CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

Measurement Planning System for Information Integrated Construction

by

Toshihiko Aoki

TECHNICAL REPORT
Number 53

July, 1991

Stanford University

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 95305-4020

Contents

1. Introduction

In Japan owing to the high-growing economy, the land problem has become serious. Hence each major general construction company has been trying to accomplish three big projects. These are development projects of sea, ground and sky space in order to extend our living space. One of them is called GEO-FRONT. This is a project digging into the ground to make large underground towns. The GEO-FRONT project is so big to tackle that many kinds of obstacle must be faced .

The uunderground is an nknown world because the ground has many uncertainties. Ground is not homogeneous and is composed of of many kinds of materials such as water, slip surfaces, soil and so on. The uncertainties of the ground is apparently wider than other man made materials such as concrete and steel. These uncertainties can't be completely considered in the design phase. These are serious problems underlaying on the base of geotechnical engineering.

In order to analyze the response of the ground, experimental equations and experienced relations have been used for a long time, and still are relied on. Even though it is small part of the ground, to predict or analyze it using results of ground survey or soil tests is very difficult because of gaps that exist between assumptions of analysis techniques, such as Finite Element Method (FEM), and that of soil tests. It is much more the case that for larger or deeper areas that we have never been to, big problems will occur immediately. For instance, the mechanism of earth-pressure in deep ground is not still clear. Therefore if some theories of earth-pressure, which are based on the our experience of shallow areas, are used to calculate the thickness of tunnel wall, additional experiments or discussion would be needed.

In these days, the technique of ground numerical analysis is going further than that of monitoring, survey and soil testing. Therefore good measurement methods or surveying

systems to bridge these gaps have been expected rather than basic equations or numerical analysis techniques. As one of those ways, there is a technique of feeding back the data picked up at a construction site into next step using back analysis in order to predict future phenomenon much more accurately. This builds a bridge between design and the actual project. This technique is called OBSERVATIONAL CONSTRUCTION, and it is part of INFORMATION INTEGRATED CONSTRUCTION (I I C) technology .

2. Information Integrated Construction Technology

2.1 Definition

The technology that uses several kinds of information in order to make the construction works or the design tasks efficient and productive is named INFORMATION INTEGRATED CONSTRUCTION [Suzuki 90]. It is defined as *"the technology in which collected technical data is given added value by bringing about optimum techniques and construction methods in each production process for construction."*

In this technology the information is not only the measurement data but also widely interpreted as expertise, and experienced construction data. The following three techniques are defined as a part of I I C Technology (Fig-2.1).

1) Observational Construction Technique

An observational construction technique mainly uses the measurement data . It includes measurement planning, measurement system and back analysis system for a construction site like the New Austrian Tunneling Method (NATM) tunnel sites or excavation sites using braced walls. The basic flow of the observational construction technique during construction stage is described in Fig-2.2.

A measurement planning task is done by experts. This task is very important for the observational construction

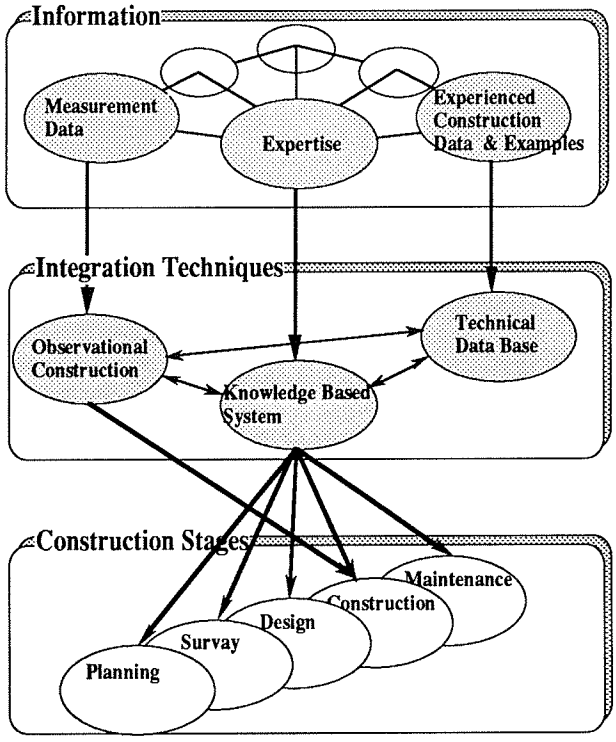


Fig-2.1 Basic Concept of IIC Technology

In the IIC technology, three kinds of information, measurement data (numerical data), expertise, and experienced project data, are used. These information are added value by each existing systems and are integrated by knowledge based system.

technique because the locating of measurement equipments by the technique of 1predicting failures which might happen during construction work influences the monitoring task's accuracy and construction safety.

A measurement system is a system to pick data up automatically from site into computers, in order to show the data in a graphic window and to make a documentation to for the client.

A back analysis system is a numerical method to obtain soil properties which explain well the ground behaviors going on at a site. A FEM Model or an Elastic-Plastic Model are often used as the back analysis system. For example, Yamagata's Expansion Method and Morishige's Method are very popular as Elastic-Plastic Model system for analyzing the braced walls at a excavation site.

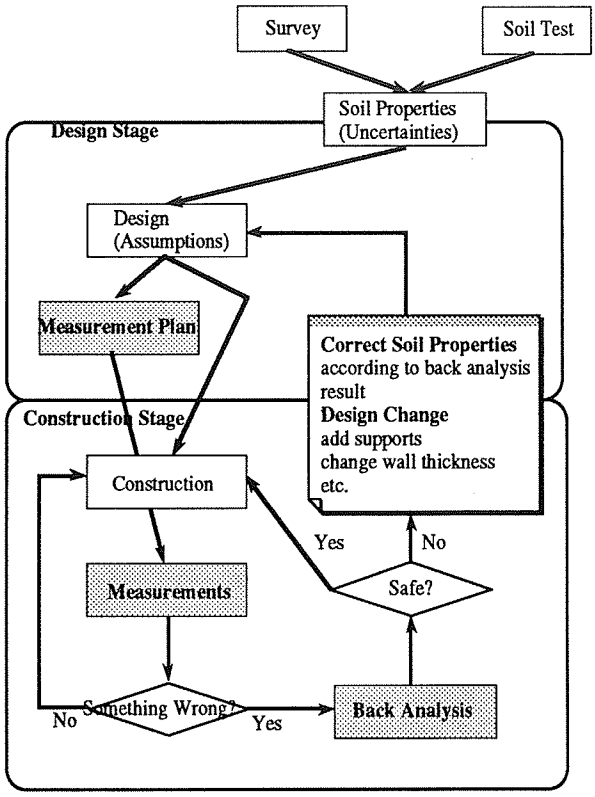


Fig-2.2 Observational Construction Flow

At survey and soil test stage, uncertainties exist and at design stage assumptions are included. To manage site safety and productivity with these two gaps between design and actual site, measurement and data analysis are necessary. In this figure the flow of observational construction, measurement planning, measurement and back analysis, are introduced.

A position of Observational Construction Technique in Information Integrated Construction Technology is shown in Fig-2.3. In this illustration, the relation between observational construction and IIC technology are explained more concretely using some existing techniques.

2) Knowledge Based System

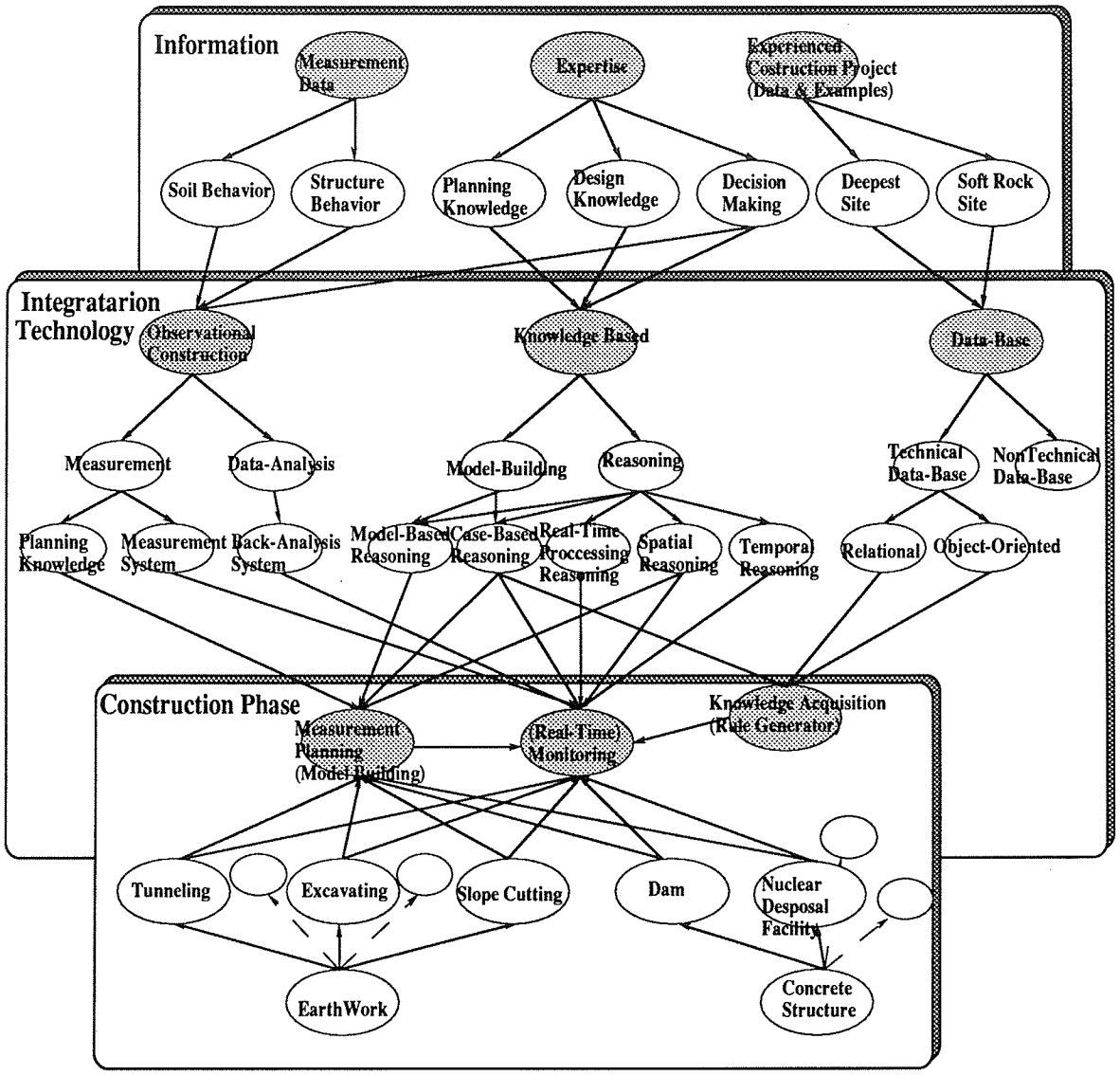


Fig-2.3 IIC Technology at Construction Phase

In this figure, the relation between observational construction and IIC technology with some existing techniques are explained in more detail . At the construction phase, monitoring is necessary for safety and productivity of the site. IIC technology integrates three kinds of Information measurement data, expertise and experienced data using the observational construction system, knowledge based system and data-base system.

A knowledge based system is using expertise, and this is a expert-supporting system for planning, design, decision making, control and diagnosis at each construction stage. This area is being explored now. There are many challenging things to establish a methodology for a ealistic system. This is true particularly in the construction industry. Although so far heuristic rule-based system has been mainly developed, Model Based Reasoning as a model

building technique and Case Based Reasoning for dealing with huge experienced project data will be come more focused.

In the Japanese construction industry the number of expert systems developed is increasing, growing four times in last a couple of years.

3) Data Base System

A data base system uses experienced construction data and examples, and is a expert-supporting system for decision making. This will be used as a resource for the knowledge based system in IIC technology. The recent tendency of recent is object oriented data base. That is why, owing to the Object Oriented Paradigm used to be in CAD system, it is becoming necessary for data-based systesms to input and output with object information. Moreover object oriented data base has some advantages such as the fact that the user can retrieve the information of data attribute, the function can be written in the system, and so on.

2.2 System Integration

For observational construction, the systems described in section 2.1 have been developed in each company begining 5 or 6 years ago. So far just for managing safety of construction sites these systems have not been integrated because of lack of methodology.

Due to the increase of construction sites, however, the shortage of labor and experts is becoming a serious issue in the construction industry. Therefore recen thinking is that in order to consider the progress of productivity of each construction sites, the observational construction technique should be established and the tasks which have been done by experts must be taken over by systems.

In order to integrate the isolated system, the knowledge based system is necessary for following reasons.

The knowledge based system can :

(1) handle the task of exchanging or adding information and data among the observational construction systems which have been conducted by experts

(2) deal with the non-numerical data easily. Decision making conducted by experts in construction work and design work includes many areas which can't be resolved by simple measurement and analysis techniques.

In order to integrate the observational construction systems, in this research the field is specified to excavation sites using braced walls. That is because in these days the number of excavation sitse is increasing, and they are becoming deeper and closer existing facilities. Therefore clients demand higher management technique. The observational construction

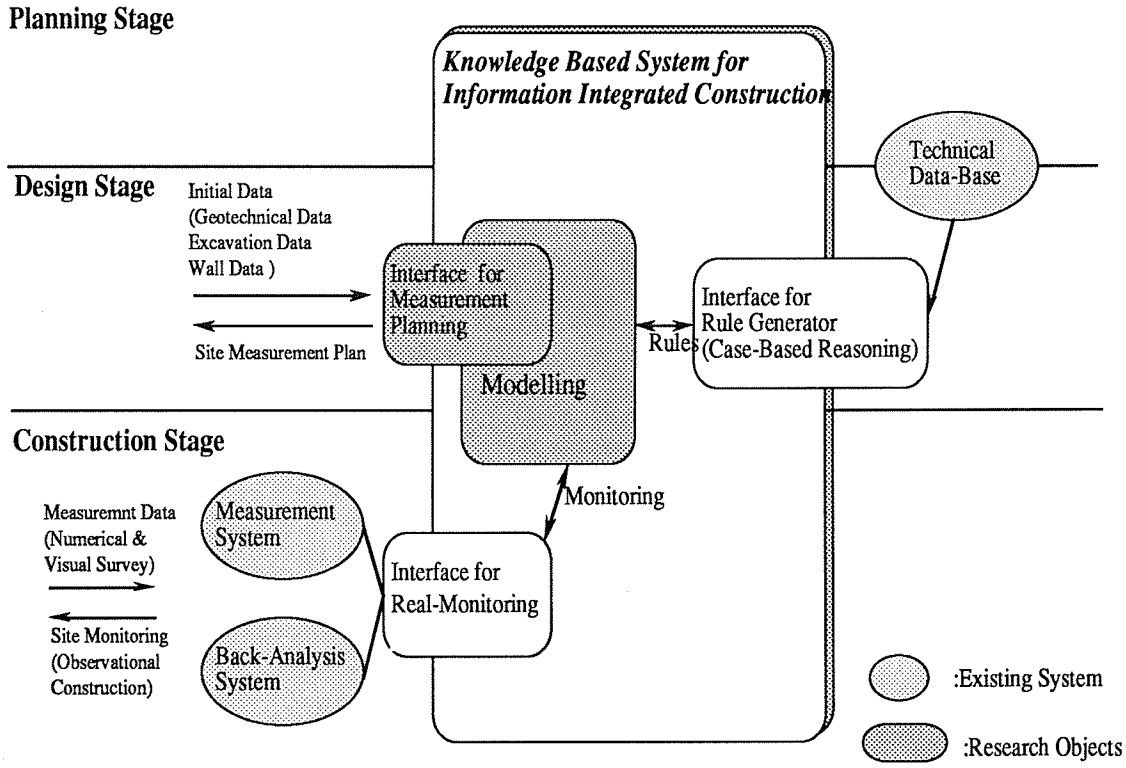


Fig-2.4 Concept of Observational Construction System in IIC Technology

In order to integrate the observational construction system, the followings are necessary:
1) Modelling technique of excavation site, site failures, measurement plan.
2) Case based reasoning to obtain experienced knowledge from data-base.
3)Real time processing reasoning technique to manage site safety and productivity.

technique is becoming the most important one in the construction fields. I am convinced that a efort in this dirrection of research will be able to create a system that integrates and manages site safety and productivity. Fig-2.4 illustrates the integration of observational construction for excavation sites. Each system of observational construction has been already developed, and to integrate these systems a good environment of knowledge based system is needed. In that figure to make a monitoring system a model which explains excavation site, site failure and measurement plan are needed. Along flow of observational construction, the modeling of site must be done at the measurement planning stage. Then that model built by measurement planning would be extended for monitoring at the next stage. Therefore in this research, I build a measurement planning system by focusing on the modeling technique of the excavation site.

3. Measurement Planning System

3.1 Research Goals

The goal of this research is to develop a measurement planning system for excavation construction using the ideas of Model-Based Reasoning. This is a knowledge-based system for the Information Integrated Construction (IIC) technology. This system provides appropriate location of measurement equipments to monitor excavation sites, While considering construction geometries, design conditions and ground properties. In this system these answers are introduced by using knowledge such as selecting measurement cross sections, predicting failures which might occur and locating measurement equipments

Through developing a measurement planning system, it is also important to confirm a methodology about how to make a model of the excavation site, and how to build a site failure model.

Fig-3.1 shows the basic function of this system. Through this system the user can get an idea of the positioning of measurement equipments and the relations among failures and environmental condition.

