIRE

PENETRATION TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	
		Standard Penetration Test														
		Dynamic Probing Light Test														
		Dynamic Probing Medium Test														
		Dynamic Probing Heavy Test														
		Dynamic Probing Superheavy Test														
		Mechanical Cone Resistance Friction Test														
		Electrical Cone Resistance Friction Test														
		Piezocone Test														
		Piezocone Friction Test														
		Weight Sounding Test														
		Static - Dynamic Penetration Test														

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Comments: .....

.....

.....

SPECIAL PENETROMETER TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
		Flat Plate Dilatometer Test													
		Cone Pressuremeter Test													
		Seismic Cone Test													
		Lateral Stress Cone Test													
		Vibratory Cone Test													
		Nuclear Density Probe Test													
		Electrical Density Probe Test													
		Electrical Conductivity Cone Test													
		Thermal Conductivity Cone Test													
		Acoustic Cone Test													

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Comments: .....

.....

.....

PRESSUREMETER and IN-SITU STRESS MEASUREMENT TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
		Ménard-type Pressuremeter Test																					
		Push-in Pressuremeter Test																					
		Self-boring Pressuremeter Test																					
		Total Stress Cell Test																					
		Iowa Stepped Blade Test																					
		Hydraulic Fracturing Test																					
		Self-boring K_o meter Test																					

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Comments: .....

.....

.....

SHEAR and BEARING TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (D_r)	COMPRESSIBILITY (mv, Cc)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
		Vane Test																					
		Self-boring Vane Test																					
		Borehole Shear Test																					
		In-situ Shear Test																					
		Plate Loading Tests																					
		Screw Plate Test																					
		Self-boring Plate Test																					
		Pressurized Chamber Test																					
		In-situ California Bearing Ratio Test																					

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Comments: .....

.....

.....

IN-SITU DENSITY TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
		Small Pouring Cylinder Test													
		Large Pouring Cylinder test													
		Scoop Test													
		Core Cutter Test													
		Weight in Water Test													
		Water Replacement Test													
		Rubber Balloon Test													
		Nuclear Backscatter Test													
		Nuclear Direct Transmission Test													
		Nuclear Air Gap Test													

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Comments: .....

.....

.....

PERMEABILITY TESTS

GEOTECHNICAL INFORMATION [H, M, L, N]	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]
Rising Head Test		
Falling Head Test		
Constant Head Test		
Self-boring Permeameter Test		
Pumping Tests		

GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
Rising Head Test													
Falling Head Test													
Constant Head Test													
Self-boring Permeameter Test													
Pumping Tests													

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Comments: .....

.....

.....

GEOPHYSICAL SURVEYING TESTS

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	
		Seismic Refraction Test														
		Seismic ReflectionTest														
		Seismic Cross-Hole Test														
		Seismic Down-Hole Test														
		Surface Wave Test														
		Resistivity Test														
		Gravimetric Test														
		Magnetic Test														

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS

KEY
[H, M, L, N] : [High, Medium, Low, None]
[R, L, S] : [Routine, Less common, Special purpose]

Comments: .....

.....

.....

PENETRATION TESTS (ALL ANSWERS)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
9 1	8 -	Standard Penetration Test		6 2	4 4	- -	2 5	- 6	2 6	- 4	- 1	- -	1 5	- -	- -	- -		- 1	3 4	4 4	10 -	4 5	2 5	1 2
- 1	-			- 1	1 -	- 8	2 -	3 -	1 -	3 2	- 8	1 8	2 1	- 9	2 7	- 9		2 6	2 -	1 1	- -	1 -	3 -	4 2
- 3	- 2	Dynamic Probing Light Test		- 1	3 1	- -	- 2	- 1	- 1	- 1	- -	- -	- 1	- -	- -	- -		- -	- -	1 -	1 2	2 2	1 3	1 2
2 6	3			- 3	- -	- 3	1 1	2 1	1 2	1 2	- 4	- 4	- 3	- 4	- 4	- 4		- 4	2 2	3 -	1 -	- -	- -	1 -
- 3	- 3	Dynamic Probing Medium Test		- 2	2 2	- -	- 2	- 1	- 1	- 1	- -	- -	- 1	- -	- -	- -		- -	- 1	2 -	2 1	2 2	1 3	2 2
2 6	2			- 2	- -	- 3	2 -	3 -	2 1	2 1	- 4	- 4	1 2	- 4	- 4	- 4		- 4	2 1	2 -	1 -	- -	- -	- -
- 2	- 2	Dynamic Probing Heavy Test		- 1	2 1	- -	- 2	- 1	- 1	- 1	- -	- -	- 1	- -	- -	- -		- -	- 1	1 -	1 1	1 2	- 3	1 2
2 7	2			- 2	- -	- 2	1 -	2 -	1 1	1 1	- 3	- 3	- 2	- 3	- 3	- 3		- 3	1 1	2 -	1 -	- -	- -	- -
1 2	- 1	Dynamic Probing		- 1	1 2	- -	- 1	- 1	- 2	- 1	- -	- -	- 1	- -	- -	- -		- 1	- 2	1 2	1 2	- 3	- 3	- 2
2 6	2	Superheavy Test		1 1	- -	- 2	2 -	2 -	1 -	1 1	- 3	- 3	1 1	- 3	- 3	1 2		- 2	1 -	- -	- -	- -	- -	1 -
4 3	3 4	Mechanical Cone Resistance		1 6	4 4	- -	1 5	- 7	1 5	- 3	- -	- -	- 4	- -	- -	- -		- -	1 1	2 2	7 2	8 1	7 2	4 2
2 2	-	Friction Test		1 -	- -	- 7	1 1	1 -	- 2	4 1	- 8	- 8	3 1	1 7	2 6	- 8		- 8	3 3	3 2	- -	- -	- -	2 -
5 4	3 4	Electrical Cone Resistance		1 8	6 3	- -	1 6	- 8	1 6	- 3	- -	- -	- 5	- -	- -	- -		- -	1 1	2 2	7 2	8 1	8 1	4 3
1 1	-	Friction Test		- -	- -	- 7	1 1	1 -	- 2	4 1	- 9	- 8	2 1	1 7	3 6	- 8		- 8	3 3	3 2	- -	- -	- -	1 -
5 1	1 3	Piezocone Test		1 6	5 2	4 2	- 5	- 7	- 6	- 2	1 4	- 3	- 3	- -	- 3	- 1		- -	2 -	1 2	6 2	8 -	8 -	5 1
4 1	3			- 1	- 1	- 1	1 2	- 1	- 2	3 2	1 2	2 2	2 2	2 5	2 3	- 6		- 7	2 3	2 3	- -	- -	- -	1 -
5 1	1 2	Piezocone Friction Test		4 4	7 1	5 2	- 6	- 8	- 6	- 3	2 5	1 3	- 5	- 3	- 4	- 1		- -	2 -	1 2	6 2	8 -	8 -	5 1
3 1	3			- -	- -	- -	1 1	- -	- 2	3 1	1 -	2 1	2 -	2 2	2 2	- 6		- 7	2 3	2 3	- -	- -	- -	1 -
- -	- -	Weight Sounding Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -
-	-			- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -
11																								
2 -	1 -	Static - Dynamic		- 1	1 1	- -	1 1	- 2	- 1	- 2	- -	- -	- 2	- -	- -	- -		- -	1 1	1 1	1 1	1 1	1 1	1 1
- 9	1	Penetration Test		- 1	- -	- 1	- -	- -	1 -	- -	- 2	- 2	- -	- 2	- 2	- 2		- 2	- -	- -	- -	- -	- -	- -

KEY	H M	R L		H M	
	L N	S		L N	

Table E.1 Summation of all the answers obtained for the Penetration Tests

PENETRATION TESTS ('HIGH FAMILIARITY' ANSWERS)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (κ)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
9	7 -	Standard Penetration Test	5 2	3 4	- -	2 5	- 6	1 6	- 4	- 1	- -	1 5	- -	- -	- -
	-		- 1	1 -	- 7	1 -	2 -	1 -	3 1	- 7	1 7	1 1	- 8	2 6	- 8
-	- -	Dynamic Probing Light Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	-		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
-	- -	Dynamic Probing Medium Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	-		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
-	- -	Dynamic Probing Heavy Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	-		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
1	- -	Dynamic Probing	- -	- 1	- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -
	-	Superheavy Test	1 -	- -	- 1	1 -	1 -	- -	- 1	- 1	- 1	1 -	- 1	- 1	1 -
4	2 -	Mechanical Cone Resistance	1 2	3 -	- -	- 3	- 2	- 3	- 1	- -	- -	- 1	- -	- -	- -
	-	Friction Test	- -	- -	- 2	- -	1 -	- -	2 -	- 3	- 3	2 -	1 2	1 2	- 3
5	1 1	Electrical Cone Resistance	1 3	4 -	- -	- 4	- 3	- 4	- 2	- -	- -	- 3	- -	- -	- -
	-	Friction Test	- -	- -	- 2	- -	1 -	- -	1 -	- 4	- 3	- -	1 2	2 2	- 3
5	1 1	Piezocone Test	1 3	3 1	2 -	- 4	- 4	- 4	- 2	1 2	- 1	- 3	- -	- 2	- 1
	-		- -	- -	- 1	- -	- -	- -	1 -	- 1	- 2	- -	1 2	1 1	- 2
5	1 -	Piezocone Friction Test	3 1	4 -	3 -	- 4	- 4	- 4	- 2	2 2	1 1	- 3	- 1	- 3	- 1
	1		- -	- -	- -	- -	- -	- -	1 -	- -	- 1	- -	1 1	1 -	- 2
-	- -	Weight Sounding Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	-		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
2	1 -	Static - Dynamic	- 1	1 1	- -	1 1	- 2	- 1	- 2	- -	- -	- 2	- -	- -	- -
	1	Penetration Test	- 1	- -	- 1	- -	- -	1 -	- -	- 2	- 2	- -	- 2	- 2	- 2

KEY	H	R L S		HM LN	

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	- 1	3 3	4 3	9 -	4 4	2 4	1 2
	2 5	2 -	1 1	- -	1 -	3 -	3 2
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- 1	1 -	- 1	- 1	- 1
	- 1	1 -	- -	- -	- -	- -	- -
	- -	1 -	- 1	3 1	4 -	4 -	2 -
	- 3	2 -	2 1	- -	- -	- -	1 -
	- -	1 -	1 1	4 -	4 -	4 -	3 -
	- 3	1 1	1 1	- -	- -	- -	- -
	- -	1 -	1 1	4 -	4 -	4 -	3 -
	- 3	1 1	1 1	- -	- -	- -	- -
	- -	1 -	1 1	4 -	4 -	4 -	3 -
	- 3	1 1	1 1	- -		- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	1 1	1 1	1 1	1 1	1 1	1 1
	- 2	- -	- -	- -	- -	- -	- -

KEY	HM LN	

Table E.2 Summation of the 'high familiarity' answers obtained for the Penetration Tests

PENETRATION TESTS (AVERAGES)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Standard Penetration Test	M	M	N	M	M	M	L	N	N	M	N	N	N			N	M	M	H	M	M	L
	R		M	M	N	M	M	M	L	N	N	M	N	N	N			N	M	M	H	M	M	L
	S	Dynamic Probing Light Test	N	H	N	L	L	L	L	N	N	N	N	N	N			N	L	L	M	M	M	M
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	Dynamic Probing Medium Test	L	M	N	L	L	L	L	N	N	L	N	N	N			N	L	M	M	M	M	M
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	S	Dynamic Probing Heavy Test	L	H	N	M	L	L	L	N	N	L	N	N	N			N	L	M	M	M	M	M
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	S	Dynamic Probing	L	M	N	L	L	M	L	N	N	L	N	N	N			N	M	M	M	M	M	M
	-	Superheavy Test	L	M	N	L	L	M	N	N	N	L	N	N	L			N	L	M	H	M	M	M
	L	Mechanical Cone Resistance	M	M	N	M	M	M	L	N	N	L	N	N	N			N	L	L	H	H	H	M
	R	Friction Test	M	H	N	M	M	M	L	N	N	L	N	N	N			N	M	L	H	H	H	M
	L	Electrical Cone Resistance	M	H	N	M	M	M	L	N	N	L	N	N	N			N	L	L	H	H	H	M
	L	Friction Test	M	H	N	M	M	M	M	N	N	M	N	N	N			N	L	L	H	H	H	H
	L	Piezocone Test	M	M	M	L	M	L	L	L	L	L	N	L	N			N	L	L	H	H	H	H
	L		M	H	M	M	M	M	M	M	L	M	N	L	L			N	L	L	H	H	H	H
	L	Piezocone Friction Test	M	H	H	M	M	L	L	M	M	M	L	L	N			N	L	L	H	H	H	H
	L		H	H	H	M	M	M	M	M	M	M	L	M	L			N	L	L	H	H	H	H
	-	Weight Sounding Test	-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	Static - Dynamic Penetration Test	L	M	N	M	M	L	M	N	N	M	N	N	N			N	M	M	M	M	M	M

KEY			
A_I	Average value taking into account all answers	A	Average value; all answers are 'high familiarity' answers
A_{II}	Average value taking into account 'high familiarity' answers		

Table E.3 Average values calculated for the Penetration Tests

SPECIAL PENETROMETER TESTS (ALL ANSWERS)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_T)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
3 1	- 1	Flat Plate Dilatometer Test		- 1	2 -	- -	- 2	1 3	- -	1 3	- -	- -	1 3	2 2	1 2	1 3
1 6	3			2 1	1 1	2 2	1 1	- -	3 1	- -	- 4	- 4	- -	- -	1 -	- -
1 2	- -	Cone Pressuremeter Test		- 1	1 2	- -	2 1	1 2	1 -	- 3	1 2	1 -	2 3	- 3	- 2	2 1
2 6	6			2 1	1 1	1 4	1 1	1 1	2 2	1 1	1 1	1 3	- -	- 2	2 1	2 -
2 -	- -	Seismic Cone Test		1 2	2 1	1 -	- 1	- 3	- 1	- 2	1 -	- 1	3 -	- 1	- 1	- 1
2 7	4			1 -	1 -	1 2	1 2	- 1	2 1	1 1	1 2	1 2	1 -	1 2	1 2	1 2
- -	- -	Lateral Stress Cone Test		- -	- -	- -	- -	- 1	- -	- 1	- -	- -	- 2	- 2	- 1	- -
1 10	3			1 1	1 1	1 1	1 1	- 1	1 1	- 1	1 1	1 1	- -	- -	1 -	1 1
- -	- -	Vibratory Cone Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
- 11	2			- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1
1 2	- 1	Nuclear Density Probe Test		- -	- -	- -	- 1	- -	2 1	- -	- -	- -	- -	- -	- -	- 0
- 7	2			- 3	- 3	- 3	- 2	1 2	- -	- 3	- 3	- 3	- 3	1 2	- 3	1 2
1 1	- -	Electrical Density Probe Test		- 1	- 1	- -	- 1	- -	1 1	- -	- -	- -	- -	- -	- -	- -
- 9	1			- 1	- 1	- 2	- 1	2 -	- -	1 1	- 2	- 2	1 1	- 2	- 2	- 2
1 1	- -	Electrical Conductivity Cone Test		1 -	- 1	- -	- -	- -	1 -	- 1	- -	- -	- 1	- -	- -	- -
- 9	2			1 -	1 -	- 2	1 1	1 1	- 1	- 1	1 1	1 1	- 1	1 1	1 1	1 1
- 1	- -	Thermal Conductivity Cone Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
- 10	2			- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1
1 -	- -	Acoustic Cone Test		- 1	- 1	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
- 10	2			- 1	- 1	- 2	1 1	1 1	1 1	1 1	- 2	- 2	1 1	- 2	1 1	- 2

KEY	H M	R L		H M	
	L N	S		L N	

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	- -	1 -	- -	3 -	5 -	5 -	4 -
	1 3	1 2	3 2	2 -	- -	- -	- -
	- -	- 2	- 1	3 -	3 2	3 1	3 1
	1 3	- 2	1 3	2 -	- -	- -	- -
	- -	- 2	- 1	3 -	3 1	3 1	2 1
	1 3	1 1	1 2	1 -	- -	- -	1 -
	- -	- 1	- -	- -	- 1	- 1	- 1
	1 -	- -	1 -	1 -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- 1	3 -	3 -	2 1	2 1
	- 3	1 2	- 2	- -	- -	- -	- -
	- -	- -	- -	2 -	2 -	2 -	2 -
	- 2	1 1	- 2	- -	- -	- -	- -
	- -	- -	- -	2 -	2 -	2 -	1 1
	- 2	- 2	- 2	- -	- -	- -	- -
	- -	- -	- -	1 -	1 -	1 -	1 -
	- 1	- 1	- 1	- -	- -	- -	- -
	- -	- -	- -	1 -	1 -	1 -	1 -
	- 1	1 -	- 1	- -	- -	- -	- -

H M
L N

Table E.4 Summation of all the answers obtained for the Special Penetrometer Tests

SPECIAL PENETROMETER TESTS ('HIGH FAMILIARITY' ANSWERS)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G:E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
3	- -	Flat Plate Dilatometer Test		- 1	1 -	- -	- 1	- 2	- -	- 2	- -	- -	- 2	- 2	- 2	- 2			- -	- -	- -	3 -	3 -	3 -	3 -
	1			- 1	- 1	1 1	- 1	- -	2 -	- -	- 2	- 2	- -	- -	- -	- -			- 2	1 1	1 2	- -	- -	- -	- -
1	- -	Cone Pressuremeter Test		- -	- -	- -	- -	- 1	- -	- 1	- 1	- -	- 1	- 1	- 1	- 1			- -	- 1	- -	1 -	1 -	1 -	1 -
	1			- 1	- 1	- 1	- 1	- -	1 -	- -	- -	- 1	- -	- -	- -	- -			- 1	- -	- 1	- -	- -	- -	- -
2	- -	Seismic Cone Test		1 1	2 -	1 -	- 1	- 1	- 1	- 1	1 -	- 1	- 1	2 -	- 1	- 1	- 1		- -	- 1	- 1	2 -	2 -	2 -	2 -
	1			- -	- -	- 1	- 1	- 1	1 -	1 -	- 1	- 1	- -	- -	- 1	- 1	- 1		- 2	1 -	- 1	- -	- -	- -	- -
-	- -	Lateral Stress Cone Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -	- -
	-			- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -	- -
-	- -	Vibratory Cone Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -	- -
	-			- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -	- -
1	- -	Nuclear Density Probe Test		- -	- -	- -	- 1	- -	1 -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	1 -	1 -	- 1	1 -
	-			- 1	- 1	- 1	- -	- 1	- -	- 1	- 1	- 1	- 1	1 -	- 1	1 -			- 1	- 1	- 1	- -	- -	- -	- -
1	- -	Electrical Density Probe Test		- 1	- 1	- -	- 1	- -	1 -	- -	- -	- -	- -	- -	- -	- -			- -	- -	- -	1 -	1 -	1 -	- 1
	-			- -	- -	- 1	- -	1 -	- -	1 -	- 1	- 1	1 -	- 1	- 1	- 1			- 1	1 -	- 1	- -	- -	- -	- -
1	- -	Electrical Conductivity Cone Test		1 -	- 1	- -	- -	- -	1 -	- 1	- -	- -	- 1	- -	- -	- -			- -	- -	- -	1 -	1 -	1 -	1 -
	-			- -	- -	- 1	1 -	1 -	- -	- -	1 -	1 -	- -	1 -	1 -	1 -			- 1	- 1	- 1	- -	- -	- -	- -
-	- -	Thermal Conductivity Cone Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -	- -
	-			- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -	- -
1	- -	Acoustic Cone Test		- 1	- 1	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	1 -	1 -	1 -	1 -
	-			- -	- -	- 1	1 -	1 -	1 -	1 -	- 1	- 1	1 -	- 1	1 -	- 1			- 1	1 -	- 1	- -	- -	- -	- -

KEY	H M	R L		H M	
	L N	S		L N	

Table E.5 Summation of the 'high familiarity' answers obtained for the Special Penetrometer Tests

SPECIAL PENETROMETER TESTS (AVERAGES)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (Dr)	COMPRESSIBILITY (mv, Cc)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	S	Flat Plate Dilatometer Test		L	M	N	L	M	L	M	N	N	M	M	M	M			N	L	L	M	H	H	H
	S			L	L	N	L	M	L	M	N	N	M	M	M	M			N	N	N	H	H	H	H
	S	Cone Pressuremeter Test		L	M	N	M	M	L	L	M	L	M	L	L	M			N	L	L	M	H	M	M
	S			N	N	N	N	M	L	M	M	N	M	M	M	M			N	M	N	H	H	H	H
	S	Seismic Cone Test		M	M	L	L	L	L	L	L	L	M	L	L	L			N	L	L	M	H	H	M
	S			M	H	L	L	L	L	L	L	L	H	L	L	L			N	L	L	H	H	H	H
	S	Lateral Stress Cone Test		N	N	N	N	L	N	L	N	N	M	M	L	L			L	M	L	L	M	M	M
	-			-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	S	Vibratory Cone Test		N	N	N	N	N	N	N	N	N	N	N	N	N			-	-	-	-	-	-	-
	-			-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	S	Nuclear Density Probe Test		N	N	N	L	N	H	N	N	N	N	N	N	N			N	N	L	H	H	H	H
	-			N	N	N	M	N	H	N	N	N	N	L	N	L			N	N	N	H	H	M	H
	S	Electrical Density Probe Test		L	L	N	L	N	M	N	N	N	N	N	N	N			N	N	N	H	H	H	H
	-			M	M	N	M	L	H	L	N	N	L	N	N	N			N	L	N	H	H	H	H
	S	Electrical Conductivity Cone		M	L	N	N	N	L	M	N	N	L	N	N	N			N	N	N	H	H	H	M
	-	Test		H	M	N	L	L	H	M	L	L	M	L	L	L			N	N	N	H	H	H	M
	S	Thermal Conductivity Cone		N	N	N	N	N	N	N	N	N	N	N	N	N			N	N	N	H	H	H	H
	-	Test		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	S	Acoustic Cone Test		L	L	N	N	N	N	N	N	N	N	N	N	N			N	L	N	H	H	H	H
	-			M	M	N	L	L	L	L	N	N	L	N	L	N									

KEY			
A_I	Average value taking into account all answers	A	Average value; all answers are 'high familiarity' answers
A_{II}	Average value taking into account 'high familiarity' answers		

Table E.6 Average values calculated for the Special Penetrometer Tests

PRESSUREMETER and IN-SITU STRESS MEASUREMENT TESTS (ALL ANSWERS)