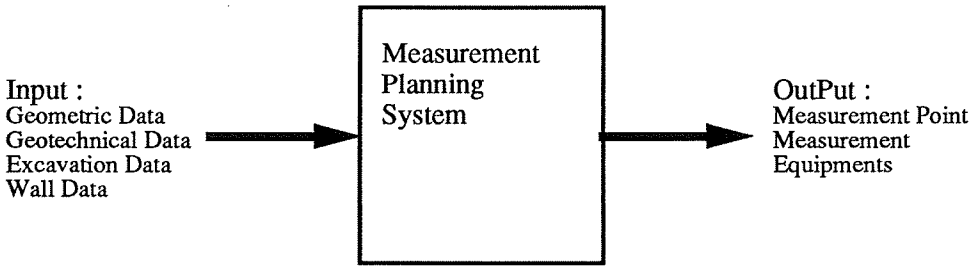


Fig-3.1 Basic Flow

By knowledge implemented in the system this system can provide users the proper location of measurement equipments.

3.2 Methodology

1. Use Model-Based Reasoning

Recently, the knowledge based system that uses only heuristic rule based reasoning seems to have almost reached its limit because of the difficulties of acquiring expertise and extracting the cause from the result of reasoning, except the specific domain such as medical science where the relation between causes and effects are not well understood. In place of the heuristic rule based reasoning the model-based reasoning is coming to the front. This reasoning idea was introduced by John C Kunz and others a couple of years ago [Kunz 88]. This approach attempt to explain the problem domain explicitly with small objects which have properties and behavior, and to use first principles to present behaviors. By means of it as numerical analysis can physically and mathematically explain the relation among model using first principles, the model using the approach of model-based reasoning could be anticipated for its extensibility and flexibility.

Up to now, the model-based reasoning has been used successfully in the domain of building and plant. This approach is useful and necessary in the geotechnical domain , because the behavior in the ground is well understood even though it has much uncertainties. In the future, because in this field there are many basic equations and functions, the qualitative reasoning and the quantitative reasoning based by model based reasoning will progress. Therefore to introduce the model based reasoning to geotechnical domain would be a major.

2 Incorporate Geometric Reasoning

It is easy for humans to explain and understand the spatial condition, such as patterns and relationships among the objects, by looking at them. However it is difficult to let a system understand relative location in particular three dimensional space. There is no formal

software in this field.

In the observational construction technology, the nonnumerical information like "There are some cracks in the ground **behind** no. 3 wall" or "Leakage of under ground water began at area 3"are very important for grasping the situation of a site globally. Object-oriented paradigm is very helpful for this kind of problems. It makes it easier for the system to incorporate geometric reasoning, because the object can have relations with others in itself as properties and behaviors.

3 Systemize the expert strategy

This systemization includes the categorization of failure patterns , the extraction of the relation between failures and causes, and summariries of locating know-how of measurement equipments.

In order to build an integrated system for observational construction, it is very important to categorize the failure pattern and to extract the relation. The failure patterns are categorized based mainly on experienced projects, geotechnical judgements and literature investigations. These efforts have important meanings in order to summarize expertise in the flow that the shortage of experts and implant the expertise into system as knowledge base. In particular in the field of observational construction, decision making are almost always relies on experts. This knowledge and model could be used in the real-time monitoring system.

4 Implement the knowledge based system.

This research is developed by three different stages in order to recognize the extensibility of the model. Fig-3.2 shows each stage. In the first stage, the geometric reasoning is incorporated for selecting measurement cross sections. In the second, the rules

for predicting failure patterns from environmental condition are written. At the last stage, the procedures for locating measurement equipments are implemented.

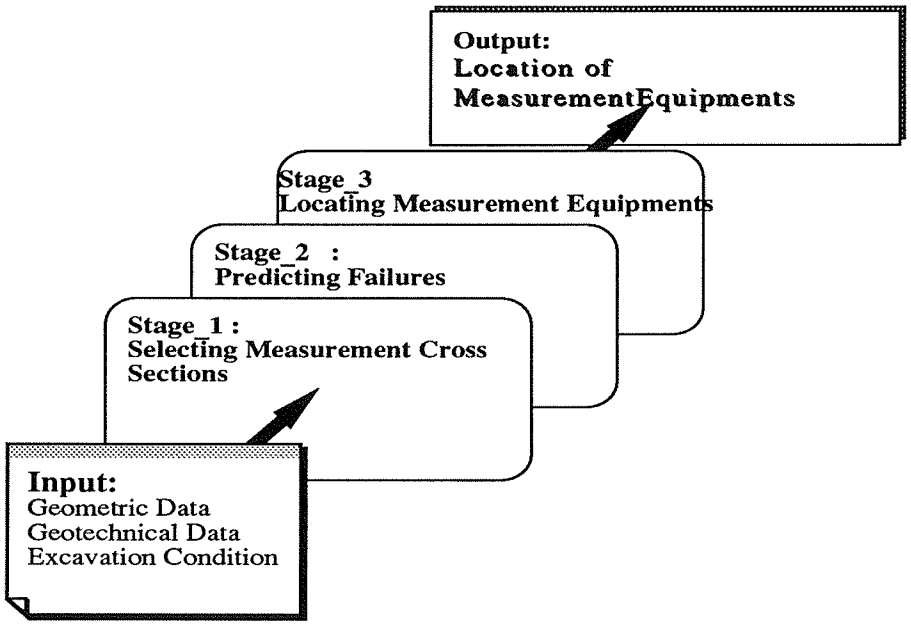


Fig-3.2 System Flow & Knowledge Stages

These stages are implemented in the prototype knowledge based system.

Stage_1: According to geometric and environmental data at site, system selects measurement cross sections as monitoring cross section in site.

Stage_2: According to soil properties and ground condition, system predicts failures which might happen during construction using forward chaining.

Stage_3: According to the failures predicted at previous stage, system decides measurement type for each measurement cross section and proper location of measurement equipments

3.3 Measurement Planning Knowledge

There are some kinds of knowledge, geometric, heuristic and algorithmic etc. in this domain. A decision making at each planning stage is conducted by combining complexly these knowledge and information. As mentioned in the section 3.2, measurement planning is subdivided into three stages, selecting measurement points, predicting failures and locating measurement equipments (See Fig-3.2). The following sections describe each knowledge I implemented in the prototype knowledge based system.

3.3.1 Selecting Measurement Cross Sections

In order to decide the measurement cross sections of wall, the displacement of wall is focused. In other words, the most dangerous cross section on the site should be selected and managed. Therefore the center of every wall facing on the deepest excavation area are chosen as measurement cross section candidates. Moreover some priority are usually put on these cross sections by means of those points showed bellow.

(1) Soil condition behind walls

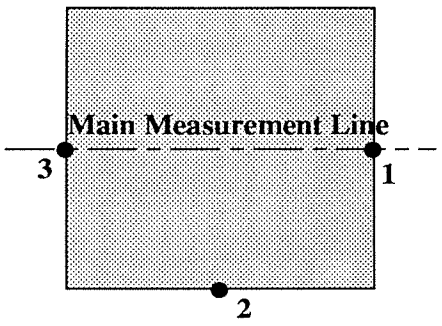
(2) Existence of important facilities near the wall.

(3) Distance from facilities

(4) Length of wall

(5) Depth of excavation

(6) Importance of facilities.



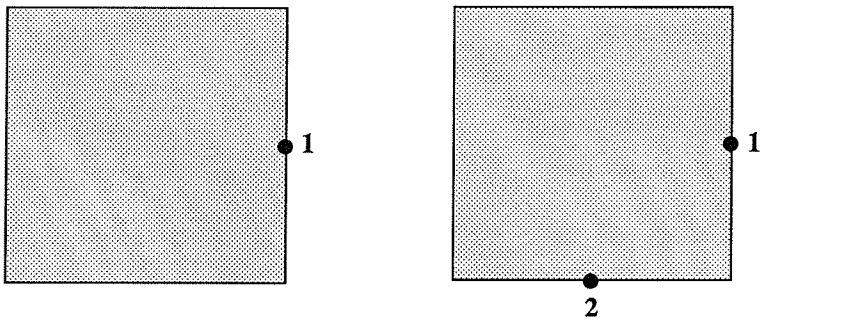
Usually 3 measurement points are chosen to manage the excavation site. The line goes through the biggest priority point in above figure is called main measurement line. It is a line to be focused during construction.

Fig-3.3 Main Measurement Line

The number of measurement cross sections depends on the construction site. It is deduced by the depth size or soil condition of theexcavation area. Usually three cross sections are selected. Two of

them are facing each other sandwiching the deepest excavation area in order to consider the global movement of the site. Using measurement data the layer movements should be checked, stress of struts, and so on. These measurement cross sections are necessary to analyze the causes of the failure using the back analysis technique when serious failure happens. The line through these two measurement cross sections is called the main measurement line. It is deduced by the top priority measurement cross section. The third one is placed on the center of the wall which is next to the top priority one's wall (Fig-3.3).

Sometimes there is the limitation of the number of measurement cross sections because of the budget for measurement. In the case of only one cross section, the first priority cross section would be up. If two cross sections can be picked up, the top priority one should be chosen first, and then the cross section which isn't facing on that one should be taken as a second one according to priority number (Fig-3.4).



a) One Candidate
If the site allows to set just one measurement point, the biggest priority point is chosen.

b) Two Candidates
If the site allows to set two measurement points, the biggest priority point is chosen first and then the measurement point on the next wall is chosen as a second point.

Fig-3.4 Measurement Cross section Location

3.3.2 Predicting failures

Predicting failures which might happen during construction is necessary in order to decide kinds of, number of and position of measurement equipments. There are two things to be conducted for predicting the failure. One is the categorization of failures. The other is making the relation between the categorized failures and causes which can be seen or measured.

1) Categorization of failures

Fig-3.5 shows that failures are categorized into five groups by focusing on the behavior of the ground and wall.

Total Sliding often happens in soft ground like reclaimed ground or along slipsurfaces. This is caused by the circular slip of poor ground or sliding of existing slip surface under the walls. Wall Failure is provoked by the sliding of the slip surface existing behind the wall, the large lateral earth-pressure or the shortage of wall strength. Anchor failure occurs by shortage of the anchorage strength in soft ground, cutting of the rod caused by the shearing force of the slip surface. Settlement and Heaving are caused by poor ground condition.

2) Relations between failures and causes

Referring to Fig-3.5, it is evident that the actual failure of the excavation site complicatedly occurs by the mixtures of some causes.

Some of them can be measured, some cannot. According to Fig-3.5 and our experiences, we chose representative causes considered for making the measurement plan.

> Existence of slip surface

> N-Value curve of ground

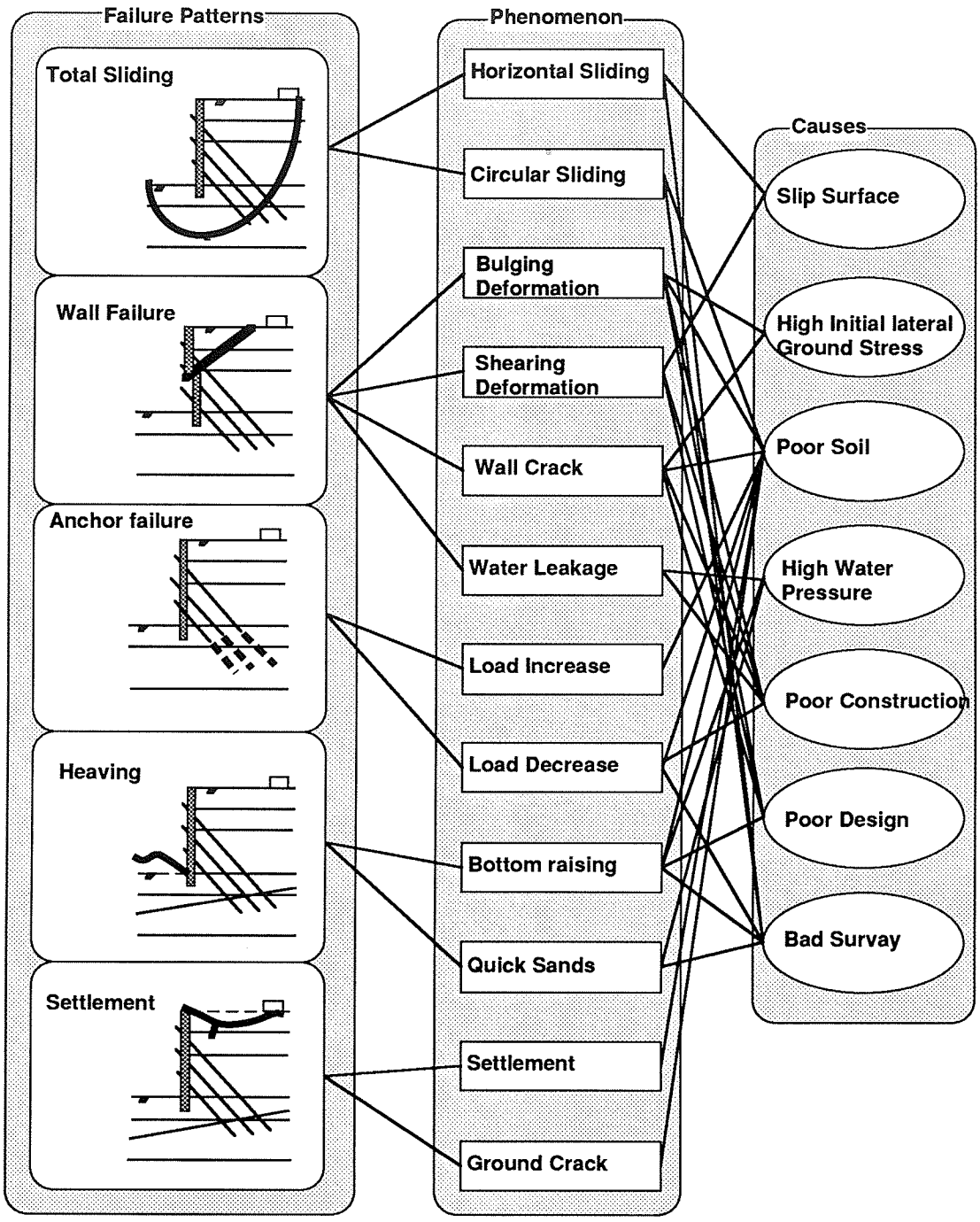


Fig-3.5 Failure patterns

According to this figure it is evident that failure is mainly caused by poor soil condition.

> Soil strength of wall bottom edge

> Pore water-pressure

> Penetration depth of wall

> Soil kinds of layer

Using above causes the relationship between these causes and failure pattern is shown in Fig-3.6. In the following flow chart, not only predicting failures but also deciding measurement types are introduced.

3.3.3 Locating measurement equipments

This locating knowledge is subdivided into two parts. One is the relation between

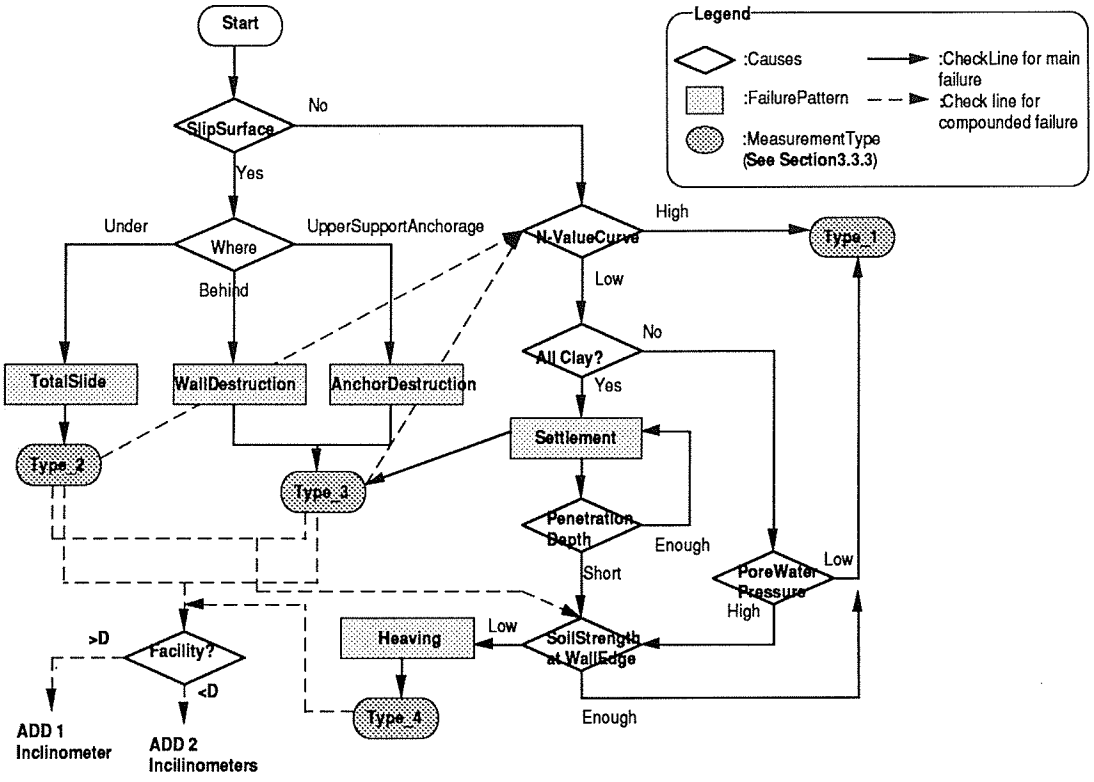


Fig-3.6 Flow Chart for Predicting Failures

By checking these causes a failure would be predicted. Considering expansion of this system because relationship of failures and causes are very complicated, these relations were implemented as rule-based reasoning. Written in rule-based system and using forward chaining, the maintenance of knowledge is easier than procedural description.