	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (Dr)	COMPRESSIBILITY (mv, Cc)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
	4 5	- 4	Ménard-type Pressuremeter		- 1	- 1	- -	1 1	1 4	- 1	- 3	- 1	- -	3 2	- 3	- 2	3 1
	- 2	3	Test		1 5	3 3	- 7	5 -	3 -	3 3	1 3	1 5	- 7	3 -	5 -	1 4	3 1
	1 1	- 2	Push-in Pressuremeter Test		1 1	- 2	- 1	1 -	1 2	- 1	- 1	1 -	- 1	2 1	- 1	- 1	- 2
	3 6	2			1 2	1 2	- 4	4 -	2 -	2 2	2 2	1 3	- 4	2 -	4 -	1 3	2 1
	6 3	- 2	Self-boring Pressuremeter Test		- 2	- 3	2 1	3 1	1 6	- 2	1 4	3 2	1 1	7 1	6 2	2 1	6 1
	- 2	5			2 3	2 2	2 2	4 -	1 -	3 2	1 1	1 2	2 4	- -	- -	1 4	- 1
	1 -	- -	Total Stress Cell Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	1 1	- 1	- -
	1 9	1			- 2	- 2	- 2	- 2	- 2	- 2	- 2	- 2	- 2	- 2	- -	- 1	- 2
	1 -	- -	Iowa Stepped Blade Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- 1	- 1	- -
	- 10	1			- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- -	- -	- 1
	1 1	- -	Hydraulic Fracturing Test		- -	- -	1 -	- -	- -	- -	- -	- -	- -	- -	- 1	- 1	- -
	- 9	3			- 1	- 1	- -	- 1	- 1	- 1	- 1	1 -	1 -	- 1	1 -	1 -	- 1
	1 1	- -	Self-boring K_o meter Test		- 1	- 1	1 -	- -	- -	- -	- -	- -	- -	- -	3 -	2 -	- -
	1 6	2			- 2	- 2	- 2	- 3	- 3	- 3	- 3	- 3	- 3	- 3	- -	- 1	- 3
KEY	H M	R L			H M												
	L N	S			L N												

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	1 1	4 3	- 4	3 4	3 6	7 2	2 2
	2 3	1 -	3 1	2 -	- -	- -	3 1
	- -	- -	- -	1 2	4 -	4 -	1 1
	- 5	2 3	2 3	1 1	1 -	1 -	3 -
	1 -	2 4	- -	4 3	8 1	9 -	5 -
	1 5	2 -	6 2	2 -	- -	- -	2 1
	- -	- -	- -	- 1	1 -	2 -	1 -
	- 2	- 2	1 1	- 1	1 -	- -	1 -
	- -	- -	- -	- 1	1 -	1 -	- 1
	- 1	- 1	- 1	- -	- -	- -	- -
	1 1	- 2	- -	- -	- 1	1 -	- -
	- -	- -	1 1	1 1	- 1	- 1	1 1
	- -	- 1	- -	- 3	3 -	3 -	2 -
	- 3	- 2	1 2	- -	- -	- -	1 -
KEY	H M						
	L N						

Table E.7 Summation of all the answers obtained for the Pressuremeter and In-Situ Stress Measurement Tests

PRESSUREMETER and IN-SITU STRESS MEASUREMENT TESTS ('HIGH FAMILIARITY' ANSWERS)

	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE		GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
E15	4 -	- 1	Ménard-type Pressuremeter	- 1	- 1	- -	1 -	1 2	- -	- -	- -	- -	2 -	- 1	- 1	1 -			1 1	3 -	- 3	1 3	1 3	3 1	1 1
	- -	1	Test	- 1	- 1	- 2	1 -	- -	1 1	1 1	1 1	- 2	1 -	2 -	1 -	2 -			- -	- -	- -	- -	- -	- -	1 -
	1 -	- -	Push-in Pressuremeter Test	1 -	- 1	- 1	- -	- 1	- -	- -	1 -	- 1	1 -	- -	1 -	1 -			- -	- -	- -	- 1	1 -	1 -	- 1
	- -	-		- -	- -	- -	1 -	- -	1 -	1 -	- -	- -	- -	1 -	- -	- -			- 1	- 1	- 1	- -	- -	- -	- -
	6 -	- 1	Self-boring Pressuremeter Test	- 2	- 2	1 1	3 1	1 4	- 2	1 2	3 -	1 1	5 -	5 -	2 -	3 1			1 -	2 1	- -	3 2	6 -	6 -	4 -
	- -	3		2 -	2 -	2 -	1 -	- -	1 1	- 1	1 1	1 2	- -	- -	- 3	- 1			- 3	2 -	4 1	1 -	- -	- -	1 -
	1 -	- -	Total Stress Cell Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- 1	- 1	- -			- -	- -	- -	- -	- -	1 -	1 -
	- -	-		- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- -	- -	- 1			- 1	- 1	- 1	- 1	1 -	- -	- -
	1 -	- -	Iowa Stepped Blade Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- 1	- 1	- -			- -	- -	- -	- 1	1 -	1 -	- 1
	- -	-		- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- -	- -	- 1			- 1	- 1	- 1	- -	- -	- -	- -
	1 -	- -	Hydraulic Fracturing Test	- -	- -	1 -	- -	- -	- -	- -	- -	- -	- -	- 1	- 1	- -			- 1	- 1	- -	- -	- 1	1 -	- -
	- -	-		- 1	- 1	- -	- 1	- 1	- 1	- 1	1 -	1 -	- 1	- -	- -	- 1			- -	- -	1 -	1 -	- -	- -	1 -
	1 -	- -	Self-boring K_o meter Test	- 1	- 1	- -	- -	- -	- -	- -	- -	- -	- -	1 -	1 -	- -			- -	- -	- -	- 1	1 -	1 -	1 -
	- -	-		- -	- -	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- -	- -	- 1			- 1	- 1	- 1	- -	- -	- -	- -

KEY	HM LN	RL S		HM LN	

HM LN	

Table E.8 Summation of the 'high familiarity' answers obtained for the Pressuremeter and In-Situ Stress Measurement Tests

PRESSUREMETER and IN-SITU STRESS MEASUREMENT TESTS (AVERAGES)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE		GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	L	Ménard-type Pressuremeter		N	L	N	L	M	L	L	N	N	M	L	L	M			L	M	L	M	M	H	M
	S	Test		L	L	N	M	M	N	N	N	N	M	L	L	M			M	H	M	M	M	H	M
	S	Push-in Pressuremeter Test		L	L	N	L	M	L	L	L	N	M	L	L	L			N	N	N	M	H	H	M
	-			H	M	M	L	M	L	L	H	M	H	L	L	L			N	N	N	M	H	H	M
	S	Self-boring Pressuremeter Test		L	L	L	M	M	L	M	M	L	H	H	L	M			L	M	L	M	H	H	M
	S			L	L	M	M	M	L	L	L	L	H	H	L	M			L	M	L	M	H	H	H
	S	Total Stress Cell Test		N	N	N	N	N	N	N	N	N	N	M	L	N			N	N	N	L	M	H	M
	-			N	N	N	N	N	N	N	N	N	N	M	M	N			N	N	N	N	L	H	H
	S	Iowa Stepped Blade Test		N	N	N	N	N	N	N	N	N	N	M	M	N			N	N	N	M	H	H	M
	S	Hydraulic Fracturing Test		N	N	H	N	N	N	N	L	L	N	L	L	N			M	M	N	N	L	L	N
	-													M	M				M	M	L	L	M	H	L
	S	Self-boring K_o meter Test		L	L	L	N	N	N	N	N	N	N	H	M	N			N	L	N	M	H	H	M
	-			M	M	N	N	N	N	N	N	N	N	H	H	N			N	N	N	M	H	H	H

KEY			
A_I	Average value taking into account all answers	A	Average value; all answers are 'high familiarity' answers
A_{II}	Average value taking into account 'high familiarity' answers		

Table E.9 Average values calculated for the Pressuremeter and In-Situ Stress Measurement Tests

SHEAR and BEARING TESTS (ALL ANSWERS)

	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
	72	52	Vane Test	--	-2	--	--	8-	--	--	--	--	--	--	-1	--
	-2	-		16	23	-7	-7	--	-7	-7	-7	-7	-7	16	-6	-7
	2-	1-	Self-boring Vane Test	-1	-1	--	--	2-	--	--	--	--	--	--	-1	--
	-9	-		-1	-1	-2	-2	--	-2	-2	-2	-2	-2	11	-1	-2
	1-	--	Borehole Shear Test	--	--	--	-1	--	--	--	--	--	--	--	--	--
	-	-		1-	1-	-1	--	1-	-1	-1	-1	-1	1-	-1	1-	-1
	10															
	--	--	In-situ Shear Test	--	--	--	1-	1-	--	--	--	--	--	--	--	--
	1-	1		-1	-1	-1	--	--	-1	-1	1-	-1	1-	-1	-1	1-
	52	-2	Plate Loading Tests	--	--	--	--	-4	-1	15	11	--	42	--	-1	25
	-4	4		15	15	-6	24	2-	-5	-1	22	24	1-	15	-5	--
	11	--	Screw Plate Test	--	--	--	--	11	-1	-2	-1	--	1-	--	-1	-2
	-9	1		11	11	-2	11	--	-1	--	1-	11	1-	11	-1	--
	1-	--	Self-boring Plate Test	-1	-1	--	--	-1	-1	-1	--	--	1-	-1	1-	--
	-	-		--	--	-1	1-	--	--	--	1-	1-	--	--	--	1-
	10															
	--	--	Pressurized Chamber Test	--	--	--	--	--	--	--	--	--	--	--	--	--
	--	-		--	--	--	--	--	--	--	--	--	--	--	--	--
	6-	32	In-situ California Bearing	--	--	--	--	-1	--	--	--	--	-1	--	--	--
	14	1	Ratio Test	-6	-6	-6	-6	42	-6	15	15	15	33	15	15	25

KEY	HM	RL		HM	
	LN	S		LN	

GROUND CONDITIONS [H, M, L, N]

HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
--	1-	--	--	15	9-	24
-8	16	-9	-9	12	--	11
--	--	--	-1	1-	2-	1-
-2	-2	-2	-1	-1	--	-1
-1	-1	--	-1	-1	--	--
--	--	1-	--	--	1-	1-
--	-1	--	--	-1	-1	--
-1	--	-1	-1	--	--	-1
23	43	43	43	52	52	32
-2	--	--	--	--	--	11
1-	1-	1-	2-	2-	2-	2-
-1	-1	-1	--	--	--	--
--	--	--	-1	1-	1-	-1
-1	-1	-1	--	--	--	--
--	--	--	--	--	--	--
--	--	--	--	--	--	--
-1	13	23	42	42	42	3-
-6	12	11	-1	-1	-1	22

HM	
LN	

Table E.10 Summation of all the answers obtained for the Shear and Bearing Tests

SHEAR and BEARING TESTS ('HIGH FAMILIARITY' ANSWERS)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (Dr)	COMPRESSIBILITY (mv, Cc)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_O)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
7 -	4 1	Vane Test		- -	- 2	- -	- -	6 -	- -	- -	- -	- -	- -	- -	- 1	- -			- -	1 -	- -	- -	1 4	7 -	2 2
- -	-			1 4	2 1	- 5	- 5	- -	- 5	- 5	- 5	- 5	- 5	1 4	- 4	- 5			- 6	1 4	- 7	- 7	1 1	- -	1 1
2 -	1 -	Self-boring Vane Test		- 1	- 1	- -	- -	2 -	- -	- -	- -	- -	- -	- -	- 1	- -			- -	- -	- -	- 1	1 -	2 -	1 -
- -	-			- 1	- 1	- 2	- 2	- -	- 2	- 2	- 2	- 2	- 2	1 1	- 1	- 2			- 2	- 2	- 2	- 1	- 1	- -	- 1
1 -	- -	Borehole Shear Test		- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -	- -	- -			- 1	- 1	- -	- 1	- 1	- -	- -
- -	-			1 -	1 -	- 1	- -	1 -	- 1	- 1	- 1	- 1	1 -	- 1	1 -	- 1			- -	- -	1 -	- -	- -	1 -	1 -
- -	- -	In-situ Shear Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -			- -	- -	- -	- -	- -	- -	- -
- -	-			- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -			- -	- -	- -	- -	- -	- -	- -
5 -	- 1	Plate Loading Tests		- -	- -	- -	- -	1 3	- 1	1 3	1 1	- -	3 1	- -	- 1	1 4			1 3	3 2	3 2	3 2	4 1	4 1	2 2
- -	3			1 4	1 4	- 5	2 3	1 -	- 4	- 1	2 1	2 3	1 -	1 4	- 4	- -			- 1	- -	- -	- -	- -	- -	1 -
1 -	- -	Screw Plate Test		- -	- -	- -	- -	- 1	- 1	- 1	- -	- -	1 -	- -	- 1	- 1			- -	- -	- -	1 -	1 -	1 -	1 -
- -	-			1 -	1 -	- 1	1 -	- -	- -	- -	1 -	1 -	- -	1 -	- -	- -			- 1	- 1	- 1	- -	- -	- -	- -
1 -	- -	Self-boring Plate Test		- 1	- 1	- -	- -	- 1	- 1	- 1	- -	- -	1 -	- 1	1 -	- -			- -	- -	- -	- 1	1 -	1 -	- 1
- -	-			- -	- -	- 1	1 -	- -	- -	- -	1 -	1 -	- -	- -	- -	1 -			- 1	- 1	- 1	- -	- -	- -	- -
- -	- -	Pressurized Chamber Test		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -			- -	- -	- -	- -	- -	- -	- -
- -	-			- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -			- -	- -	- -	- -	- -	- -	- -
6 -	3 1	In-situ California Bearing		- -	- -	- -	- -	- 1	- -	- -	- -	- -	- 1	- -	- -	- -			- 1	1 3	2 3	3 2	3 2	3 2	2 -
- -	1	Ratio Test		- 5	- 5	- 5	- 5	4 1	- 5	1 4	1 4	1 4	3 2	1 4	1 4	2 4			- 5	- 2	- 1	- 1	- 1	- 1	2 2

KEY	H M L N	R L S		H M L N	

H M L N	

Table E.11 Summation of the 'high familiarity' answers obtained for the Shear and Bearing Tests

SHEAR and BEARING TESTS (AVERAGES)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE		GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Vane Test	N	L	N	N	H	N	N	N	N	N	N	N	N			N	L	N	N	M	H	M
	R		N	L	N	N	H	N	N	N	N	N	N	N	N			N	L	N	N	M	H	M
	R	Self-boring Vane Test	L	L	N	M	L	N	N	N	N	L	N	L	N			N	N	N	L	L	H	L
	-	Borehole Shear Test	L	L	N	M	L	N	N	N	N	L	N	L	N			M	M	L	M	M	L	L
	S	In-situ Shear Test	N	N	N	H	H	N	N	L	N	L	N	N	L			N	M	N	N	M	M	N
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	S	Plate Loading Tests	N	N	N	N	M	N	M	L	N	M	N	N	M			M	H	H	H	H	H	M
	S		N	N	N	N	M	N	M	L	N	M	N	N	M			M	H	H	H	H	H	M
	S	Screw Plate Test	N	N	N	N	M	L	M	L	N	M	N	L	M			L	L	L	H	H	H	H
	-		L	L	N	L	M	M	M	L	L	H	L	M	M			N	N	N	H	H	H	H
	-	Self-boring Plate Test	M	M	N	L	M	M	M	L	L	H	M	H	L			N	N	N	M	H	H	M
	-	Pressurized Chamber Test	-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	In-situ California Bearing	N	N	N	N	L	N	N	N	N	L	N	N	N			N	L	M	M	M	M	M
	L	Ratio Test	N	N	N	N	L	N	N	N	N	L	N	N	N			N	L	M	M	M	M	L

KEY			
A_I	Average value taking into account all answers	A	Average value; all answers are 'high familiarity' answers
A_{II}	Average value taking into account 'high familiarity' answers		

Table E.12 Average values calculated for the Shear and Bearing Tests

IN-SITU DENSITY TESTS (ALL ANSWERS)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_T)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
5 1	4 1	Small Pouring Cylinder Test	1 -	- -	- -	- -	- -	5 1	- -	- -	- -	- -	- -	- -	- -
- 5	-		- 3	- 3	- 3	- 3	- 3	- -	- 3	- 3	- 3	- 3	- 3	- 3	- 3
6 -	4 2	Large Pouring Cylinder test	1 -	- -	- -	- -	- -	6 -	- -	- -	- -	- -	- -	- -	- -
- 5	-		- 2	- 2	- 2	- 2	- 2	- -	- 2	- 2	- 2	- 2	- 2	- 2	- 2
- 1	- 1	Scoop Test	- -	- -	- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -
- 10	1		- 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1	- 1	- 1	- 1	- 1
3 2	2 2	Core Cutter Test	- -	- -	- -	- -	- -	5 1	- -	- -	- -	- -	- -	- -	- -
1 5	-		1 3	- 3	- 3	- 3	- 3	- -	- 3	- 3	- 3	- 3	- 3	- 3	- 3
2 -	1 -	Weight in Water Test	- -	- -	- -	- -	- -	1 1	- -	- -	- -	- -	- -	- -	- -
- 9	2		1 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1	- 1	- 1	- 1	- 1
1 4	- 3	Water Replacement Test	- -	- -	- -	- -	- -	3 2	- -	- -	- -	- -	- -	- -	- -
- 6	2		1 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1	- 1	- 1	- 1	- 1
1 -	- 1	Rubber Balloon Test	- 1	- -	- -	- -	- -	1 1	- -	- -	- -	- -	- -	- -	- -
1 8	2		- 1	- 1	- 1	- 1	- 1	1 -	- 1	- 1	- 1	- 1	- 1	- 1	- 1
2 2	2 -	Nuclear Backscatter Test	- 1	- -	- -	- -	- -	2 1	- -	- -	- -	- -	- -	- -	- -
- 7	-		- 3	- 3	- 3	- 3	- 3	1 -	1 2	1 2	1 2	- 3	- 3	- 3	- 3
2 1	2 -	Nuclear Direct Transmission Test	- 1	- -	- -	- -	- -	1 2	- -	- -	- -	- -	- -	- -	- -
- 8	-		- 2	- 2	- 2	- 2	- 2	- -	- 2	- 2	- 2	- 2	- 2	- 2	- 2
- 1	- -	Nuclear Air Gap Test	- -	- -	- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -
- 10	2		1 -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -

KEY	HM	RL		HM	
	I.N	S		I.N	

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	- -	2 3	2 -	4 2	4 2	4 1	- 1
	1 5	1 -	3 1	- -	- -	1 -	3 2
	- -	2 2	3 2	5 1	5 1	4 1	3 1
	1 5	1 1	1 -	- -	- -	1 -	2 -
	- -	- -	- -	- 1	- 1	- 1	- -
	- 1	1 -	1 -	- -	- -	- -	1 -
	- 1	- 2	- -	- 1	1 3	3 2	- 4
	- 5	2 2	1 5	1 4	- 2	1 -	2 -
	1 1	- 1	- -	- -	- -	- -	- -
	- -	1 -	- 2	- 2	- 2	2 -	- 2
	- 2	2 3	- 1	1 2	1 2	1 2	- 2
	1 2	- -	1 3	- 2	- 2	2 -	1 2
	- -	- -	- -	1 -	1 -	1 -	- -
	1 2	1 2	2 1	2 -	2 -	2 -	2 1
	- 1	1 1	2 1	2 1	2 1	2 1	- 1
	1 2	1 1	1 -	1 -	1 -	1 -	2 1
	- 2	- 3	- 3	1 2	1 2	1 2	- 1
	- 1	- -	- -	- -	- -	- -	2 -
	- -	- -	- -	- -	- -	- -	- -
	1 -	1 -	1 -	1 -	1 -	1 -	1 -

HM	
I.N	

Table E.13 Summation of all the answers obtained for the In-Situ Density Tests

IN-SITU DENSITY TESTS ('HIGH FAMILIARITY' ANSWERS)

	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (Su)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
1	5 -	4 1	Small Pouring Cylinder Test	1 -	- -	- -	- -	- -	5 -	- -	- -	- -	- -	- -	- -	- -
	- -	-		- 2	- 2	- 2	- 2	- 2	- -	- 2	- 2	- 2	- 2	- 2	- 2	- 2
2	6 -	4 2	Large Pouring Cylinder test	1 -	- -	- -	- -	- -	6 -	- -	- -	- -	- -	- -	- -	- -
	- -	-		- 2	- 2	- 2	- 2	- 2	- -	- 2	- 2	- 2	- 2	- 2	- 2	- 2
3	- -	- -	Scoop Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	- -	-		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
4	3 -	2 -	Core Cutter Test	- -	- -	- -	- -	- -	2 1	- -	- -	- -	- -	- -	- -	- -
	- -	-		1 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1	- 1	- 1	- 1	- 1
5	2 -	1 -	Weight in Water Test	- -	- -	- -	- -	- -	1 1	- -	- -	- -	- -	- -	- -	- -
	- -	1		1 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1	- 1	- 1	- 1	- 1
6	1 -	- -	Water Replacement Test	- -	- -	- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -
	- -	1		1 -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
7	1 -	- -	Rubber Balloon Test	- 1	- -	- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -
	- -	1		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
8	2 -	2 -	Nuclear Backscatter Test	- -	- -	- -	- -	- -	1 -	- -	- -	- -	- -	- -	- -	- -
	- -	-		- 2	- 2	- 2	- 2	- 2	1 -	- 2	- 2	- 2	- 2	- 2	- 2	- 2
9	2 -	2 -	Nuclear Direct Transmission Test	- -	- -	- -	- -	- -	1 -	- -	- -	- -	- -	- -	- -	- -
	- -	-		- 2	- 2	- 2	- 2	- 2	- -	- 2	- 2	- 2	- 2	- 2	- 2	- 2
10	- -	- -	Nuclear Air Gap Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	- -	-		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -

KEY	H M	R L		H M
	L N	S		L N

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
1	- -	2 2	2 -	4 1	4 1	4 -	- -
	1 4	1 -	3 -	- -	- -	1 -	3 2
2	- -	2 2	3 2	5 1	5 1	4 1	3 1
	1 5	1 1	1 -	- -	- -	1 -	2 -
3	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
4	- -	- 1	- -	- -	- 2	2 1	- 3
	- 3	1 1	- 3	1 2	- 1	- -	- -
5	1 1	- 1	- -	- -	- -	- -	- -
	- -	1 -	- 2	- 2	- 2	- 2	- 2
6	- 1	- 1	- -	- -	- -	- -	- -
	- -	- -	- 1	- 1	- 1	1 -	- 1
7	- -	- -	- -	- -	- -	- -	- -
	- 1	- 1	1 -	1 -	1 -	1 -	1 -
8	- -	1 -	1 -	1 -	1 -	1 -	- -
	1 1	1 -	1 -	1 -	1 -	1 -	2 -
9	- 1	- 2	- 2	1 1	1 1	1 1	- -
	- 1	- -	- -	- -	- -	- -	2 -
10	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -

HM
L N

Table E.14 Summation of the 'high familiarity' answers obtained for the In-Situ Density Tests