---

*1 The value obtained by Standard Penetration Test (SPT).
This is an indicator of soil softness.

measurement pattern and failure pattern. What kind of equipment and how many equipments are needed for monitoring during construction must be, in accordance with the predicted failure pattern. The other is the knowledge to put equipments on the wall or in the ground.

1) Measurement types

The relation measurement types and failure pattern is shown in Fig-3.7. At every measurement cross section, inclinometer in wall and load cells on each support are at least needed for back analysis. The inclinometer tells behavior of the wall, i.e. lateral displacement, and load cells provide external force acting on the wall. External force from the ground behind the wall is calculated according to the result of soil tests. Therefore inclinometer and load cells are necessary for data analysis when something wrong happens at the site. For the Total Sliding some inclinometer should be located to pick up the behavior of slipsurface up sensitively, in advance to the soil or wall corruption. Against the Wall Failure and the Anchor failure reinforcing bar stress gauges are needed to know the internal stress acting in the wall, otherwise site engineer can't decide whether wall is going to corrupt or not, or and when it will occure. Moreover earth&water-pressure gauges are needed to check earth-pressure behind wall. For ground's raising and sinking, settlement gauge are convenient but leveling survey can replace it. If there is the important structure near the site or high possibilities of existence of the slip surface, in this system inclinometer is added behind wall in ground.

Type	Measurement Equipments*	Failure Pattern
1	Inclinometer(w) + LoadCell	None
2	Inclinometer(w),(g) + LoadCell	Totally Sliding
3	Inclinometer(w) + LoadCell + ReinForcingGauge	Wall or Anchor Failure
4	Inclinometer(w) + LoadCell + SettlementGauge	Settlement,Heaving

*(w):in wall (g): in ground

2) Equipment Position

There is know-how as to how to put the measurement equipments on a site. That

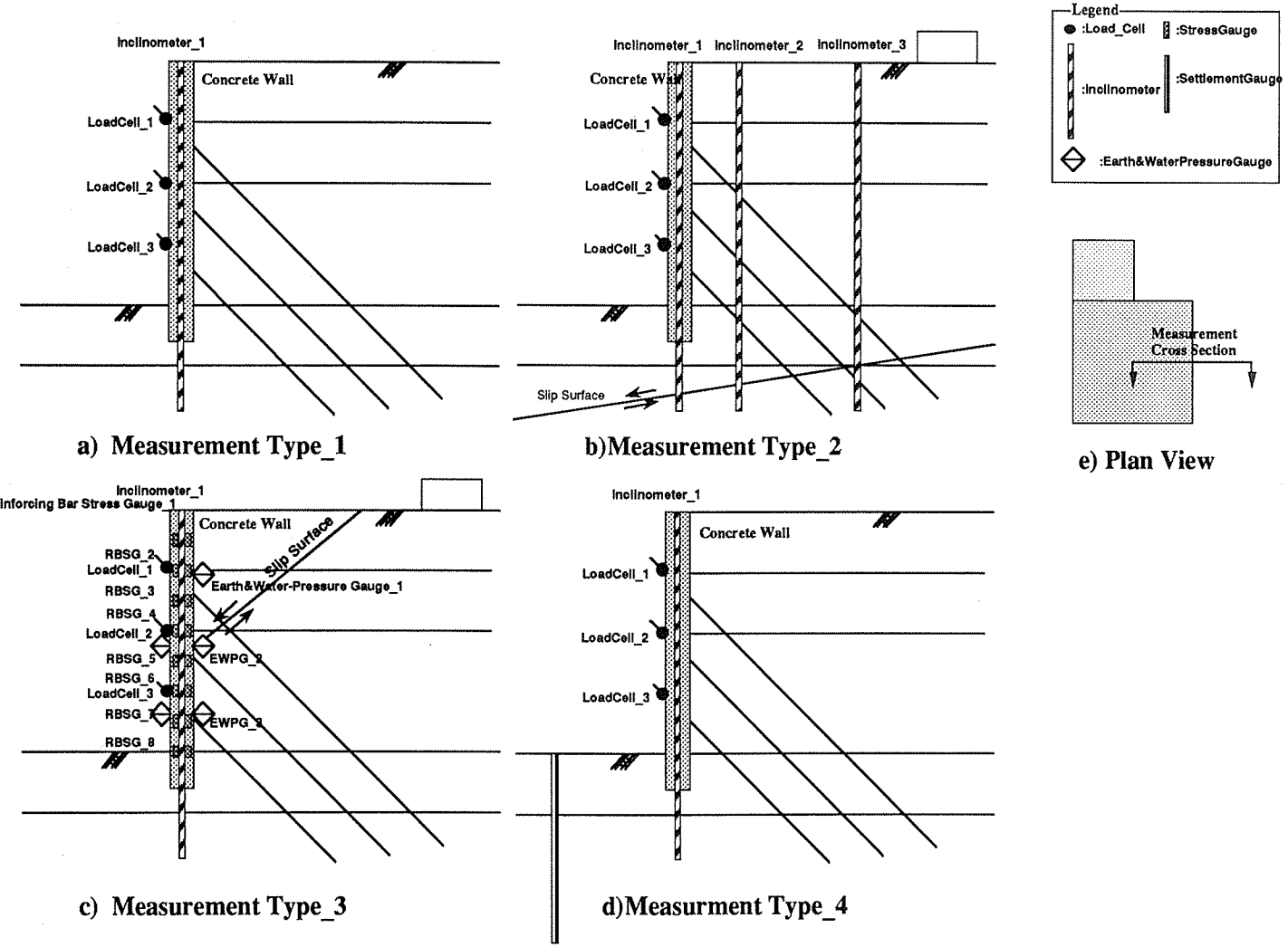


Fig-3.7 Measurment Type

In this system some possible compounded failures are considered. e.g. If the system says that Total Sliding and Heaving are predicted, the forward chaining would answer, "Add settlement gauge to measurement type_2". This is the flexibility of rule based reasoning.

depends on the purpose and what kinds of data are needed for managing the site. This section describes the purpose of each equipment and rules to be setteld.

a. Inclinometer

<purpose>

This equipment is used to get the behavior of lateral movement in the ground or wall. There are two kinds of it. One is insert-type, and the other is fixed-type. The former can continuously pick the lateral displacement up from the bottom to the top of the hole by inserting the pick-ups through the hole around 2 inches diameter dug before excavation starts. Therefore it is easy to know a cross section where the shearing displacement occurs. But because it can't be automated to send the data from the pick-up to the computer directly, the frequency of measurement is not often. It is usually once a week, increasing according to the magnitude of ground or wall movement. This is so sensitive that it is easy to know movement in ground or wall. The latter fixed-type can pick the lateral displacement up automatically from the pick-ups to the computer by setting pick-ups in ground or wall in advance every some vertical pitches. The good point of this type of measurement is that data can be picked upwhenever . But because the position of these pick-ups are set before the excavation begins, it can't be changed and if the shearing displacement occurs between pick-ups it can't be reconized when it happens. From this point of view, the former type is often used and recommended.

<rules>

Two kinds of position are needed to locate equipments. One is the position from the plan view, and the other is depth from the elevation view. Moreover there are two cases considered. One is in the wall, and the other is in the ground.

From the point of the plan view, position in the wall is the same as the measurement

cross section. In the case of the ground, it is deduced by the location of facility and the existence of slip surface. However, it is usually 5m - 15m behind the wall and on the line of the measurement cross sections.

From the point of the elevation view, the depth of each inclinometer is deduced by the condition of the wall's bottom edge. In the case that the layer of the wall's bottom edge is soft clay, we have to put the inclinometer 5 or 10m deeper than it in order to get a unmoving point. Thus there are some rules for deciding the depth of inclinometer. The following Fig-3.8 shows details.

b. Load Cell

<purpose>

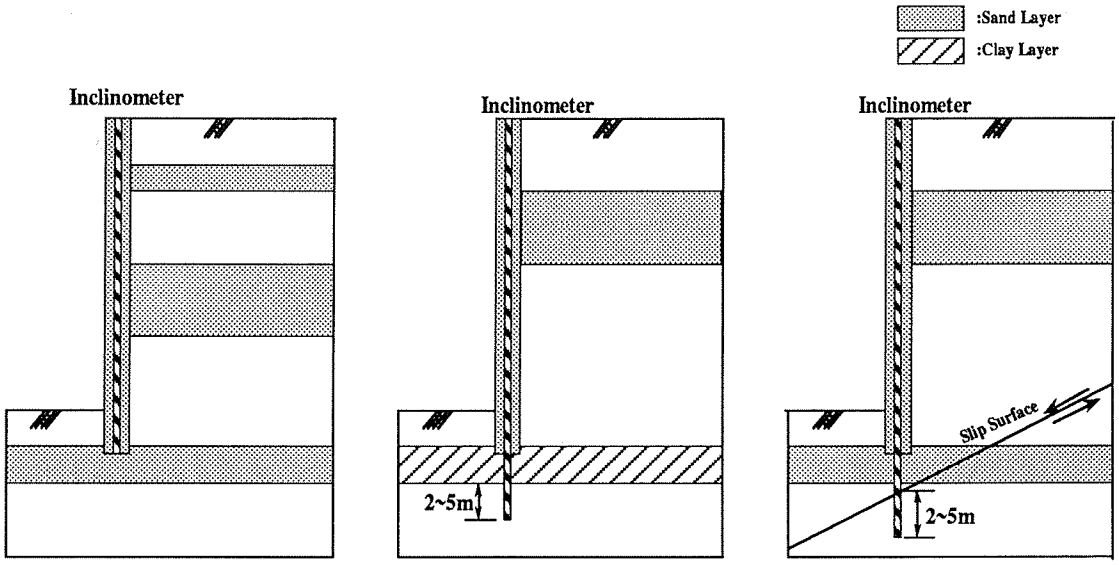


Fig-3.8 Inclinometer Setting Depth

a) In the case that ground condition at wall bottom edge is **sandstone or sandgravel**, the setting depth of inclinometer is **the same as the edge of wall.**

b)In the case that ground condition at wall bottom edge is **clay or mudstone**, the setting depth of inclinometer must be **5~10m deeper than the edge of wall** or **2~5m deeper than the bottom border of that layer.**

c)In the case that **slipsurface exists under wall**, the setting depth of inclinometer is needs to be **2 ~5m below the surface.**

In order to check the supporting materials such as struts and anchors if they are working well and in order to estimate the earth-pressure and external forces, load cells are needed on every supports. To analyze the behavior or the wall, these load conditions are necessary for our analysis model.

<rules>

Usually these load cells are put on every supports in every measurement cross section. Therefore the position of equipments depends on the position of those supports. Sometimes, to check buckling of long struts, strain gauge is used instead of load cell.

c. Reinforcing Bar Stress Transducer

<purpose>

This purpose of this measurement equipment is to check the stress acting on the wall.

<rules>

These are put on the reinforcing bar with 2~5m pitch from top of a wall to half depth of an embedment depth. The reason these aren't put around the wall bottom edge is that large moment doesn't occur. In order to obtain moment of the wall it is usual to set these equipments front bar and behind bar in the wall at same depth. Besides according to the result of detail design there are densely put these on the maximum moment depth. It is very important not to set these equipments on the same depth as supports.

d. Earth-Pressure Gauge and Porewater-Pressure Gauge

<purpose>

In ordinary cases, it is not necessaly to place these gauge. If there is slip surface behind wall and the behavior of sliding should be monitored, these gauges are useful for monitor ing the increase of the earth-pressure. These two kinds of gauges should be put

together to distinguish water-pressure from earth-pressure. Because the initial setting of these gauges is very difficult and the data depends on it, the accuracy of these gauges is not good enough to use in the numerical analysis. Therefore these are used for qualitative analysis or for the tendency analysis from the relative change of those data.

<rules>

Setting know how is to put front and behind of the wall at same depth, to prevent setting at same depth with supports and not to set on the wall shallow than 5m because there are no large pressure around the wall top edge.

e. Settlement Gauge

<purpose>

This is for getting the settlement value behind a wall or getting a bottom raising value in the excavation area. This is particularly necessary for soft clay like reclaimed ground.

<rules>

To check heaving, this must to be set at the deepest point or the center of excavation site. In order to check settlement behind the wall, it must to be set within distance a equal to the excavation depth.

3.4 System Architecture

3.4.1 System

This system was developed using KAPPA as expert shell. KAPPA is the system developed and marketed by IntelliCorp which has been providing good expert shell such as KEE, Pro KAPPA and so on. KAPPA is implemented in Microsoft Window on IBM PC. KAPPA provides Object-Oriented Paradigm and and Rule-Based Reasoning so that makes this system able to use Model-Based Reasoning. There are some expert shells which have almost the same function or more. As a representation expert shell KEE, ProKAPPA and KAPPAPC developed by IntelliCorp, and Nexpert-Object developed by Neuron Data. As a representative intelligent CAD system, DESIGN++ developed by Design Power Inc., ICAD by ICAD.

The reason why KAPPA was chosen for this research is that even through it is relatively small system, it has enough capacity to create the model and to use the model based reasoning. Moreover in this research a gorgeous user interface was not necessary.

The remarkable functions in KAPPA and word definitions is show below. The figure or word in the box [ ] indicates the number or methodology or name used in this measurement planning system.

Class [41] : A class object which has children objects and represents a collection of related objects with similar characteristics. [e.g., Wall, FailurePattern, MeasurementType, Support, etc.]

Slot[>200] : This exists in the object such as class and instance to explain properties of each object. [e.g., Coordinate, NearestWall, NumLoadcell, etc.]

Instance [95] : An object which is a termination of the hierarchy and a real-world entity. .[e.g., Facility_1, ExcavationArea_1, Wall_5, Inclinometer_3]

Method[26] : So called Demon. This is a sort of small program assigned to each object

because it specifies object behavior. Equations, if-then rules and procedures can be written in the function. There are three types of methods, If-Needed, Before Change and After Change. [e.g., DeepestExcavationSearch, Distance, CalPriority, etc,]

Reasoning [forward] : There is a Rule-Based Reasoning and KAPPA serves both backward and forward chaining

Rule[21] : This is formed with if-part and then-part. Using an inference engine this allows the user to specify the logical relation among objects. [e.g., WallFailureCheck, PenetrationDepth Check, LayerCondition, etc.]

Function[54] : Kind of Procedure. This can control the system globally. [e.g., Initialize, CheckLayerCondition, CountNumEquips, etc.]

The list of function and rule is appended to this paper (See Appendix _1).

Fig-3.9 shows the system architecture.

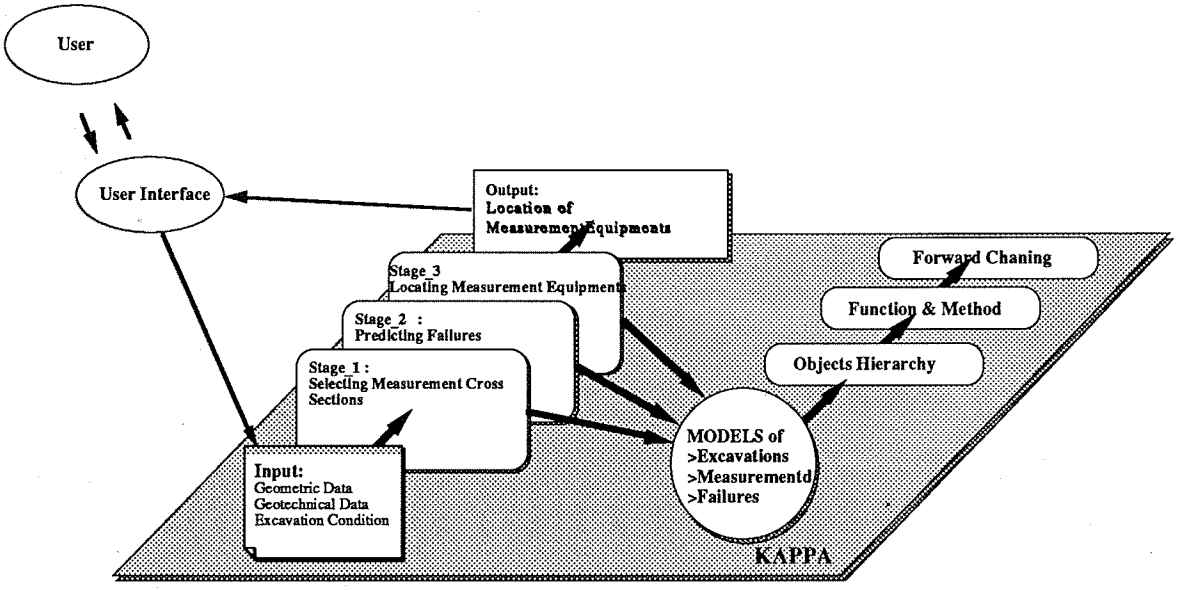


Fig-3.9 System Architecture

3.4.2 Assumptions

Listed bellow are assumptions of the system.

1) Excavation area is rectangular

2) Layers are horizontal.

3) All wall is made by same material, design and size. So quality of each wall is same.

3.5 Object Oriented Framework

There are three major advantages using the object oriented paradigm. The first one is knowledge inheritance. Using the hierarchy, this provides the specific world for a knowledge-based system unlike the case of semantic net work in which the relation among objects might diffuse.

The second advantage is that a model is build explicitly. The object oriented paradigm shows it capability, extensibility, flexibility and maintainability, by disconnecting the actual project into small objects and by making these objects have its properties and behavior in itself. This paradigm allows object to store its properties as slots value and its behavior as methods explicitly.

The last one is abstraction. The system using object oriented paradigm can communicate among the objects by sending and receiving message via the methods. By this capability objects can exchange information with others and tell the next behavior to another one or ask them to give some value. At the top level the functions don't know detail procedures but just ask some question to the children. The children know the next procedure according to the question so that they ask their children along next procedure. Thus at the top level functions don't need to be described in detail and these detail procedures are executed by each child implicitly until the answer is obtained.

Fig-3.10 shows the hierarchy of the system.

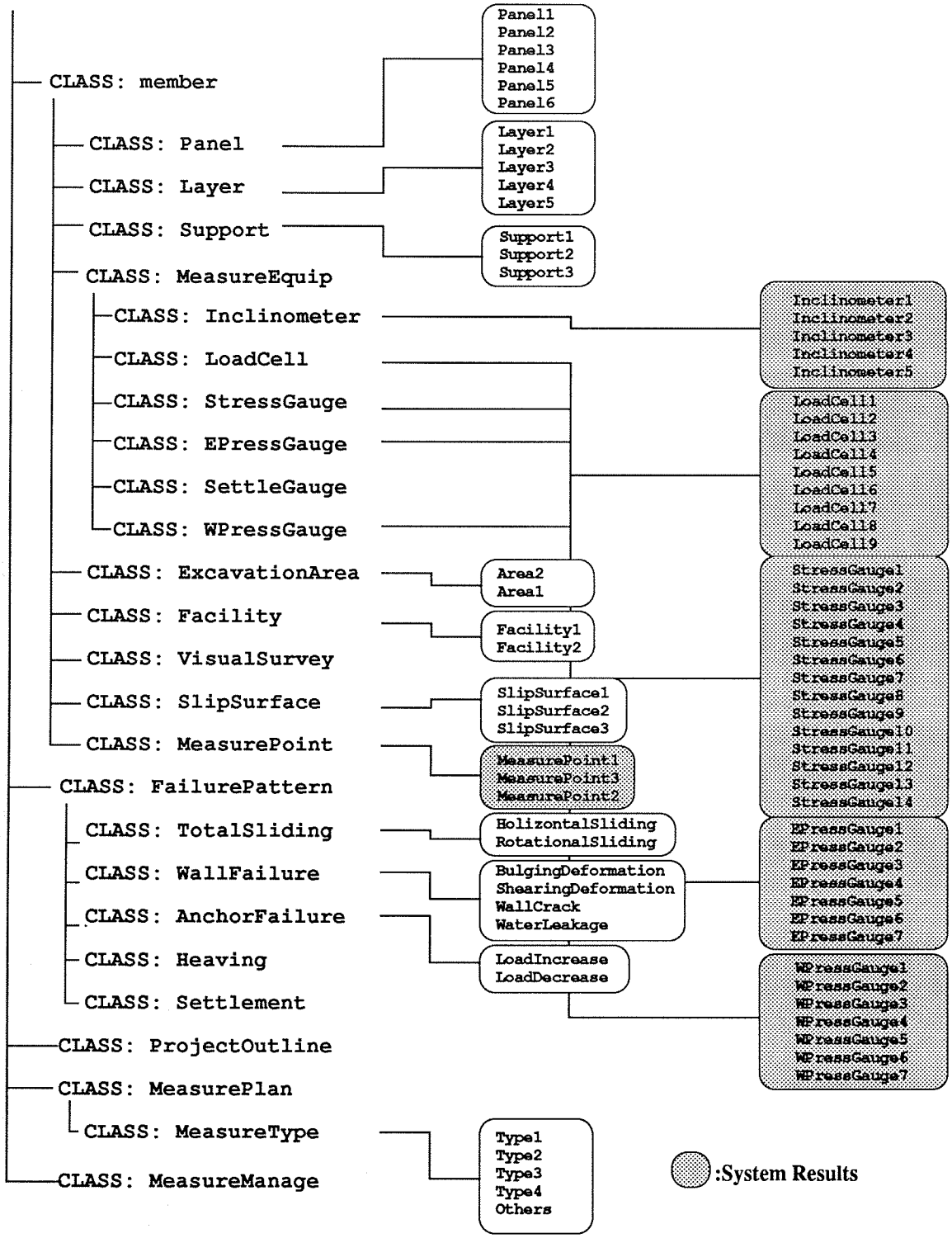


Fig-3.10 Hierarchy After Finishing Reasoning

The instances of measurement equipments and measurement points were created as results of reasoning.

The system has two kinds of knowledge representation as below.

1) Representation of project data

2) Representation of knowledge

3.5.1 Project Data

The system presents its project data hierarchy as below (Fig -3.11). There are geometric information, geotechnical properties, braced wall information and project information in it as slot value. These data could be easy to change by each project engineer according to the preliminary design.

Basically this hierarchy uses the "IS-A" type relation between parents and children. For the geometric reasoning the system focus on the "Part-of" type relation and uses the slots in which the relation could be written. Each object automatically inherits the parents slots and methods.

According to each project, only the terminations of hierarchy are changed. Therefore Basic knowledge, calculation of weight or geometric position of each object etc., could be written as a method in the top of hierarchy.

Fig-3.11 shows hierarchy of project data.

3.5.2 Knowledge Representation

There are three kinds of ways for knowledge representation.

1) Slot

Slot could be both heuristic knowledge and first principle knowledge. In the case of object oriented paradigm, the structure of hierarchy itself is knowledge representation. Moreover slot is a good place to store the knowledge and it is so explicit that the user can change it or see it easily. This knowledge is stored in the hierarchy as slots and slot value,

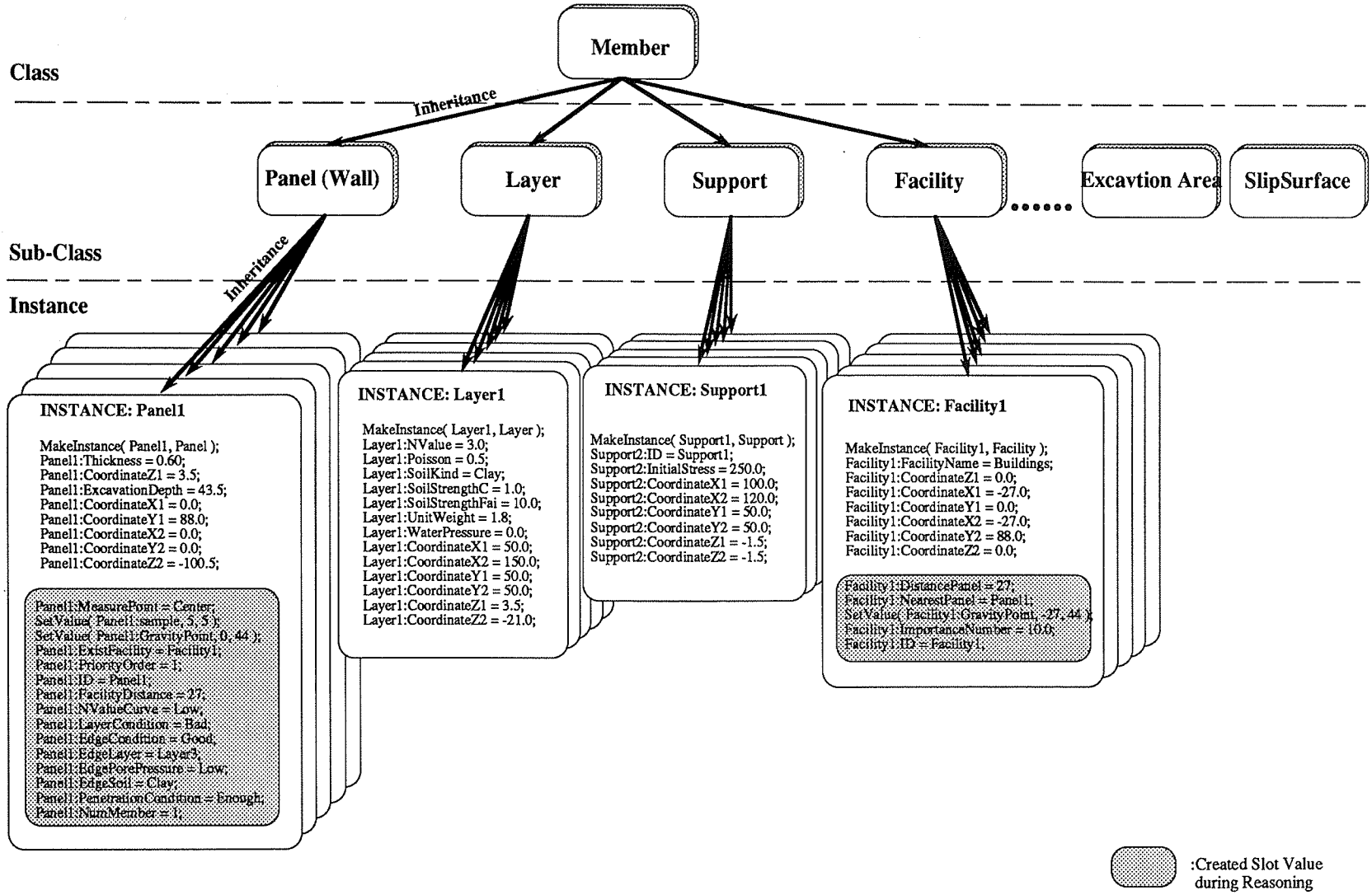


Fig-3.11 Project Data

These data are changed in each project. The information about slots, methods is in class sub-class level Therefore on each project data is inherited from class and these instances are created.

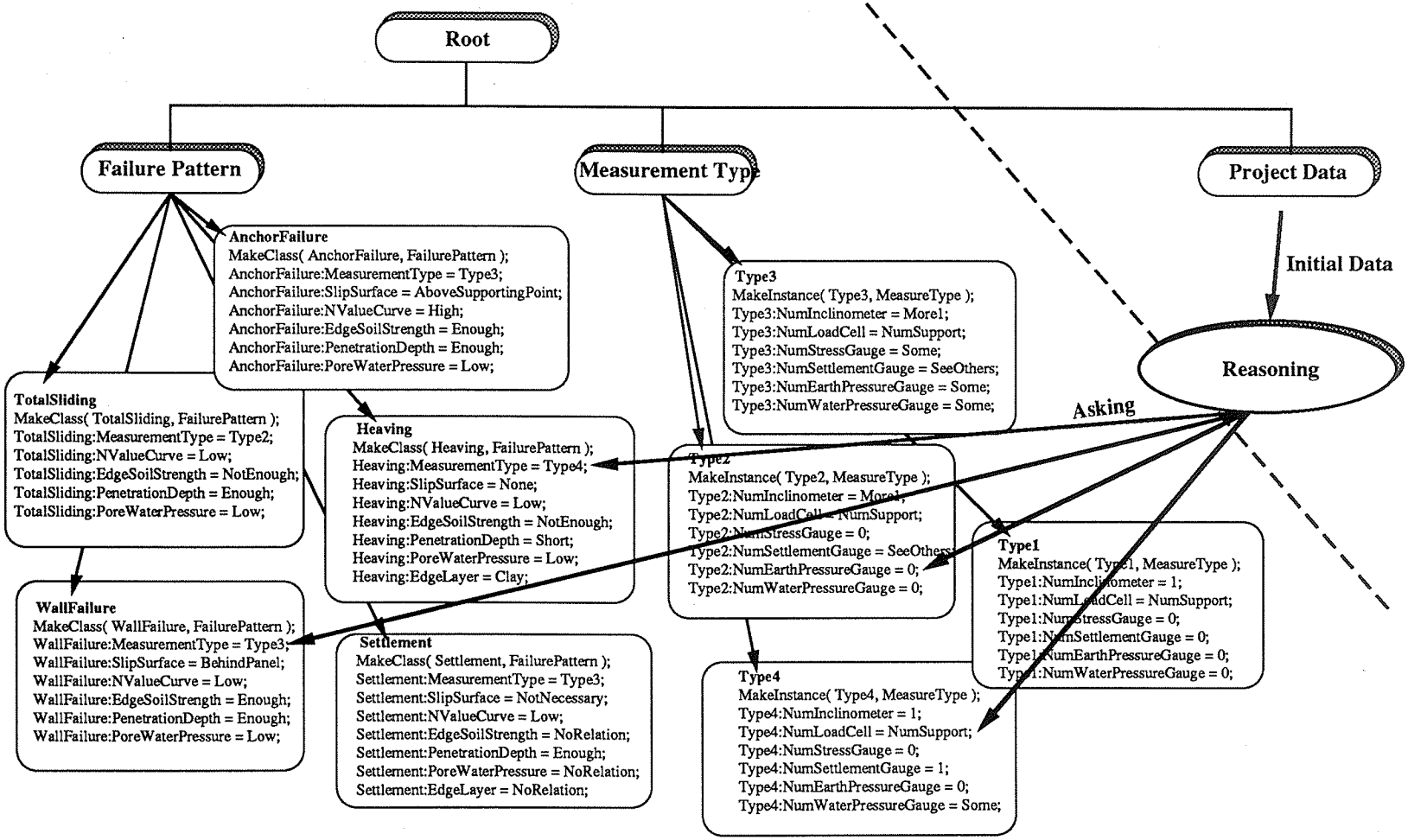


Fig-3.12 Knowledge Representation by Slot Value

These knowledge are easy to change.
These knowledge are used with methoods.

and hit by inference engine or functions. In the system the knowledge of failure patterns and measurement types are represented as a hierarchy (Fig-3.12) and slots.

Moreover combining with method, the knowledge representation capacity of slot becomes broader. The slot could be used as multiple inheritance and part of geometric reasoning, spatial reasoning and temporal reasoning. The explanation of slot, e.g. "Next_To", "Put_On", "Behind", "Yesterday" and so on, and searching function of method make those reasonings easier.

2) Method

In a method, the first principle knowledge is stored. For example, in the system if system is asked "What is a priority number of measurement cross sction_1 ? ". The system doesn't know the answer. Therefore the system asks the above question to the object "measurement cross section_1". Measurement cross section doesn't know the answer but know how to calculate and what kind of values are needed and where the values are. In this way the methods are triggered one after another to reach the object which has the answer to the question. Fig-3.41 shows the way of message exchanging among objects.

According to the above, it is evident that there is the information of address and calculation in a method. By using this kind of knowledge representation, the system doesn't care about the change of project data. Moreover the user can trace how to get the answer.

3) Function

Function can be both procedure and first principle knowledge. In order to manage the knowledge based system, the procedure is written by functions in KAPPA. These functions could be thought that they are same as subroutines in the program used as conventional numerical analysis. Therefore in the system the procedure is written by functions. The Function controls the rules and methods as below (Fig-3.14).

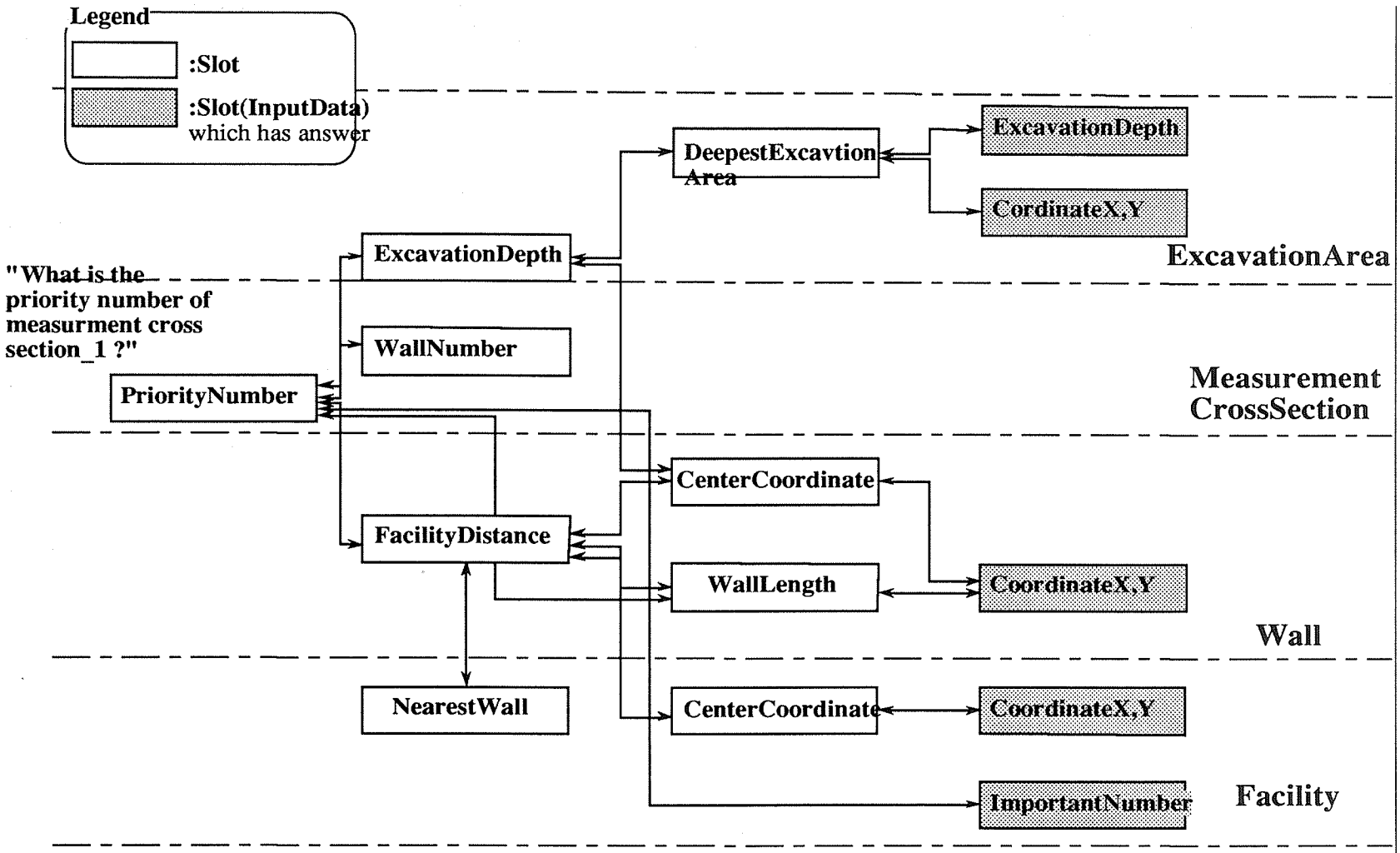


Fig-3.13 Method Network
This is a network made by methods. Each slot knows how to calculate the value asked by upper slot and who knows the value. The network starts from the slot "PriorityNumber" to lower slot, and then goes back to the first place when it meets the slot which has value.