IN-SITU DENSITY TESTS (AVERAGES)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (Φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Small Pouring Cylinder Test	L	N	N	N	N	H	N	N	N	N	N	N	N			N	M	L	H	H	M	L
	R		L	N	N	N	N	H	N	N	N	N	N	N	N			N	M	M	H	H	H	L
	R	Large Pouring Cylinder test	L	N	N	N	N	H	N	N	N	N	N	N	N			N	M	M	H	H	M	M
	R																							
	S	Scoop Test	N	N	N	N	N	N	N	N	N	N	N	N	N			N	L	L	M	M	M	L
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	Core Cutter Test	N	N	N	N	N	H	N	N	N	N	N	N	N			N	L	N	N	L	M	M
	R		N	N	N	N	N	H	N	N	N	N	N	N	N			N	L	N	N	L	H	M
	L	Weight in Water Test	N	N	N	N	N	M	N	N	N	N	N	N	N			M	L	N	N	N	N	N
	L																							
	L	Water Replacement Test	N	N	N	N	N	H	N	N	N	N	N	N	N			L	L	L	L	L	M	L
	S		L	-	-	-	-	M	-	-	-	-	-	-	-			M	L	N	N	N	L	N
	S	Rubber Balloon Test	L	N	N	N	N	M	N	N	N	N	N	N	N			N	N	L	M	M	M	L
	S		M	-	-	-	-	M	-	-	-	-	-	-	-			N	N	L	L	L	L	L
	R	Nuclear Backscatter Test	N	N	N	N	N	M	N	N	N	N	N	N	N			L	L	M	M	M	M	L
	R		N	N	N	N	N	M	N	N	N	N	N	N	N			L	M	M	M	M	M	L
	R	Nuclear Direct Transmission	L	N	N	N	N	M	N	N	N	N	N	N	N			L	M	M	M	M	M	L
	R	Test	N	N	N	N	N	M	N	N	N	N	N	N	N			L	M	M	M	M	M	L
	S	Nuclear Air Gap Test	L	-	-	-	-	M	-	-	-	-	-	-	-			L	L	L	L	L	L	L
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-

KEY			
A_I	Average value taking into account all answers	A	Average value; all answers are 'high familiarity' answers
A_{II}	Average value taking into account 'high familiarity' answers		

Table E.15 Average values calculated for the In-Situ Density Tests

PERMEABILITY TESTS (ALL ANSWERS)

	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
Rising Head Test	5 1 1 4	6 1 -		- - 1 5	- - - 6	2 1 - 3	- - - 6	- - - 6	- - - 6	- - - 6	- 1 1 4	3 3 1 -	- - - 6	- - - 6	- - - 6	- - - 6	
Falling Head Test	6 1 1 3	7 1 -		- - 1 6	- 1 - 6	1 1 - 5	- - - 7	- - - 7	- - - 7	- - - 7	- 1 1 5	3 3 1 -	- - - 7	- - - 7	- - - 7	- - - 7	
Constant Head Test	4 3 - 4	5 2 -		- - 1 5	- - - 6	1 2 - 3	- - - 6	- - - 6	- - - 6	- - - 6	1 1 - 4	3 4 - -	- - - 6	- - - 6	- - - 6	- - - 6	
Self-boring Permeameter Test	- - 3 8	- 1 2		- - - 2	- - - 2	- 1 - 1	- - - 2	- - - 2	- - - 2	- - - 2	- - - 2	3 - - -	- - - 2	- - - 2	- - - 2	- - - 2	
Pumping Tests	2 3 2 4	1 4 1		- - 1 4	- - - 5	1 1 - 3	- - - 5	- - - 5	- - - 5	- - - 5	- 1 - 4	5 1 - -	- - - 5	- - - 5	- - - 5	- - - 5	

KEY	HM LN	RL S		HM LN	

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	1 1 1 4	1 2 - 4	4 2 - 1	5 2 - -	3 4 - -	- 3 2 2	- 2 4 1
	1 1 2 4	1 3 - 4	5 2 - 1	7 1 - -	3 5 - -	- 2 4 2	- 2 4 2
	1 1 - 5	1 3 - 3	2 2 2 1	3 2 1 1	2 5 - -	3 - 3 1	1 1 4 1
	- - - 3	- 1 - 2	- - 1 2	1 2 - -	2 1 - -	2 - 1 -	- 1 1 1
	3 1 - 2	2 2 - 2	6 - - -	6 - - -	2 4 - -	- 2 2 2	- 2 1 3

HM LN	

Table E.16 Summation of all the answers obtained for the Permeability Tests

PERMEABILITY TESTS ('HIGH FAMILIARITY' ANSWERS)

	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
Rising Head Test	5 -	5 -		- -	- -	1 1	- -	- -	- -	- -	- 1	2 3	- -	- -	- -	- -		1 1	1 2	3 2	3 2	3 2	- 3	- 2
	- -	-		1 3	- 4	- 2	- 4	- 4	- 4	- 4	- 3	- -	- 4	- 4	- 4	- 4		1 2	- 2	- -	- -	- -	1 1	2 1
Falling Head Test	6 -	6 -		- -	- 1	- 1	- -	- -	- -	- -	- 1	2 3	- -	- -	- -	- -		1 1	1 3	4 2	5 1	3 3	- 2	- 2
	- -	-		1 4	- 4	- 4	- 5	- 5	- 5	- 5	- 4	- -	- 5	- 5	- 5	- 5		2 2	- 2	- -	- -	- -	3 1	2 2
Constant Head Test	4 -	4 -		- -	- -	- 1	- -	- -	- -	- -	- -	1 3	- -	- -	- -	- -		1 1	1 1	1 2	2 2	2 2	1 -	1 1
	--	-		- 3	- 3	- 2	- 3	- 3	- 3	- 3	- 3	- -	- 3	- 3	- 3	- 3		- 2	- 2	1 -	- -	- -	2 1	1 1
Self-boring Permeameter Test	- -	- -		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -
	- -	-		- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -		- -	- -	- -	- -	- -	- -	- -
Pumping Tests	2 -	- 2		- -	- -	1 -	- -	- -	- -	- -	- -	2 -	- -	- -	- -	- -		1 1	1 1	2 -	2 -	1 1	- 1	- 1
	- -	-		- 1	- 1	- -	- 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1	- 1		- -	- -	- -	- -	- -	- 1	- 1

KEY	H M L N	R L S		H M L N	

Table E.17 Summation of the 'high familiarity' answers obtained for the Permeability Tests

PERMEABILITY TESTS (AVERAGES)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]		GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (ϕ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (m_v, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE		GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	R	Rising Head Test		N	N	L	N	N	N	N	N	M	N	N	N	N			L	L	M	H	M	L	L
	R			N	N	L	N	N	N	N	N	M	N	N	N	N			L	L	H	H	H	L	L
	R	Falling Head Test		N	N	L	N	N	N	N	N	M	N	N	N	N			L	L	M	H	M	L	L
	R			N	N	N	N	N	N	N	N	M	N	N	N	N			L	L	H	H	M	L	L
	R	Constant Head Test		N	N	L	N	N	N	N	L	M	N	N	N	N			L	L	M	M	M	M	L
	R			N	N	L	N	N	N	N	N	M	N	N	N	N			L	L	M	M	M	L	L
	S	Self-boring Permeameter Test		N	N	L	N	N	N	N	N	H	N	N	N	N			N	L	N	M	H	M	L
	-			-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	Pumping Tests		N	N	L	N	N	N	N	N	H	N	N	N	N			M	M	H	H	M	L	L
	L			N	N	H	N	N	N	N	N	H	N	N	N	N			M	M	H	H	M	L	L

KEY			
A_I	Average value taking into account all answers	A	Average value; all answers are 'high familiarity' answers
A_{II}	Average value taking into account 'high familiarity' answers		

Table E.18 Average values calculated for the Permeability Tests

GEOPHYSICAL SURVEYING TESTS (ALL ANSWERS)

	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (Cv, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
Seismic Refraction Test	23	22		-1	21	--	--	--	-3	--	--	--	-2	--	--	--		2-	2-	3-	3-	3-	3-	1-
	-6	-		21	-1	-4	13	13	-1	22	-4	13	11	-4	-4	-4		--	--	--	--	--	--	1-
Seismic ReflectionTest	21	11		-1	2-	--	--	--	-3	--	--	--	-2	--	--	--		2-	2-	3-	21	21	21	1-
	17	1		11	-1	-3	12	-3	--	21	-3	12	1-	-3	-3	-3		--	--	--	--	--	--	1-
Seismic Cross-Hole Test	21	-2		--	-1	--	--	--	--	--	--	--	2-	--	--	--		3-	21	2-	2-	2-	2-	2-
	-8	1		2-	2-	-2	-2	-2	11	-2	-2	-2	1-	-2	-2	-2		--	--	1-	1-	1-	1-	-1
Seismic Down-Hole Test	21	-2		-1	-2	--	--	--	-1	--	--	--	2-	--	--	--		3-	3-	21	21	21	21	2-
	-8	1		1-	1-	-2	-2	-2	-1	-1	-1	-1	1-	-2	-2	-2		--	--	--	--	--	--	-1
Surface Wave Test	1-	--		--	--	--	--	--	--	--	--	--	1-	--	--	--		1-	1-	1-	1-	1-	1-	1-
	-10	1		1-	1-	-1	-1	-1	-1	-1	-1	-1	--	-1	-1	-1		--	--	--	--	--	--	--
Resistivity Test	-3	11		-1	11	-1	--	--	--	11	--	--	--	--	--	--		1-	11	21	21	21	21	11
	-8	-		11	-1	11	-3	-3	-3	-1	12	-3	-3	-3	-3	-3		-1	--	--	--	--	--	--
Gravimetric Test	--	-1		-1	11	--	--	--	11	--	--	--	--	--	--	--		1-	1-	1-	1-	1-	1-	1-
	38	1		-2	-1	-3	-3	-3	-1	12	-3	-3	-3	-3	-3	-3		--	--	--	--	--	--	--
Magnetic Test	-1	-2		-1	11	--	--	-1	-1	--	--	--	--	--	--	--		1-	1-	1-	1-	1-	1-	1-
	28	-		-2	-1	12	-3	-2	11	-3	-3	-3	-3	-3	-3	-3		--	--	--	--	--	--	--

KEY	HM	RL		HM		HM
	LN	S		LN		LN

Table E.19 Summation of all the answers obtained for the Geophysical Surveying Tests

GEOPHYSICAL SURVEYING TESTS ('HIGH FAMILIARITY' ANSWERS)

GEOTECHNICAL INFORMATION [H, M, L, N]	FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (mv, Cc)	RATE OF CONSOLIDATION (Cv, Ch)	PERMEABILITY (K)	MODULUS; SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE
Seismic Refraction Test	2 -	1 -	- -	1 -	- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -
	- -	-	1 -	- -	- 1	- 1	- 1	- -	1 -	- 1	1 -	1 -	- 1	- 1	- 1
Seismic ReflectionTest	2 -	- 1	- -	1 -	- -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -
	- -	-	1 -	- -	- 1	- 1	- 1	- -	1 -	- 1	1 -	1 -	- 1	- 1	- 1
Seismic Cross-Hole Test	2 -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -	2 -	- -	- -	- -
	- -	1	1 -	1 -	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1
Seismic Down-Hole Test	2 -	- -	- -	- 1	- -	- -	- -	- -	- -	- -	- -	2 -	- -	- -	- -
	- -	1	1 -	1 -	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1
Surface Wave Test	1 -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	1 -	- -	- -	- -
	- -	-	1 -	1 -	- 1	- 1	- 1	- 1	- 1	- 1	- 1	- -	- 1	- 1	- 1
Resistivity Test	1 -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	- -	-	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
Gravimetric Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	- -	-	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
Magnetic Test	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
	- -	-	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -	- -
KEY	H M L N	R L S		H M L N											

GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	1 -	1 -	2 -	2 -	2 -	2 -	- -
	- -	- -	- -	- -	- -	- -	1 -
	1 -	1 -	2 -	1 1	1 1	1 1	- -
	- -	- -	- -	- -	- -	- -	1 -
	2 -	2 -	2 -	2 -	2 -	2 -	2 -
	- -	- -	- -	- -	- -	- -	- -
	2 -	2 -	2 -	2 -	2 -	2 -	2 -
	- -	- -	- -	- -	- -	- -	- -
	1 -	1 -	1 -	1 -	1 -	1 -	1 -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	1 -	1 -	1 -	1 -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
	- -	- -	- -	- -	- -	- -	- -
KEY	H M L N						

Table E.20 Summation of the 'high familiarity' answers obtained for the Geophysical Surveying Tests

GEOPHYSICAL SURVEYING TESTS (AVERAGES)

FAMILIARITY WITH TEST [H, M, L, N]	TEST FREQUENCY [R, L, S]	GEOTECHNICAL INFORMATION [H, M, L, N]	SOIL TYPE	PROFILE	PIEZOMETRIC PRESSURE (u)	ANGLE OF FRICTION (φ)	UNDRAINED SHEAR STRENGTH (S_u)	DENSITY (D_r)	COMPRESSIBILITY (mv, C_c)	RATE OF CONSOLIDATION (C_v, C_h)	PERMEABILITY (K)	MODULUS: SHEAR & YOUNG'S (G,E)	IN SITU STRESS (K_o)	STRESS HISTORY (OCR)	STRESS STRAIN CURVE	GROUND CONDITIONS [H, M, L, N]	HARD ROCK	SOFT ROCK - TILL, etc	GRAVEL	SAND	SILT	CLAY	PEAT - ORGANICS
	L	Seismic Refraction Test	L	M	N	N	N	L	N	N	N	L	N	N	N			H	H	H	H	H	H	M
	R		L	H	N	N	N	M	L	N	L	L	N	N	N			H	H	H	H	H	H	L
	L	Seismic ReflectionTest	L	M	N	N	N	M	L	N	N	M	N	N	N			H	H	H	H	H	H	M
	L		L	H	N	N	N	M	L	N	L	L	N	N	N			H	H	H	M	M	M	L
	L	Seismic Cross-Hole Test	L	L	N	N	N	N	N	N	N	M	N	N	N			H	H	M	M	M	M	M
	S		L	L	N	N	N	N	N	N	N	H	N	N	N			H	H	H	H	H	H	H
	L	Seismic Down-Hole Test	L	M	N	N	N	L	N	N	N	M	N	N	N			H	H	H	H	H	H	M
	S		L	L	N	N	N	N	N	N	N	H	N	N	N			H	H	H	H	H	H	H
	S	Surface Wave Tests	L	L	N	N	N	N	N	N	N	H	N	N	N			H	H	H	H	H	H	H
	-		L	L	N	N	N	N	N	N	N	H	N	N	N			H	H	H	H	H	H	H
	L	Resistivity Test	L	M	L	N	N	L	N	N	N	N	N	N	N			L	M	H	H	H	H	M
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	H	H	H	H	-
	S	Gravimetric Test	L	M	N	N	N	M	N	N	N	N	N	N	N			H	H	H	H	H	H	H
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-
	L	Magnetic Test	L	M	N	N	L	L	N	N	N	N	N	N	N			H	H	H	H	H	H	H
	-		-	-	-	-	-	-	-	-	-	-	-	-	-			-	-	-	-	-	-	-

KEY			
A_I	Average value taking into account all answers	A	Average value; all answers are 'high familiarity' answers
A_{II}	Average value taking into account 'high familiarity' answers		

Table E.21 Average values calculated for the Geophysical Surveying Tests

APPENDIX F

PROKAPPA PROGRAM

```
/*************************
*    File REPRESENT.APP    *
*************************/
```

```
#PrkDefn ProKappa : $Revision: 3.132 $
#
# Definition for: Represent
#
  Application: Represent

  CFiles            =
  ProTalkFiles      =
  ProTalkCompileFlags =
  LoadFlags         =
  ObjectBase        = :Represent.ob
  UserModules       = RepresentUI
  RequiredModules   = DialogBoxApp
  AboutAppFile      =
  AfterLoadInitFnName =
  RunFnName         =
#
#
#
  Module: RepresentUI

  CFiles            =
  ProTalkFiles      = :GRinit.ptk,:GRfunc1.ptk,:GRfunc2.ptk,
                      :GRfunc3.ptk,:GRfunc4.ptk,:GRmisc.ptk
  ProTalkCompileFlags =
  LoadFlags         =
  ObjectBase        = :RepresentUI.ob
  UserModules       =
  RequiredModules   = DialogBoxApp
```

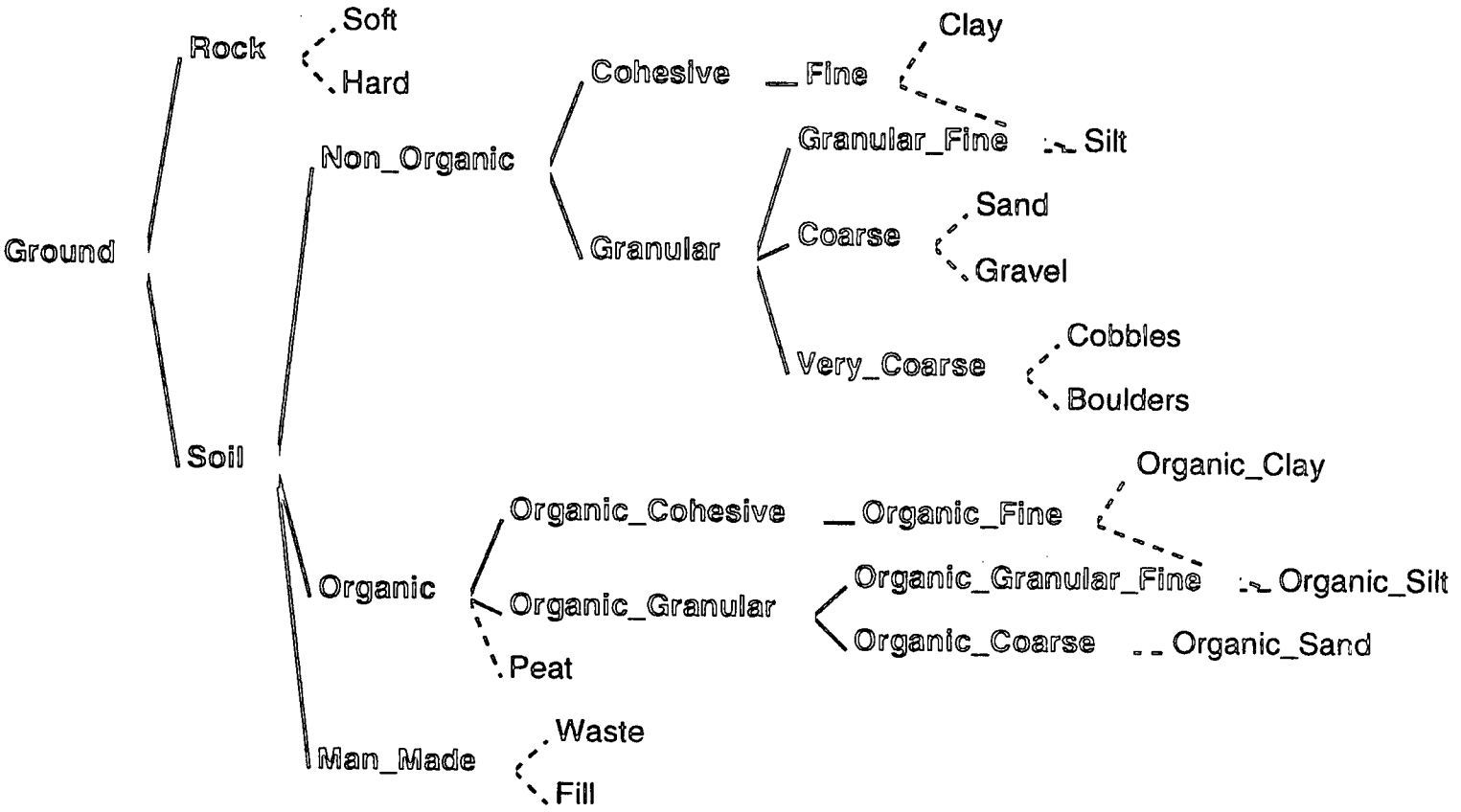


Figure F.1 The Ground hierarchy included in the object base developed for the 'Representing the Ground' application in the PROKAPPA system

```
/**********************
*     File GRINIT.PTK     *
**********************/
```

/ This file contains the functions required for the construction of the dialog boxes and their controls, and for the activation of the interface. */*

```
#include <prk/lib.pth>


function MakeInterfaceElements()

{

   bound inputs;
   MakeDialogBox(RepresentUI, Main_Menu_Box);
   MakeDialogBox(RepresentUI, Func_Box);
   MakeDialogBox(RepresentUI, Display_Box);
   MakeDialogBoxControl(ListBox, RepresentUI, LM);
   MakeDialogBoxControl(ListBox, RepresentUI, L1);
   MakeDialogBoxControl(ListBox, RepresentUI, L2);
   MakeDialogBoxControl(CommandRow, RepresentUI, C1);
   MakeDialogBoxControl(CommandRow, RepresentUI, C2);
   MakeDialogBoxControl(CommandRow, RepresentUI, C3);
   MakeDialogBoxControl(EntryBox, RepresentUI, E1);
   MakeDialogBoxControl(PushButton, RepresentUI, P1);
   MakeDialogBoxControl(PushButton, RepresentUI, P2);
   MakeDialogBoxControl(TextDisplay, RepresentUI, OP1);
}


function MainMenu()

{
   bound inputs;
   GetSlotList();
   FindSlotRangeList();
   ListObjMods();
   SetDialogBoxControls(Main_Menu_Box, `(LM@, C1@));
   SetDialogBoxControlValues(Main_Menu_Box, LM, SelectionItems, `("List Ancestors", "List Slots",
"Find Object Modifiers", "Find Objects and Modifiers", "Exit"));
   SetDialogBoxControlValue(Main_Menu_Box, LM, MaxNumOfLines, 5);
   SetDialogBoxControlValue(Main_Menu_Box, C1, React!, `?C1.React!);
   Main_Menu_Box.Title = "Function Menu";
   LM.Title = "Please choose one of the following:";
 C:PrkSendMsg(Main_Menu_Box@, `PutOnScreenAndWait!);
}
```

```
method C1.React! ()

{
  bound inputs;

  ?menu_item = GetDialogBoxControlValue(Main_Menu_Box, LM, Values);
  select
  {
   case:?menu_item == "List Ancestors";
     {
      C:PrkSendMsg(Main_Menu_Box@, `TakeOffScreen!);
        MakeAncestors();
     }
   case:?menu_item == "List Slots";
     {
      C:PrkSendMsg(Main_Menu_Box@, `TakeOffScreen!);
        MakeSlots();
     }
   case:?menu_item == "Find Object Modifiers";
     {
      C:PrkSendMsg(Main_Menu_Box@, `TakeOffScreen!);
        MakeMods();
     }
   case:?menu_item == "Find Objects and Modifiers";
     {
      C:PrkSendMsg(Main_Menu_Box@, `TakeOffScreen!);
        MakeObjMods();
     }
  case:?menu_item == "Exit";
     {
      C:PrkSendMsg(Main_Menu_Box@, `TakeOffScreen!);
        fail;
     }
  }
}
```

```
/*************************
*     File GRFUNC1.PTK     *
*************************/
```