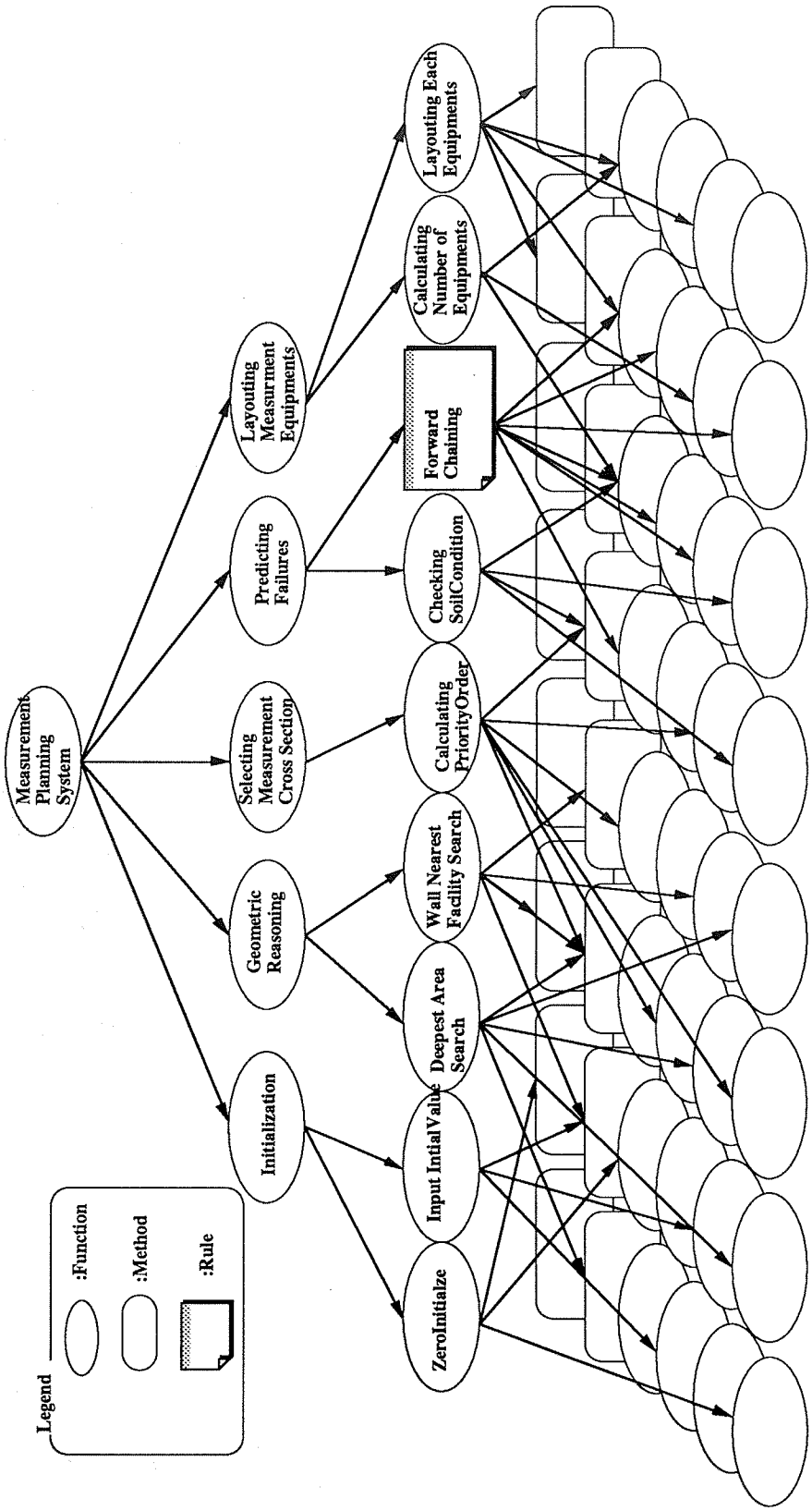


Fig-3.14 Reasoning Abstruction by Function
The top function controls lower functions just after it. Those functions then do their task using lower functions or methods. The top function doesn't need to know in detail.

Thus if the procedure is obvious, it is easy to make the system using only function like a conventional numerical method. In that case, If-Then logical relation would nest 2 or 3 in one function. Therefore the flexibility of system would be lacked. On the other hand, rules it is not easy to make system but easy to expand system. Therefore the knowledge should be divided according to the character.

4) Rule

Both heuristic knowledge and procedure knowledge are expressed as rules. The knowledge written in the rules are likely immutable. The system uses forward chaining to predict the failure pattern from the soil condition and excavation condition.

3.6 Implementation of Reasoning

3.6.1 Model Based Reasoning

The definition of Model Based Reasoning depends on the person who is trying to make a model of actual issues. Model Based Reasoning can take any actual issues as its model. It is often used for building or plant, and sometimes used for making a model for schedule or cost . As mentioned before, one of advantages of Model Based Reasoning is that flexibility and extensibility of model which conventional heuristic rule based reasoning have not had. By using Model Based Reasoning, the user can approach the actual issues object in many ways with only one model. That is because it can explain the actual phenomenon by disconnecting the actual issues into small parts and it makes each of them have the properties and behaviors in themselves.

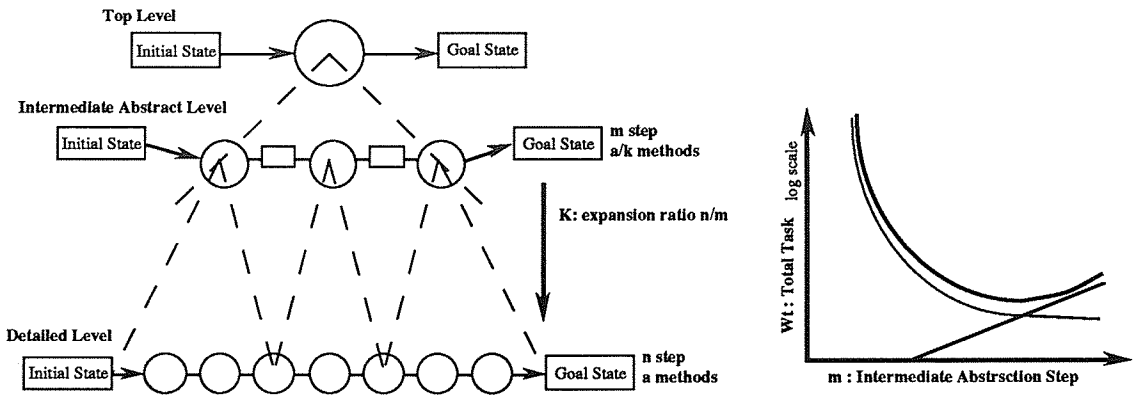
It isn't necessary to use Object Oriented Paradigm for Model Based Reasoning. But if it used, it serves good environment where those small parts are called "object", properties are stored as "slots" or "attributes" and behaviors are defined as "method" or "demons". These names depend on system user uses. In that way, Object Oriented

Paradigm provides user good environment to express a model explicitly and implicitly.

The problem of Model Based Reasoning is how detail issues should be disconnected. It determines the flexibility of model. If the parts become smaller and smaller, at last it would enhance its capacity more than conventional numerical analysis. That is why the numerical analysis has only the purpose obtaining deformation of wall, while the Model Based Reasoning, constituted with small parts, can do many things with only one basic model. It can obtain the deformation of wall, and it also can make the measurement plan or do monitoring and back analysis at the site. On the other hand, the more objects are taken down into small parts, the greater the task of describing the relation among object. The size of th task for making a model would grow in an arithmetical progression in accordance with the amount of parts function. Therefore there is large difference in the point of where the starting point of model is. For instance, thinking excavation wall as a rectangular geometric object is quite different from that considering it as a concrete wall by conceiving its design. If starts from the geometric object, it would need many properties and behavior to describe excavation site model.

Therefore, rule based reasoning would be necessary. Using some rules as a assembler of some first-principles and methods, somehow the long way would be reduced.

Using idea of task optimization [Stefik 90], it is understood obviously that in the Model Based Reasoning some rules are needed as abstraction of object relationship to make its task efficiently. Considering that there is one intermediate abstraction level between the top-level description of the problem and detailed-level in the hierarchical task (Fig-3.15), the amount of task in Model Based Reasoning ensues from the following equations.



a) Hierarchy Structure

It is assumed that there is a intermediate abstraction level between top and detailed.

b) Task - Abstraction Relation

There is a optimum point for abstraction. Intermediate abstraction is necessary for reducing the amount of task.

Fig-3.15 Hierarchical Task Analysis

$$Wa = (a/k)^m = (a/k)^{(n/k)} \quad (3.1)$$

$$Wd = m(a^{n/m}) = (n/k)(a^k) \quad (3.2)$$

$$Wt = Wa + Wd \quad (3.3)$$

a	number of first-priciple and method at the detailed level
n	number of steps in the detailed model
m	number of steps in abstract model
k	Expansion Ratio (n/m)
Wa	task at the abstraction level
Wd	task at the detailed level
Wt	total task

In this research, I used the geometric reasoning as a example of relatively detailed explanation of model and rule based reasoning as a relatively intensive example of model.

I used geometric reasoning to select measurement cross sections and rule based reasoning to predict failure pattern.

3.6.2 Geometric Reasoning

Recently, according to the progress of intelligent CAD system, the system which has the function not to reason but to represent geometric relationships among objects appears. As mentioned previously, even though it is the very simple spatial location from human's eye, e.g."In front of " or "Behind", it is difficult for the computer to understan and search it. But object-oriented paradigms provide a way for writing these relations and making it easier. These geometric relations in the object-oriented paradigm are written 1) in the slot as a result of searching like "connected to" , 2) in the methods as a procedure to obtain relationships among objects like " If (the distance between A and B > 0) Then A is located **Behind** B", 3) in the rules as a searching technique like "If (connected_to = nil) Then name it termination". By means of that these information are

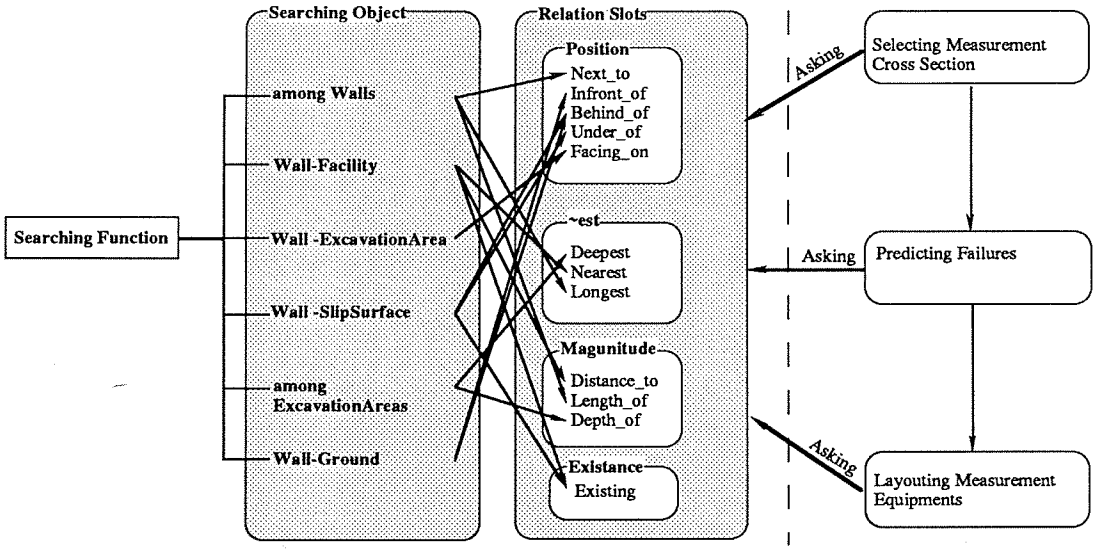


Fig-3.16 Geometric Reasoning

controlled by rules or procedures, the geometric reasoning can achieve it purpose implicitly.

In the system, the following rules are used as a geometric reasoning in order to deduce measurement cross section (Fig-3.16).

3.6.3 Rule Based Reasoning

In the system, the relation between causes and failures, failures and measurements are described using If-Then rules. These rules are hit by inference engine implicitly and change the slot values in the system. As a inference engine there are two kinds. One is forward chaining and the other is backward chaining. In the system, it is finding consequences of causes so that the forward chaining is used.

In the system, the environment of Kappa supples some browser for managing the rule base reasoning. A user can trace the relation of each rule and the rule chaining during reasoning.

3.7 User Interface

KAPPA has its own window system. It is pre-defined graphic functions such as Bottom, Line Plot and so on. However these are not enough to describe the result of reasoning flexibly. Using it, the system allows the user to view the hierarchical relations among objects or chaining relations among the rules, and to follow the impact of reasoning on particular slots in the knowledge base. Thus through the window system the user can acquire reasons of the result. The system also provides input management window and reasoning control buttons as user interface.

The system doesn't have the interactive graphic interface like KEE or other

intelligent CAD systems. KAPPA has a capability to connect the graphic software using C language [Fischer 91]. However it is not this researches main object so that making interactive graphic interface will be the next step.

3.8 Validation

In order to validate the measurement planning system, it was adapted for two kinds of actual projects conducted in Japan a couple of years ago. One is a pumping plant and the other is wastewater treatment plant. The former is a project on reclaimed ground so that it has bad soil condition. Moreover it is a very special project with deep excavation (43.5m) and needs long penetration depth (60.5m). The latter has good soil condition compared to the former one but there is a slip surface behind excavation walls. In the following sections the result provided by the system according to each excavation site's condition is compared to the result done by experts. And then the gaps between these two results is discussed.

Considered the assumptions in this system, an actual project which had a rectangular excavation site was selected.

3.8.1 A Pumping Plant

1) Input Data

Fig-3.17 showed plan view and Fig-3.18 illustrates representative elevation view of the construction site for a pumping plant. Because this project is a pumping plant, the location of this site is in city and near river. Therefore ground condition is not good and there is resident area near this site.

2) Output Result

The results of this system are shown in Fig-3.19 and -3.20 comparing with the actual plan.

From the point of plan view, the differences between those two are that the following equipments are lacking in the result of this system.

Wall_1 : Two Inclinometers in the wall, Concrete Effective Stress Gauges and Distance between Inclinometers.

Wall_2 : One Inclinometer in the ground behind wall and Earth-Pressure Gauges

Wall_3 : Reinforcing Stress Gauges and Earth & Water-Pressure Gauges

Wall_4 : Distance between Inclinometers

From the point of elevation view, the difference between the two are the number and pitch of reinforcing bar stress gauges and earth & water-pressure gauges measurement points. The system provides the necessary number to monitor the site, assuming the site is very important without considering cost-benefit relation. In the actual plan, the pitch and number was decided by how important the expert thought this site was according to the past experienced projects. He is doing so based on his own reasoning.

3) Contemplation

These additional measurement equipments done in the actual plan were added because this project is un unusual one. This is one of the deepest excavation area near a city , The walls, which have 60.5m long penetration depth, have never existed in Japan. Therefore additional inclinometers, reinforcing bar stress gauges and concrete effective stress gauges were added for two reason. One is for Monitoring the corruption of walls by vertical axis force because the walls are so deep. The other is for research. Because we have never experienced deep walls in the soft ground like this, this is a good actual-size experiment for the future project. But the system only recommended measurement points necessary for safety and good practice. Therefore it can't follow the research purpose or special case yet. The system provides user the general idea of measurement plan against a normal project.

The difference of distance between inclinometers in wall and in ground is not so important as long as that the location of inclinometer isn't too close the wall or facilities.