/ This file contains the functions that are activated on selecting the "List Ancestors" option from the Function Menu dialog box in order to identify and display the ancestors of an object within the hierarchy. */*

```
#include <prk/lib.pth>


function MakeAncestors()

{
  SetDialogBoxControls(Func_Box, `(L1@, P1@,C2@));
  ?result = ListObjs(Ground);
  SetDialogBoxControlValues(Func_Box, L1, SelectionItems, ?result);
  SetDialogBoxControlValue(Func_Box, L1, MaxNumOfLines, 15);
  SetDialogBoxControlValue(Func_Box, C2, React!, `?C2Ancestors.React!);
  SetDialogBoxControlValue(Func_Box, P1, React!, `?P1Ancestors.React!);
  SetDialogBoxControlValue(Func_Box, C2, ButtonLabels, "Cancel");
  P1.ButtonLabel = "Push";
  P1.Title = "Display Ancestors";
  L1.Title = "Please choose one of the following:\n\n\n[ All objects in model listed ]";
  Func_Box.Title = "List Ancestors Function";
 C:PrkSendMsg(Func_Box@, `PutOnScreen!);
}


method C2Ancestors.React!()

{
 C:PrkSendMsg(Display_Box@, `TakeOffScreen!);
 C:PrkSendMsg(Func_Box@, `TakeOffScreen!);
 C:PrkSendMsg(Main_Menu_Box@, `PutOnScreen!);
}


method P1Ancestors.React!()

{
  MakeAncestorsDisplay();
 C:PrkSendMsg(Display_Box@, `PutOnScreen!);
}
```

```
function MakeAncestorsDisplay()

{
   SetDialogBoxControls(Display_Box, `(OP1@, C3@));
   ?selection = ConvertToSymbol(GetDialogBoxControlValue(Func_Box, L1, Values));
   ?ancestors = FindAncestors(?selection);
   Display_Box.Title = "Display Ancestors";
   SetDialogBoxControlValue(Display_Box, Display_Box, PositionX, 715);
   SetDialogBoxControlValue(Display_Box, Display_Box, PositionY, 634);
   SetDialogBoxControlValues(Display_Box, OP1, Values, ?ancestors);
   SetDialogBoxControlValue(Display_Box, C3, React!, `?C3.React!);
   SetDialogBoxControlValue(Display_Box, C3, ButtonLabels, "OK");
}


function FindAncestors(?obj)

{
   bound inputs;
   if IsInstance(?obj);
   then ?ancestors = all classof ?obj;
   else ?ancestors = all superclassof ?obj;
   return ?ancestors;

}


method C3.React!()

{
 C:PrkSendMsg(Display_Box@, `TakeOffScreen!);
}
```

```
/*************************
*     File GRFUNC2.PTK     *
*************************/
```

/ This file contains the functions that are activated on selecting the "List Slots" option from the Function Menu dialog box in order to find the attributes of an object within the hierarchy and the values that these attributes have. */*

```
#include <prk/lib.pth>


function MakeSlots()

{
   SetDialogBoxControls(Func_Box, `(L1@, P1@, L2@, P2@, C2@));
   ?result = calc.ObjSlots;
   SetDialogBoxControlValues(Func_Box, L1, SelectionItems, ?result);
   SetDialogBoxControlValue(Func_Box, L1, MaxNumOfLines, 15);
   SetDialogBoxControlValues(Func_Box, L2, SelectionItems, `());
   SetDialogBoxControlValue(Func_Box, C2, React!, `?C2Slots.React!);
   SetDialogBoxControlValue(Func_Box, C2, ButtonLabels, "Cancel");
   Func_Box.Title = "List Slots Function";
   P1.ButtonLabel = "Push";
   P2.ButtonLabel = "Push";
   P1.Title = "List Attributes";
   P2.Title = "Display Values";
   L1.Title = "Please choose one of the following:\n\n\n[ Only those objects having slots listed ]";
   L2.Title = "Attribute Table";
   SetDialogBoxControlValue(Func_Box, P1, React!, `?P1Slots.React!);
   SetDialogBoxControlValue(Func_Box, P2, React!, `?P2Slots.React!);
 C:PrkSendMsg(Func_Box@, `PutOnScreenAndWait!);
}


method C2Slots.React!()

{
    C:PrkSendMsg(Display_Box@, `TakeOffScreen!);
    C:PrkSendMsg(Func_Box@, `TakeOffScreen!);
    C:PrkSendMsg(Main_Menu_Box@, `PutOnScreen!);
}
```

```
method P1Slots.React!()

{
  ?selection = ConvertToSymbol(GetDialogBoxControlValue(Func_Box, L1, Values));
  ?slot_list = ObjectSlots(?selection);
  SetDialogBoxControlValues(Func_Box, L2, SelectionItems, ?slot_list);
}


method P2Slots.React!()

{
  ?obj = ConvertToSymbol(GetDialogBoxControlValue(Func_Box, L1, Values));
  ?slot = ConvertToSymbol(GetDialogBoxControlValue(Func_Box, L2, Values));
  ?val = GetValues(?obj, ?slot);
  ?min = ListFirst(?val);
  ?max = ListNth(?val, 1);
  Display_Box.Title = "Display Slot Values";
  ?ans = AppendStrings(ConvertToString(?slot),"\n\nMin Value = ",ConvertToString(?min),"\nMax
Value = ",ConvertToString(?max));
  SetDialogBoxControlValue(Display_Box, OP1, Values, ?ans);
  SetDialogBoxControlValue(Display_Box, Display_Box, PositionX, 731);
  SetDialogBoxControlValue(Display_Box, Display_Box, PositionY, 697);
  SetDialogBoxControlValue(Display_Box, C3,ButtonLabels, "OK");
 C:PrkSendMsg(Display_Box@, `PutOnScreen!);
}
```

```
/*************************
*     File GRFUNC3.PTK     *
*************************/
```

/ This file contains the functions that are activated on selecting the "Find Object Modifiers" option from the Function Menu dialog box in order to find the modifiers that correspond to a value in a slot in an object within the hierarchy. */*

```
#include <prk/lib.pth>


function MakeMods()

{
  SetDialogBoxControls(Func_Box, `(L1@, P1@, L2@, E1@, P2@, C2@));
  ?result = calc.ObjMods;
  SetDialogBoxControlValues(Func_Box, L1, SelectionItems, ?result);
  SetDialogBoxControlValue(Func_Box, L1, MaxNumOfLines, 7);
  SetDialogBoxControlValue(Func_Box, C2, React!, `?C2Mods.React!);
  SetDialogBoxControlValues(Func_Box, L2, SelectionItems, `());
  SetDialogBoxControlValue(Func_Box, C2, ButtonLabels, "Cancel");
  Func_Box.Title = "Find Object Modifiers Function";
  P1.ButtonLabel = "Push";
  P2.ButtonLabel = "Push";
  P1.Title = "List Attributes";
  P2.Title = "Display Results";
  L1.Title = "Please choose one of the following:\n\n[Only objects having attributes\n with defined
modifiers listed]";
  L2.Title = "Attribute Table\n\n[Only those attributes with\ndefined modifiers listed]";
  E1.Title = "Enter value for chosen attribute\n";
  SetDialogBoxControlValue(Func_Box, P1, React!, `?P1Mods.React!);
  SetDialogBoxControlValue(Func_Box, P2, React!, `?P2Mods.React!);
 C:PrkSendMsg(Func_Box@, `PutOnScreenAndWait!);
}


method C2Mods.React!()

{
   C:PrkSendMsg(Display_Box@, `TakeOffScreen!);
   C:PrkSendMsg(Func_Box@, `TakeOffScreen!);
   C:PrkSendMsg(Main_Menu_Box@, `PutOnScreen!);
}
```

```
method P1Mods.React!()

{
   ?range_list = `();
   ?selection = ConvertToSymbol(GetDialogBoxControlValue(Func_Box, L1, Values));
   ?slot_list = CheckSlots(?selection);
   for ?list_mem inlist ?slot_list;
   do {
         ?slot_values = GetValues(?selection, ?list_mem);
         ?list_first = ConvertToString(ListFirst(?slot_values));
         ?list_last = ConvertToString(ListNth(?slot_values, 1));
         ?list_mem = AppendStrings(ConvertToString(?list_mem), " [",?list_first, ",",?list_last, "]");
         ?list_mem = `(?list_mem);
         append ?list_mem into ?range_list;
   }
   SetDialogBoxControlValues(Func_Box, L2, SelectionItems, ?range_list);
}


method P2Mods.React!()

{
   ?obj = ConvertToSymbol(GetDialogBoxControlValue(Func_Box, L1, Values));
   ?slot_str = ConvertToString(GetDialogBoxControlValue(Func_Box, L2, Values));
   ?f_val = ConvertToNumber(GetDialogBoxControlValue(Func_Box, E1, Values));
   ?loc = FindSubstring(?slot_str, "[");
   ?slot_name = ConvertToSymbol(Substring(?slot_str, 0, ?loc-1));
   ValidFacVal(?obj, ?slot_name, ?f_val);
}


function CheckSlots(?sel_obj)

{
   bound inputs;
   ?ret_slot_list = `();
   ?slot_list = ObjectSlots(?sel_obj);
   for ?slot_name inlist ?slot_list;
   do {
         ?facet_list = SlotFacets(?sel_obj, ?slot_name);
         if ListLength(?facet_list) > 0;
         then {
            ?slot_name = `(?slot_name);
            append ?slot_name into ?ret_slot_list;
         }
   }
   return ?ret_slot_list;
}
```

```
function ValidFacVal(?obj, ?slot_name, ?f_val)

{
   bound inputs;
   ?slot_range = GetValues(?obj, ?slot_name);
   ?min = ListFirst(?slot_range);
   ?max = ListNth(?slot_range, 1);
   if
   {
         IsNumber(?f_val);
         ?f_val<=?max;
         ?f_val>=?min;
   }
   then MakeModsDisplay(?obj, ?slot_name, ?f_val);
   else {
         ?mesg = AppendStrings(ConvertToString(?slot_name), " must be a number between the range
[", ConvertToString(?min), ", ",ConvertToString(?max), "] for ", ConvertToString(?obj));
         SetDialogBoxControlValue(Error_Box, TE, Values, ?mesg);
         SetDialogBoxControlValue(Error_Box, CE, ButtonLabels, "OK");
         SetDialogBoxControlValue(Error_Box, CE, React!, `?CEValid.React!);
    C:PrkSendMsg(Error_Box@, `PutOnScreenAndWait!);
   }
}


function MakeModsDisplay(?obj, ?slot_name, ?f_val)

{
   bound inputs;
   ?ans_list =  FindFacets(?obj, ?slot_name, ?f_val);
   Display_Box.Title = "Display Object Modifiers";
   SetDialogBoxControlValues(Display_Box, OP1, Values, ?ans_list);
   SetDialogBoxControlValue(Display_Box, Display_Box, PositionX, 690);
   SetDialogBoxControlValue(Display_Box, Display_Box, PositionY, 647);
   SetDialogBoxControlValue(Display_Box, C3, ButtonLabels, "OK");

 C:PrkSendMsg(Display_Box@, `PutOnScreen!);
}


method CEValid.React!()
{
   bound inputs;
   Error_Box.TakeOffScreen!();
}
```

```
function FindFacets(?obj, ?slt, ?f_val)
{
   bound inputs;
   ?ans_list = `();
   ?facet_list = SlotFacets(?obj, ?slt);
   for ?facet_name inlist ?facet_list;
   do
   {
         ?f_vallist = GetFacetValues(?obj, ?slt, ?facet_name);
         ?min = ListFirst(?f_vallist);
         ?max = ListNth(?f_vallist, 1);
         if
         {
            ?f_val >= ?min;
            ?f_val <= ?max;
         }
         then
      {
            ?ans = AppendStrings(ConvertToString(?facet_name)," ", ConvertToString(?obj));
            collect ?ans into ?ans_list;
         }
   }
   return ?ans_list;
}
```

```
/*************************
*     File GRFUNC4.PTK     *
*************************/
```

/ This file contains the functions that are activated on selecting the "Find Objects and Modifiers" option from the Function Menu dialog box in order to find the objects and modifiers (if any) that correspond to a value in a slot within the hierarchy. */*

```
#include <prk/lib.pth>


function MakeObjMods()

{
  SetDialogBoxControls(Func_Box, `(L1@, E1@, P1@, C2@));
  SetDialogBoxControlValues(Func_Box, L1, SelectionItems, `calc.Attr_Range);
  SetDialogBoxControlValue(Func_Box, L1, MaxNumOfLines, 3);
  SetDialogBoxControlValue(Func_Box, C2, ButtonLabels, "Cancel");
  SetDialogBoxControlValue(Func_Box, C2, React!, `?C2ObjMods.React!);
  Func_Box.Title = "Find Objects and Modifiers Function";
  L1.Title = "Please choose one of the following: [ All attributes of model listed ]";
  P1.ButtonLabel = "Push";
  P1.Title = "Display Results";
  E1.Title = "Enter value for chosen attribute\n";
  SetDialogBoxControlValue(Func_Box, P1, React!, `?P1ObjMods.React!);
 C:PrkSendMsg(Func_Box@, `PutOnScreenAndWait!);
}


method C2ObjMods.React!()

{
 C:PrkSendMsg(Display_Box@, `TakeOffScreen!);
 C:PrkSendMsg(Func_Box@, `TakeOffScreen!);
 C:PrkSendMsg(Main_Menu_Box@, `PutOnScreen!);
}


method P1ObjMods.React!()

{
  ?obj = ConvertToSymbol("Ground");
  ?slot_string = ConvertToSymbol(GetDialogBoxControlValue(Func_Box, L1, Values));
  ?slot_val = ConvertToNumber(GetDialogBoxControlValue(Func_Box, E1, Values));
  ?loc = FindSubstring(?slot_string, "[");
  ?slot_name = ConvertToSymbol(Substring(?slot_string, 0, ?loc-1));
  ValidSlotVal(?obj, ?slot_name, ?slot_string, ?slot_val);
}
```

```
function ValidSlotVal(?obj, ?slot_name, ?slot_string, ?slot_val)

{
   bound inputs;
   ?loc1 = FindSubstring(?slot_string, "[");
   ?loc2 = FindSubstring(?slot_string, ",");
   ?loc3 = FindSubstring(?slot_string, "]");
   ?min = ConvertToNumber(Substring(?slot_string, ?loc1+1, ?loc2));
   ?max = ConvertToNumber(Substring(?slot_string, ?loc2+2, ?loc3));
   if
   (
         IsNumber(?slot_val);
         ?slot_val<=?max;
         ?slot_val>=?min;
    )
   then {
         SetDialogBoxControlValue(Display_Box, OP1, Values, `());
         FindObjectsAndFacets(?obj, ?slot_name, ?slot_val);
         ?ans_list = GetDialogBoxControlValues(Display_Box, OP1, Values);
         SetDialogBoxControlValues(Display_Box, OP1, Values, ListRest(?ans_list));
         Display_Box.Title = "Display Objects and  Modifiers";
         SetDialogBoxControlValue(Display_Box, Display_Box, PositionX, 688);
         SetDialogBoxControlValue(Display_Box, Display_Box, PositionY, 550);
         SetDialogBoxControlValue(Display_Box, C3, ButtonLabels, "OK");
    C:PrkSendMsg(Display_Box@, `PutOnScreen!);
   }
   else {
         ?mesg = AppendStrings(ConvertToString(?slot_name), " must be a number between the range
[", ConvertToString(?min), ", ",ConvertToString(?max), "]");
         SetDialogBoxControlValue(Error_Box, TE, Values, ?mesg);
         SetDialogBoxControlValue(Error_Box, CE, ButtonLabels, "OK");
         SetDialogBoxControlValue(Error_Box, CE, React!, `?CEValid.React!);
    C:PrkSendMsg(Error_Box@, `PutOnScreenAndWait!);
   }
}
```

```
function FindObjectsAndFacets(?start, ?attr, ?attr_val)

{
  bound inputs;
  for find ?obj = direct subclassof ?start;
  do
  {
        Checker(?obj, ?attr, ?attr_val);
        ?ans = CheckPath(?obj, ?attr, ?attr_val);
        ?ans == 1;
        FindObjectsAndFacets(?obj, ?attr, ?attr_val);
  }
  for find ?inst = direct instanceof ?start;
  do
  {
        ?ans = CheckPath(?inst, ?attr, ?attr_val);
        ?ans == 1;
        SearchForModifiers(?inst, ?attr, ?attr_val);
  }
}


function Checker(?obj, ?attr, ?attr_val)

{
  bound inputs;
  if not IsSlot(?obj, ?attr);
  then
  {
        find ?name = direct subclassof ?obj;
        FindObjectsAndFacets(?name, ?attr, ?attr_val);
  }
  else
  {
        succeed;
  }
}


function CheckPath(?obj, ?attr, ?attr_val)

{
  bound inputs;
  if IsSlot(?obj, ?attr);
  then {?values = GetValues(?obj, ?attr);
        ?min = ListFirst(?values);
        ?max = ListNth(?values, 1);
        if {?attr_val >= ?min;
                ?attr_val <= ?max;}
        then return 1;
            else return 0;
      }
  else return 0;
}
```

```
function SearchForModifiers(?obj, ?attr, ?attr_val)

{
   bound inputs;
   ?facet_list = SlotFacets(?obj, ?attr);
   if ListLength(?facet_list) == 0;
   then {
         ?obj = ConvertToString(?obj);
         ?attr = ConvertToString(?attr);
         ?attr_val = ConvertToString(?attr_val);
         ?new_value = `(AppendStrings( ?obj,"   [ No modifiers defined ]"));
         SetNewValue(?new_value);
   }
   else {
         for ?facet_name inlist ?facet_list;
         do
         {
            ?facet_values = GetFacetValues(?obj, ?attr, ?facet_name);
            ?min = ListFirst(?facet_values);
            ?max = ListNth(?facet_values, 1);
            if
            {
                  ?attr_val >= ?min;
                  ?attr_val <= ?max;
            }
            then
            {
                  ?new_value = `(AppendStrings("        ", ConvertToString(?facet_name)," ",
ConvertToString(?obj),"        "));
                  SetNewValue(?new_value);
            }
         }
   }
}



function SetNewValue(?new_value)

{
   bound inputs;
   ?exist_values= GetDialogBoxControlValues(Display_Box, OP1, Values);
   append ?exist_values into ?ans_list;
   append ?new_value into ?ans_list;
   SetDialogBoxControlValues(Display_Box, OP1, Values, ?ans_list);
}
```

```
/************************
*      File GRMISC.PTK     *
************************/
```

/ This file contains miscellaneous functions. */*

```
#include <prk/lib.pth>
#include <prk/math.pth>


function ListObjs(?start)

{
   bound inputs;
   ?cls = all subclassof ?start;
   ?inst = all instanceof ?start;
   append ?cls into ?full_list;
   append ?inst into ?full_list;
   return ?full_list;
}



function IsListMember(?main_list, ?member)

{
   bound inputs;
   for ?x inlist ?main_list;
   do
   {
         if ?x == ?member;
         then return 0;
   }
   return 1;
}
```

```
function ListObjMods()

{
  ?ret_obj_list = `();
  ?obj_list=ListObjs(Ground);
  for ?mem_list inlist ?obj_list;
  do
  {
        ?cur_slot_list=ObjectSlots(?mem_list);
        if ?cur_slot_list != `();
        then append `(?mem_list) into ?sel_obj_list;
        calc.ObjSlots = ?sel_obj_list;
  }
  for ?sel_obj inlist ?sel_obj_list;
  do
  {
        ?test = CheckFacets(?sel_obj);
        if ?test == 1;
        then append `(?sel_obj) into ?ret_obj_list;
  }
  calc.ObjMods = ?ret_obj_list;
}


function CheckFacets(?obj)

{
  bound inputs;
  ?slot_list = ObjectSlots(?obj);
  for ?slot_name inlist ?slot_list;
  do
  {
        ?facet_list = SlotFacets(?obj,?slot_name);
        if ListLength(?facet_list) > 0;
        then return 1;
  }
  return 0;
}
```

```
function GetSlotList()

{
  ?obj_list=ListObjs(Ground);
  ?main_slot_list = `();
  for ?obj inlist ?obj_list;
  do
  {
        ?cur_slot_list=ObjectSlots(?obj);
        if ?cur_slot_list != `();
        then
        {
          ?accum_list = GetUniqueSlots(?main_slot_list, ?cur_slot_list);
          ?main_slot_list = ?accum_list;
        }
  }
  calc.Main_slot_list = ?accum_list;
}


function GetUniqueSlots(?main_slot_list, ?cur_slot_list)

{

  bound inputs;
  append ?main_slot_list into ?temp_list;
  for ?attr inlist ?cur_slot_list;
  do
  {
        ?test = IsListMember(?temp_list, ?attr);
        if ?test == 1;
        then append `(?attr) into ?temp_list;
  }
  return ?temp_list;
}


function FindSlotRangeList()

{
  ?full_range_list = `();
  ?main_slot_list = calc.Main_slot_list;
  for ?slot_name inlist ?main_slot_list;
  do
  {
        ?slot_range = FindSlotRange(Represent, RepresentUI, ?slot_name);
        append `(?slot_range) into ?slot_range_list;
  }
  calc.Attr_Range = ?slot_range_list;
}
```

```
function FindSlotRange(?app,?mod, ?attr)

{
  bound inputs;
  ?val_list = `();
  ?app_inst_list = AppInstances(?app);
  ?mod_inst_list=ModuleInstances(?mod);
  for
  {
        ?list_mem inlist ?app_inst_list;
        ?list_mem inlist ?mod_inst_list;
  }
  do DeleteListElmt(?list_mem, ?app_inst_list);
  for ?list_mem inlist ?app_inst_list;
  do
  {
        if IsSlot(?list_mem, ?attr);
        then
        {
          ?slot_vals = GetValues(?list_mem, ?attr);
          append ?slot_vals into ?val_list;
        }
  }
  for ?list_mem inlist ?val_list;
  do
  {
        ConvertToFloat(?list_mem);
        append `(?list_mem) into ?num_list;
  }
  ?num_list = Sort(?num_list, ">");
  ?min = ConvertToString(ListFirst(?num_list));
  ?max = ConvertToString(ListFirst(ListLastCons(?num_list)));
  ?attr_range = AppendStrings(ConvertToString(?attr)," [",?min,", ",?max,"]");
  return ?attr_range;
}
```