3.8.2 A Wastewater Treatment Plant

1) Input Data

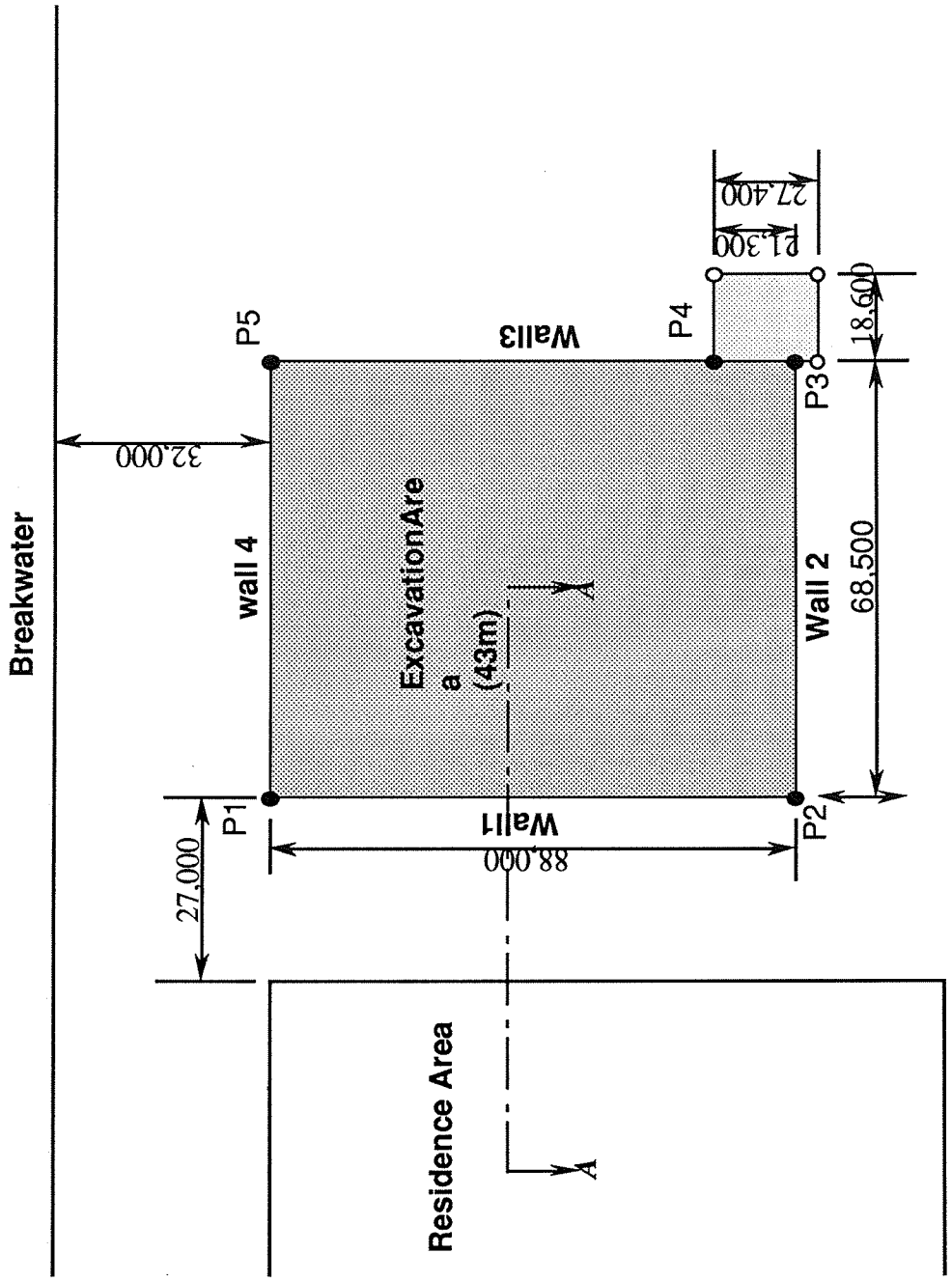


Fig-3.17 Pumping Plant - Plan View
(Test Case_1 Used in the Research)

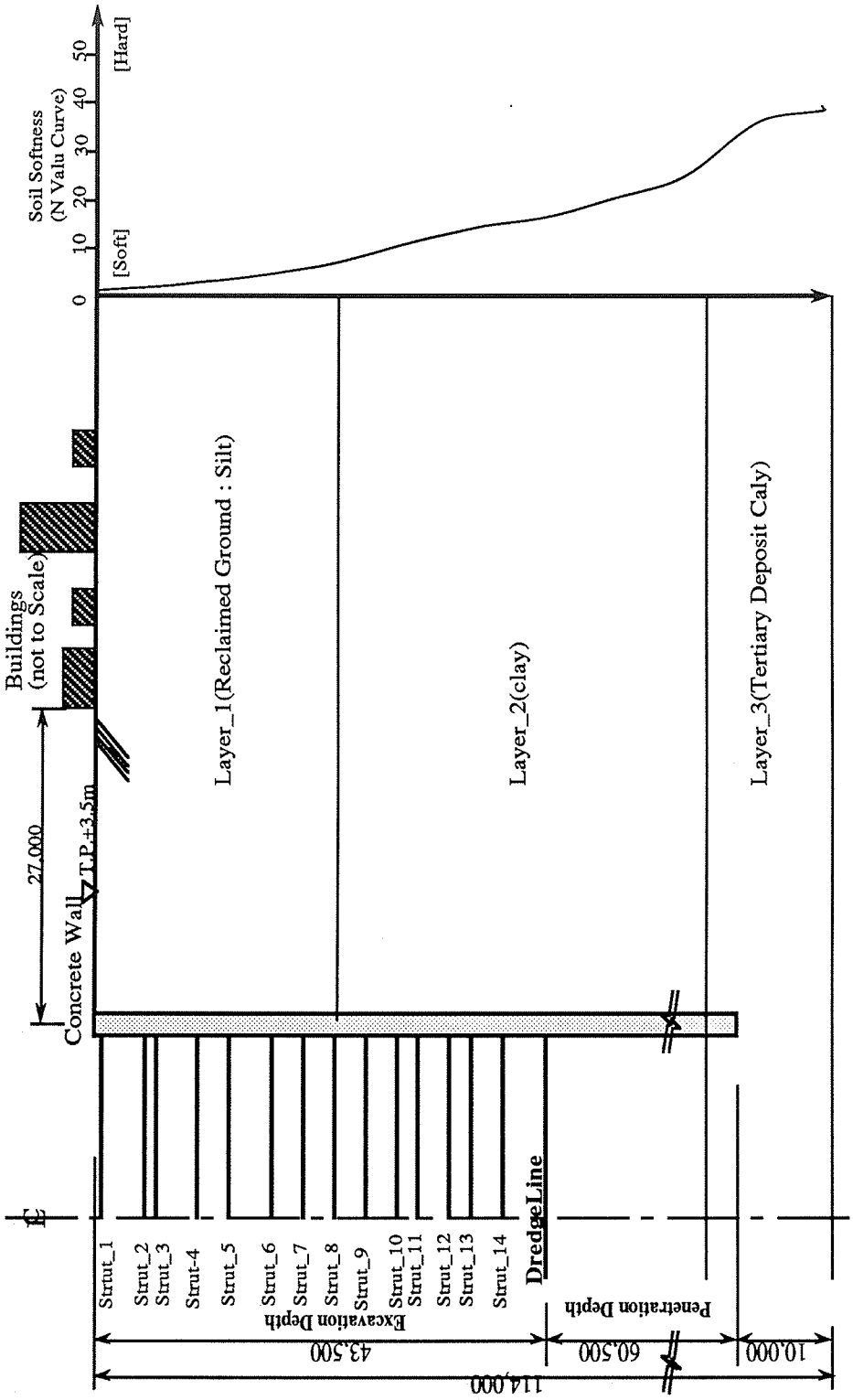


Fig-3.18 Pumping Plant - Elevation View
(Test Case_1 Used in the Research)
The soil condition is not good and near the wall there is a redidence area. The excavation depth and penetration depth is very deep.

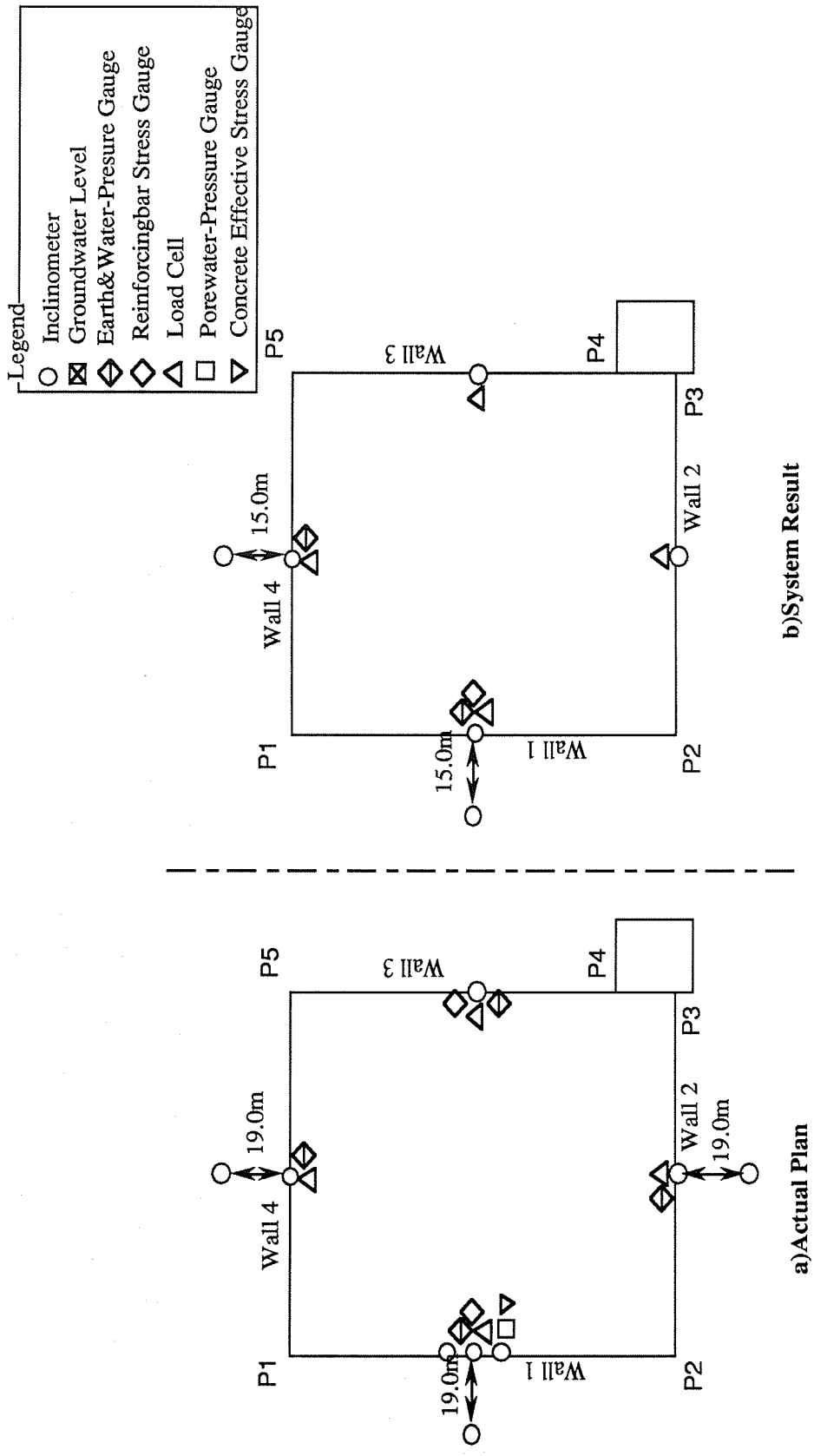


Fig-3.19 Comparison of the results (Test Case_1 : Plan View)
The differences between the actual plan and system results are some. These additional measurement points were added mainly for soil research in the actual plan. The system only recommended measurement points necessary for safety and good practice.

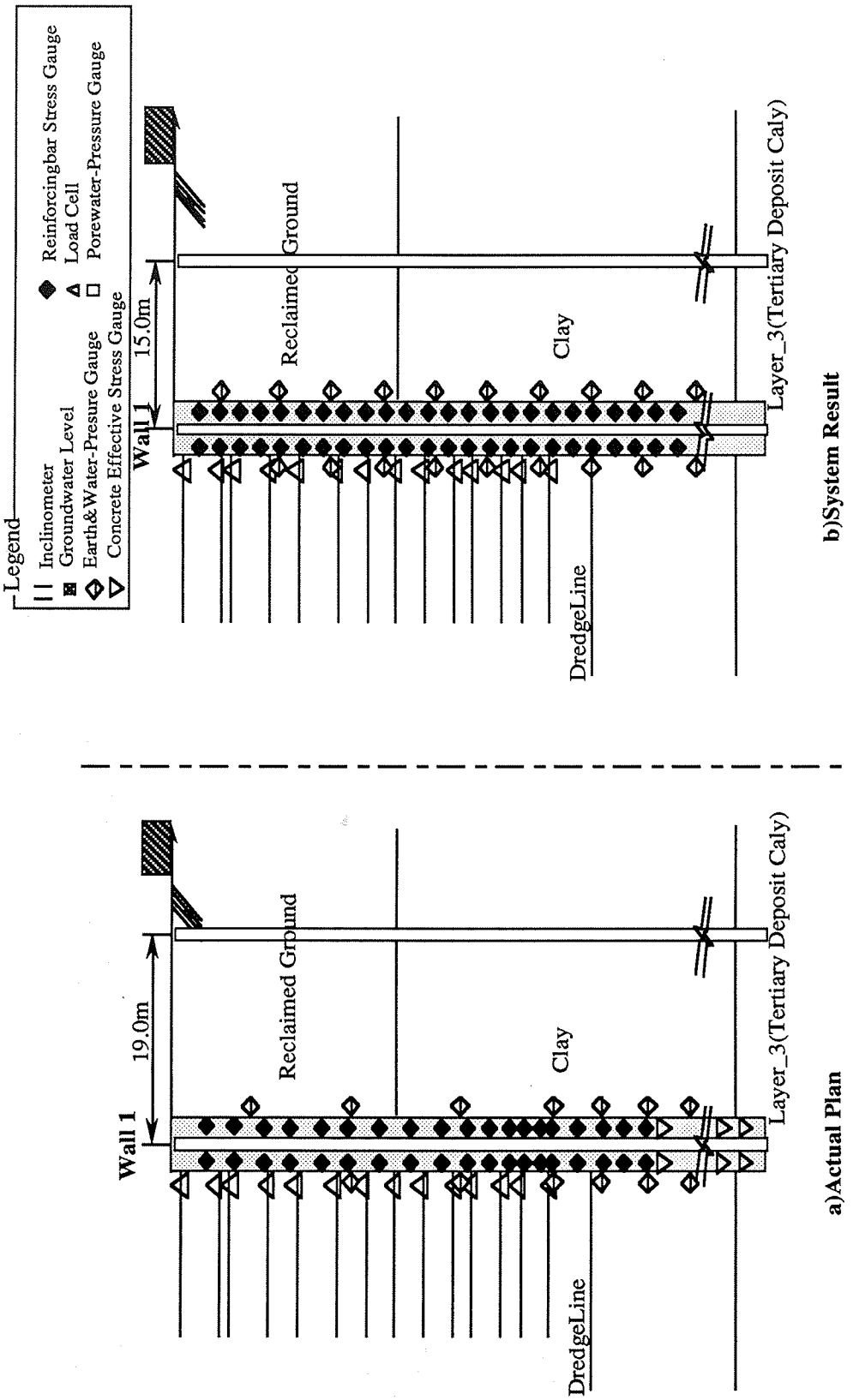


Fig-3.20 Compariсon of the result(Test Case_1 : Elevation View)

The differences between the actual and system results are

1. Distance to inclinometer (Unimportant in practice)
2. Use of concrete stress gauges (Special case not yet considered in system)
3. Pitch of reinforcingbar stress gauges (not yet considered in system)

Fig-3.21 showed plan view and Fig-3.22 illustrates representative elevation view of excavation site for a wastewater treatment plant. In this case the site has relatively good soil condition, but in the alluvium a slip surface exists. This is a test case for a site with a slip surface.

2) Output Result

Fig-3.23 and -3.24 shows the results of this system. From the point of plan view, the following equipments are lacked in the result of the system.

Wall_2 : Two Inclinometers behind the wall,

Wall_3 : One Inclinometer in the ground behind wall

Wall_4 : One Inclinometer in the ground behind wall

From the point of elevation view, the difference between the two is that the measurement points of the reinforcing bar stress gauges. In the actual case, the reinforcing bar stress gauge was located just around the point where slipsurface is facing. The system, however, doesn't think about the that point and put the reinforcing bar stress gauge from top to bottom.

3) Contemplation

The system recommended many more reinforcing bar stress gauges than the actual plan. Indeed in terms of cost, the actual case is preferable. But in terms of from engineering e.g. because is difficult to measure the position of slipsurface or slipsurface which might cause another corruption in the ground etc.., these measurement points the system presented have good engineering value. The system does not do cost-benefit analysis yet.

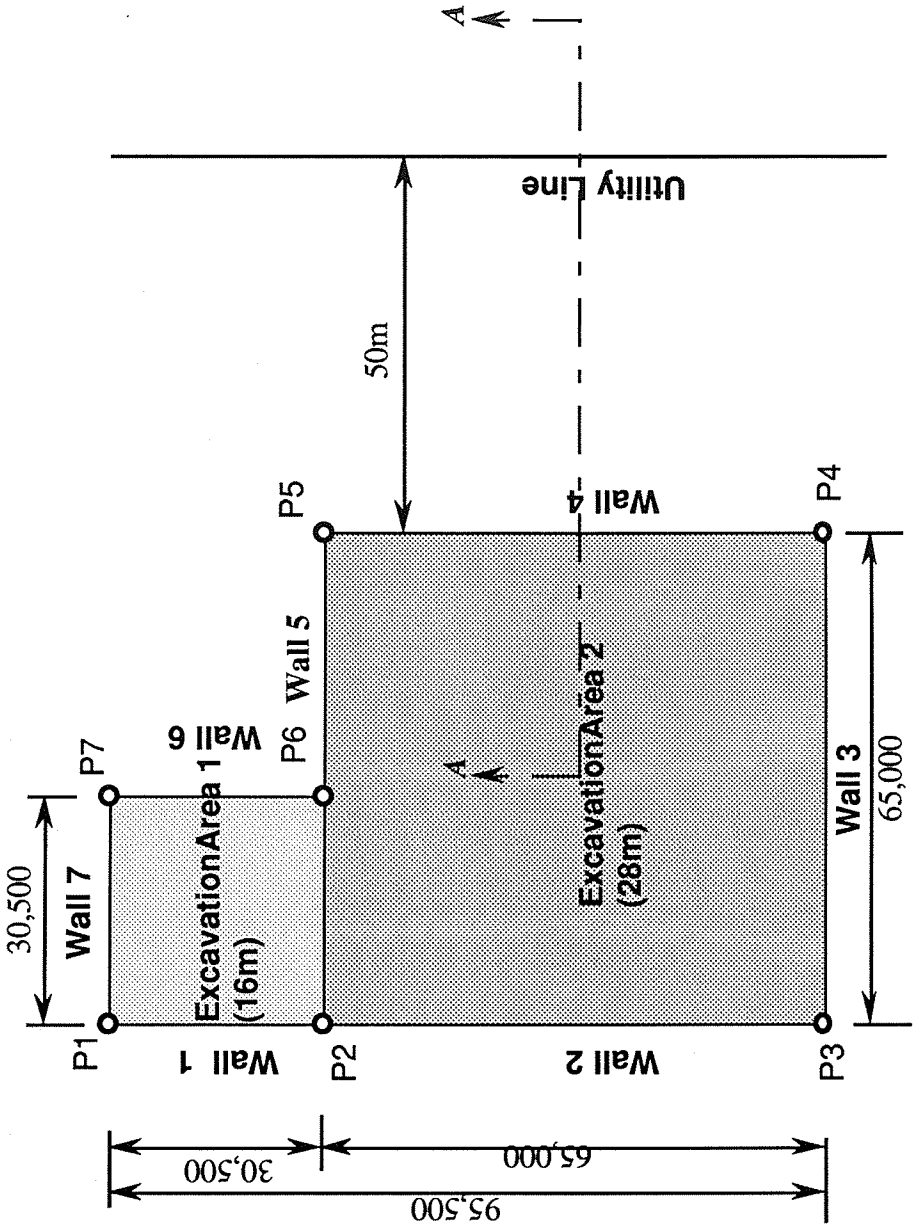


Fig-3.21 Wastewater Treatment Plant -Plan View
(Test Case_2 Used in the Research)

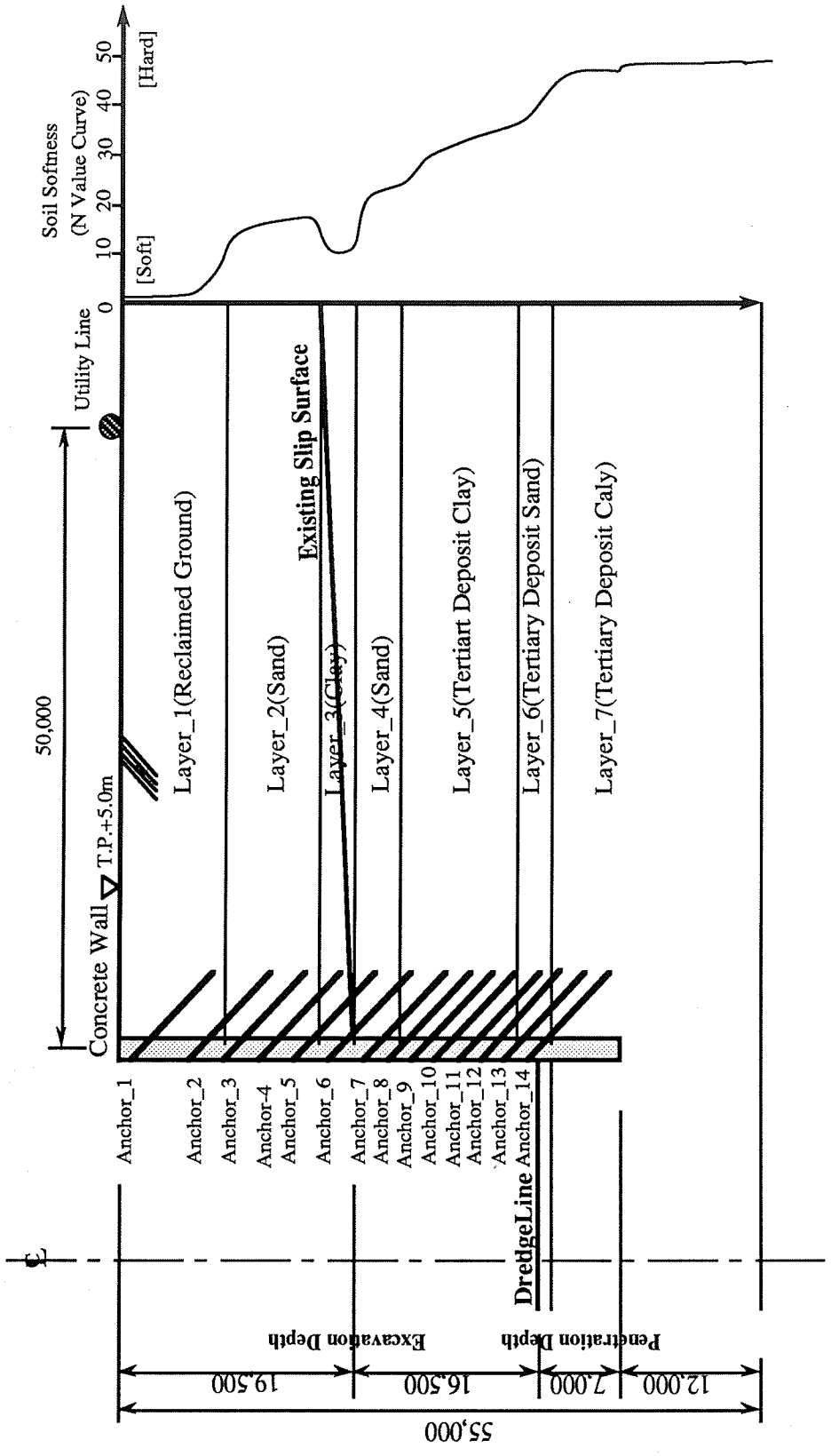


Fig-3.22 Wastewater Treatment Plant - Elevation View
(Test Case_2 Used in the Research)
The soil condition is good but there is a slipsurface behid wall.
The existing facility is not near the wall.

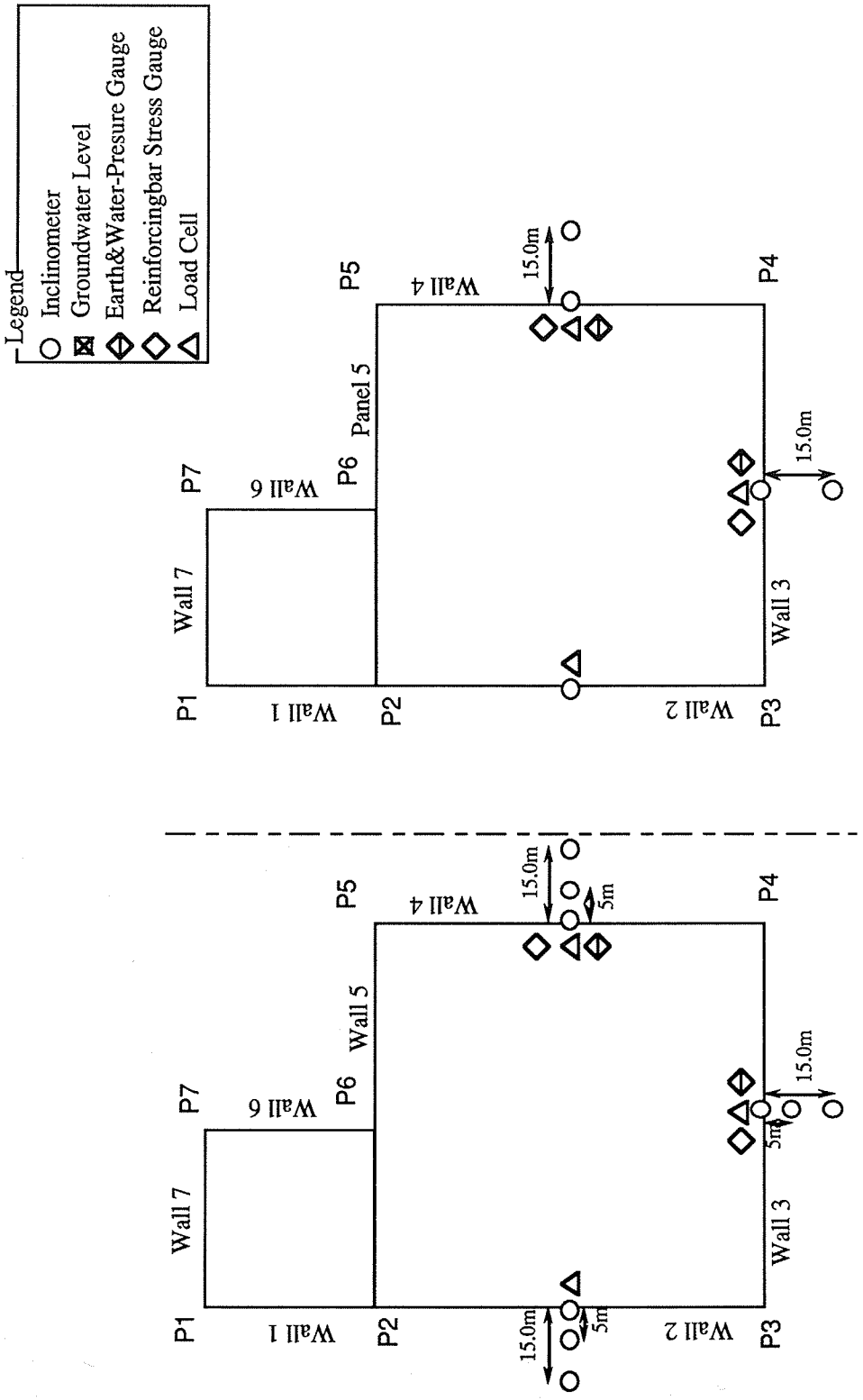


Fig-3.23 Comparison of the results (Test Case_2 : Plan View)
The differences between the actual plan and system results are the number of pieces of inclinometer. It would be decided by expert opinion how much of a problem the slipsurface would cause.

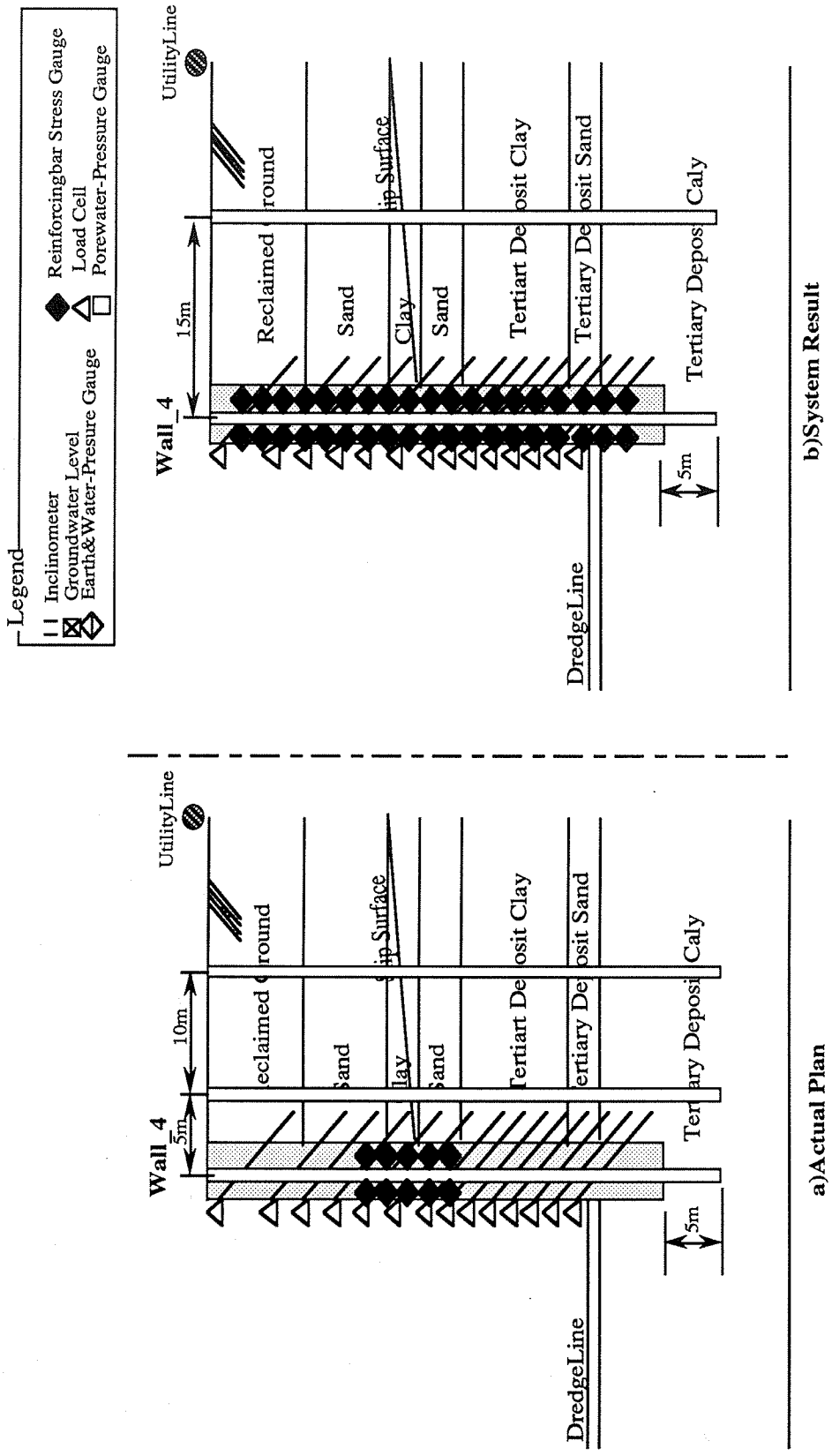


Fig-3.24 Comparison of the results (Test Case_2 : Elevation View)
The differences between the actual plan and system results are measurement points of reinforcing bar stress gauges. The system recommended many more points than the actual plan. These measurements have good engineering value. The system doesn't do cost-benefit analysis.

3.8.3 Conclusion

It became evident that the result from this system doesn't always coincide with the actual case, because of many specific opinions, such as budget issue or academic interest, involved in the actual case. These problems are often involved in the planning type system. It is difficult to take into account all of specific situations which exists in the actual complicated project and to build the knowledge based system to also resolve that kind of issue. In this area, the interactive relationship between expert and system must be needed. The following illustration explains these problems well (Fig-3.25). In the figure the curve means the relation between the quantity and difficulty of the expert's task. In short, the less the difficulty of task tasks, the more the quantity of the tasks. It means that if knowledge system replace the easy parts of the expert's task, the total time expert have to spend for the task would be reduced much and he can deal with another tasks with same amount of his time. In it the purpose of the knowledge system to help the productivity of expert's task and make

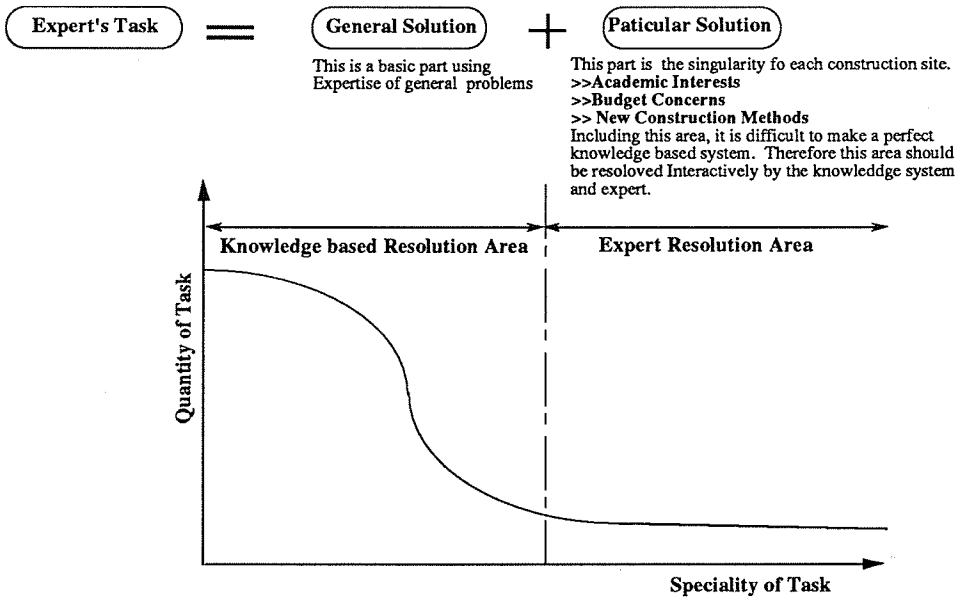


Fig-3.25 Relation between expert and knowledge system

In the construction industry each project, in particular infrastructure, has singularity unlike the manufacturing industry. In that point the knowledge based system has a limitation, like how much knowledge it should cover. Therefore a general part of project would be done by knowledge based system and some specific part of project would be resolved by communication between system and experts. The Knowledge based system is located as expert-supporting system.

it higher . Therefore in the construction industry where we often meet special case, knowledge base system has been developed as expert-supporting system. In order to make this procedure more efficient, the good interface will be necessary. This seem to be one of reasons the Intelligent CAD system has been making progress recently.

4. Conclusion

Listed below are conclusions of this research.

1) The methodology of this system was validated by comparing with actual cases. Moreover through this research activity, it was confirmed that the model-based reasoning by using object-oriented paradigm and rule-based reasoning has flexibility and extensibility. That is because it is composed of many objects and each object has explicit and implicit way s of communicating with each other and explaining its own functions. The model based reasoning is more powerful than I had expected, and also explains well this geotechnical domain . This attempt was a big step fowords the integration of Observational Construction Systems. Moreover this idea would be used in the geotechnical domain as qualitative reasoning and quantitative reasoning in order to resolve uncertainties in this field.

2) As mentioned in section 3.8, there is indeed a gap between the actual project and knowledge system. In this construction industry, unlike the manufacturing industry, each project the engineers face on is not same. Some originality or special issues engineers must solve one by one. Therefore, it is obvious that in the near future that system contents must be explicitly explained like Object Oriented paradigm, and good intelligent CAD system which make engineers able to easily communicate and exchange ideas and opinions with system will a ppear.

3). As a result of this research activity, and through investigations, site inspections and literatures, done beside this research, I believe that Observational Construction is not very important in the United States. In particular in CIFE there is no research focusing on this kind of issue. Therefore this research provides a good clue for the next step in the future of CIFE.

5. Future Projects

In order to achieve the integration of observational construction systems, the followings are next obstacles.

1) Case Based Reasoning

In the construction industry, a huge amount of past data exists. Recently, each company is trying to built a technical data-base. It needs to establish a methodology to acquire knowledge directly from the data-base. Although many researchers are struggling with this approach, no good result have yet appeared. Some research activities using frame-based reasoning are reported [Riesbeck 89][Ichioka 91].

2) Real-Time processing reasoning

This is also a challenging type of reasoning and is necessary for the monitoring system. The research using black board reasoning is reported and adapted to medical science as monitoring system of patient [Hayes-Roth 89]. In the case of Observational Construction, it doesn't require the short time interval such as a couple of seconds or minutes, but it has to deal with huge amount of data at once. In the future, in consideration of the shortage of experts, data interpretation and decision making must be systematized using real-time processing reasoning.

3) Interactive Knowledge Exchanging System

In order to bridge the gap between knowledge based system and actual project discussed in section 3.8, a interactive user interface to obtain the opinion of expert is necessary. Intelligent CAD system is one of them. But for observational construction system, more flexible graphic system should be needed, because the object is not a beam or a column or a window etc. but soil, slipsurface and ground water.

6. References

[Bowles 88] Bowles.J.E,"Foundation Analysus and Design",1988

[Chang 89] Chang and Hoque " A Knowledge-Based Simulation System for Construction Process Planning ",*Proceedings of the Sixth International Sysmposium on Robotics and Artificial Intelligent in Building Construction*,1989.

[Chang 91] Chang " An Object-Oriented Simulation System for Construction Process Planning ", *Proceedings on the American Society of Civil Engineers Construction Congress*,April 1991.

[Dym 91] Dym,C.L., and Levitt,R.E.," " *Knowledge-Based Systems in Engineering*,1991

[Fikes 85] Fikes,R., and Kehler.T.,"The Role of Frame-BasedRepresentation in Reasoning",September 1985.

[Fischer 91] Fischer,M.A.,"Using Construction Knowledge During Preliminary Design of Reinforced Concrete Structure",A Dissertation in Stanford Univ. at the Department of Civil Engineering,1991

[Intelli 88] IntelliCorp., "Model-based reasoning in KEE and SimKit Systems", *IntelliNews*,August 1986.

[Kunz 88] Kunz " Model Based Reasoning in CIM " in *Intelligent Manufacturing: Expert Systems and the Leading Edge in Production Planning and Control*,Addison Wesley,1988.

[Kunz 89] Kunz,J.C., Stelzner and illiams " From Classic Expert Systems to Models: Introduction to a Metholodology for Building Model-Based Systems ",1989

[Levitt 85] Levitt and Kunz " Using Knowledge of Construction and Project Management for Automated Scedule Updating " *Project Management*

Journal,Pages 57-81, December 1985.

[Levitt 87] Levitt,R.E.," Expert Systems in Construction : State of the Art" *Expert Systems for Civil Engineers:Technology and application, Chapter 5*,1987

[Riesbeck 89] Riesbeck,C.K., andShank,R.C.,"Inside Case-Based Reasoning", *Lawrence Erlbaum Associates,Inc.*,1989

[Siller 87] Siller " Expert Systems in Geotechnical Engineering" *Expert Systems for Civil Engineers:Technology and application, Chapter 5*,1987

[Stefik 86] Stefik and Bobrow " Object-Oriented Programming : Themes and Variations " *The AI Magazine*,1986

[Suzuki 90] Suzuki, Koyama and Kumano " Present Status of Expert System in Construction Companies ", *Proceedings on the Center for Integrated Facility Engineering Symposium*, March 1990.

[Zweben 91] Zweben, Davis and Deale " An Empirical Study of Rescheduling Using Constraint-Based Simulated Annealing ",April 1991.

Appendix 1
(Rule List)

```
/*********************************************************/
/**          ALL RULES ARE SAVED BELOW                  **/
/*********************************************************/

/************************************
****  RULE: SlipDepth
************************************/
MakeRule( SlipDepth, [x|MeasurePoint y|SlipSurface],
x:FailurePattern #= NotDecided And x:PanelName #= y:MeasurementPoint,
{
SetValue( x:SlipSurface, y );
SetValue( x:SlipSurfaceDepth, y:SurfaceDepth );
} );

/************************************
****  RULE: WallFailure
************************************/
MakeRule( WallFailure, [x|MeasurePoint],
{
Let [p x:PanelName]
{
Not( x:SlipSurface #= None ) And x:SlipSurfaceDepth >=
p:CoordinateZ2 And x:FailurePattern #= NotDecided;
};
},
{
SetValue( x:SurfacePosition, Behind );
SetValue( x:MeasurementType, WallFailure:MeasurementType );
SetValue( x:FailurePattern, WallFailure );
{
ForAll [ y|StateBox ]
UpdateImage( y );
};
PostMessage( " At this " # x # "  " # x:FailurePattern # " is predicted."
# " So we need the Measurement " # x:MeasurementType );
} );

/************************************
****  RULE: AddInclino2
************************************/
MakeRule( AddInclino2, [x|MeasurePoint],
{
( x:FacilityDistance < 200 ) And {
( x:MeasurementType #= Type2 )
Or ( x:MeasurementType
#= Type3 );
} And ( x:FacilityDistance
< x:ExcavationDepth )
And ( x:FacilityDistance > 0.0 );
},
{
SetValue( x:AddNumInclino, 2 );
PostMessage( x # "  needs to be added 2 more inclinometers." );
} );

/************************************
****  RULE: AddInclino1
************************************/
```

```
MakeRule( AddInclino1, [x|MeasurePoint],
{
x:FacilityDistance < 200 And {
( x:MeasurementType #= Type2 )
Or ( x:MeasurementType #= Type3 );
} And {
x:FacilityDistance >= x:ExcavationDepth;
};
},
{
SetValue( x:AddNumInclino, 1 );
PostMessage( x # "  needs to be added 1 more inclinometer." );
} );

/************************************
****  RULE: SettlementCheck
************************************/
MakeRule( SettlementCheck, [x|MeasurePoint],
{
Let [p x:PanelName]
{
x:MeasurementType #= NotDecided And p:NValueCurve #= Low
And p:LayerCondition #= Bad;
};
},
{
SetValue( x:MeasurementType, Settlement:MeasurementType );
SetValue( x:FailurePattern, Settlement );
{
ForAll [ y|StateBox ]
UpdateImage( y );
};
PostMessage( " At this " # x # "  " # x:FailurePattern # " is predicted."
# " So we need the Measurement " # x:MeasurementType );
} );

/************************************
****  RULE: HeavingCheck
************************************/
MakeRule( HeavingCheck, [x|MeasurePoint],
{
Let [p x:PanelName]
{
x:MeasurementType #= NotDecided And p:NValueCurve #= Low
And p:LayerCondition #= Good And p:EdgeCondition #=
Bad;
};
},
{
SetValue( x:FailurePattern, Heaving );
SetValue( x:MeasurementType, Heaving:MeasurementType );
{
ForAll [ y|StateBox ]
UpdateImage( y );
};
PostMessage( " At this " # x # "  " # x:FailurePattern # " is predicted."
# " So we need the Measurement " # x:MeasurementType );
} );
```

```
/************************************
****  RULE: NoneFailure
************************************/
MakeRule( NoneFailure, [x|MeasurePoint],
{
Let [p x:PanelName]
{
x:SlipSurface #= None And x:MeasurementType #= NotDecided
And Not( p:NValueCurve #= Low And p:LayerCondition #=
Bad ) And Not( p:NValueCurve #= Low And p:LayerCondition
#= Good And p:EgdeCondition
#= Bad );
};
},
{
SetValue( x:MeasurementType, Type1 );
SetValue( x:FailurePattern, None );
{
ForAll [ y|StateBox ]
UpdateImage( y );
};
PostMessage( " At this " # x # "  " # x:FailurePattern # " is predicted."
# " So we need the Measurement " # x:MeasurementType );
} );

/************************************
****  RULE: HeavingCheck2
************************************/
MakeRule( HeavingCheck2, [x|MeasurePoint],
{
Let [p x:PanelName]
{
( x:MeasurementType #= ( TatalSliding Or WallFailure ) )
And p:NValueCurve #= Low And p:LayerCondition #= Bad
And p:EdgeCondition #= Bad;
};
},
{
SetValue( x:Others, Heaving );
PostMessage( " At  " # x # "  " # x:Others # " is also predicted."
# " So we need to pay attention to " );
} );

/************************************
****  RULE: NoSlipSurface
************************************/
MakeRule( NoSlipSurface, [x|MeasurePoint],
x:SlipSurface #= None,
{
PostMessage( "There is no SlipSurface in " # x # ".  So we need more investigation here" );
SetValue( x:MeasurementType, NotDecided );
SetValue( x:FailurePattern, NotDecided );
SetValue( x:SurfacePosition, None );
} );

/************************************
****  RULE: TotalSliding
************************************/
MakeRule( TotalSliding, [x|MeasurePoint],
```

```
{
Let [p x:PanelName]
{
Not( x:SlipSurface #= None ) And x:SlipSurfaceDepth <
p:CoordinateZ2 And x:FailurePattern #= NotDecided;
};
},
{
SetValue( x:SurfacePosition, Under );
SetValue( x:MeasurementType, TotalSliding:MeasurementType );
SetValue( x:FailurePattern, TotalSliding );
{
ForAll [ y|StateBox ]
UpdateImage( y );
};
PostMessage( " At this " # x # "   " # x:FailurePattern # " is predicted."
# " So we need the Measurement " # x:MeasurementType );
} );

/************************************
****  RULE: EdgeSoilCheck
************************************/
MakeRule( EdgeSoilCheck, [x|Panel],
{
ForAll [ y|Layer ]
{
x:MeasurePoint #= Center And x:PriorityOrder <= ProjectOutline:NumMeasurePoint
And x:EdgeElevation > y:CoordinateZ2 And x:EdgeElevation
< y:CoordinateZ1;
};
},
{
SetValue( x, EdgeLayer, y:ID );
SetValue( x, EdgeSoil, y:SoilKind );
} );
SetRulePriority( EdgeSoilCheck, 1 );

/************************************
****  RULE: LongPenatration
************************************/
MakeRule( LongPenatration, [x|Panel],
{
Let [p x:ExcavationDepth / 10.0]
{
x:PenetrationDepth > p And x:MeasurePoint #= Center And
x:PriorityOrder <= ProjectOutline:NumMeasurePoint;
};
},
SetValue( x:PenetrationCondition, Enough ) );

/************************************
****  RULE: ShortPenetration
************************************/
MakeRule( ShortPenetration, [x|Panel],
{
Let [p x:ExcavationDepth / 10.0]
{
x:PenetrationDepth <= p And x:MeasurePoint #= Center And
x:PriorityOrder <= ProjectOutline:NumMeasurePoint;
```

```
};
},
SetValue( x:PenetrationCondition, Short ) );

/***********************************
****  RULE: CheckLayerCondition
***********************************/
MakeRule( CheckLayerCondition, [x|Panel],
{
x:MeasurePoint #= Center And x:PriorityOrder <= ProjectOutline:NumMeasurePoint
Or {
ForAll [ y|Layer ]
{
x:ExcavationElevation <= y:CoordinateZ1 And y:SoilKind
#= Sand;
};
};
},
SetValue( x, LayerCondition, Good ) );

/***********************************
****  RULE: NValueCurve
***********************************/
MakeRule( NValueCurve, [x|Panel],
{
x:MeasurePoint #= Center And x:PriorityOrder <= ProjectOutline:NumMeasurePoint
And {
ForAll [ y|Layer ]
{
x:ExcavationElevation <= y:CoordinateZ1 And {
y:SoilKind
#=
Clay
And
y:NValue
<=
15.0;
}
Or {
y:SoilKind #= Sand And y:NValue <= 30.0;
};
};
};
},
SetValue( x, NValueCurve, Low ) );

/***********************************
****  RULE: NValueCurve2
***********************************/
MakeRule( NValueCurve2, [x|Panel],
{
x:MeasurePoint #= Center And x:PriorityOrder <= ProjectOutline:NumMeasurePoint
And {
ForAll [ y|Layer ]
{
x:ExcavationElevation <= y:CoordinateZ1 And {
y:SoilKind
#=
Clay
```

```
And
y:NValue
>=
15.0;
}
Or {
y:SoilKind #= Sand And y:NValue >= 30.0;
};
};
};
},
SetValue( x, NValueCurve, High ) );

/************************************
****  RULE: EdgeCondition
************************************/
MakeRule( EdgeCondition, [x|Panel],
{
Let [EL x:EdgeLayer]
{
x:EdgeSoil #= Clay And EL:NValue < 30.0 And x:MeasurePoint
#= Center And x:PriorityOrder <= ProjectOutline:NumMeasurePoint;
};
},
SetValue( x, EdgeCondition, Bad ) );

/************************************
****  RULE: HighPorePressure
************************************/
MakeRule( HighPorePressure, [x|Panel],
{
ForAll [ y|Layer ]
{
Let [k x:EdgeLayer]
{
( k:SoilStrengthC * x:PenetrationDepth ) < Layer5:WaterPressure
And x:EdgeElevation > y:CoordinateZ2 And x:EdgeElevation
< y:CoordinateZ1 And y:SoilKind #= Clay And y:NValue
< 30.0 And x:MeasurePoint #= Center And x:PriorityOrder
<= ProjectOutline:NumMeasurePoint;
};
};
},
SetValue( x, EdgePorePressure, High ) );

/************************************
****  RULE: BadLayerCondition
************************************/
MakeRule( BadLayerCondition, [x|Panel],
{
x:MeasurePoint #= Center And x:PriorityOrder <= ProjectOutline:NumMeasurePoint
And {
ForAll [ y|Layer ]
{
x:ExcavationElevation <= y:CoordinateZ1 And y:SoilKind
#= Clay;
};
};
},
```

```
SetValue( x, LayerCondition, Bad ) );

/************************************
****  RULE: GoodEdgeCondition
************************************/
MakeRule( GoodEdgeCondition, [x|Panel],
{
Let [EL x:EdgeLayer]
{
x:EdgeSoil #= Clay And EL:NValue > 30.0 And x:MeasurePoint
#= Center And x:PriorityOrder <= ProjectOutline:NumMeasurePoint;
};
},
SetValue( x, EdgeCondition, Good ) );

/************************************
****  RULE: GoodLayerCondition
************************************/
MakeRule( GoodLayerCondition, [x|Panel],
{
x:EdgeSoil #= Sand And x:MeasurePoint #= Center And x:PriorityOrder
<= ProjectOutline:NumMeasurePoint;
},
SetValue( x, EdgeCondition, Good ) );

/************************************
****  RULE: MainMPoint
************************************/
MakeRule( MainMPoint, [x|MeasurePoint],
x:FailurePattern #= None And x:PriorityNumber = 1 And {
ForAll [ y|MeasurePoint
]
y:FailurePattern
#=
None;
},
SetValue( x:MeasurementType, Type3 ) );
SetRulePriority( MainMPoint, 5 );

/************************************
****  RULE: MainMPoint2
************************************/
MakeRule( MainMPoint2, [x|MeasurePoint],
x:FailurePattern #= Settlement And x:PriorityNumber > 1,
{
SetValue( x:MeasurementType, Type1 );
PostMessage( "The MeasurementType of  " # x # " is changed to "
# x:MeasurementType );
} );
SetRulePriority( MainMPoint2, 5 );


/*********************************************************/
/**          ALL GOALS ARE SAVED BELOW                  **/
/*********************************************************/
```

Appendix 2
(Function List)

```
/*********************************************************/
/**          ALL FUNCTIONS ARE SAVED BELOW              **/
/*********************************************************/

        /************************************
         ****  FUNCTION: Initialize
         ************************************/
MakeFunction( Initialize, [Object],
    {
    ForAll [ x|Object ]
           {
           ClearList( x:GravityPoint );
           ClearList( x:Point1 );
           ClearList( x:Point2 );
           ClearList( x:Point3 );
           ClearList( x:Point4 );
           {
           ForAll [ z|MeasurePoint ]
                  SetValue( z:FailurePattern, NotDecided );
           };
           {
           ForAll [ y|StateBox ]
                  UpdateImage( y );
           };
           };
    } );

       /************************************
        ****  FUNCTION: PointCoordinate
        ****  X;ObjectName
        ************************************/
MakeFunction( PointCoordinate, [Object],
    {
    ForAll [ x|Object ]
           {
           SetValue( x, Point1, x:CoordinateX1, x:CoordinateY1,
                  x:CoordinateZ1 );
           SetValue( x, Point2, x:CoordinateX2, x:CoordinateY2,
                  x:CoordinateZ1 );
           SetValue( x, Point3, x:CoordinateX2, x:CoordinateY2,
                  x:CoordinateZ2 );
           SetValue( x, Point4, x:CoordinateX1, x:CoordinateY1,
                  x:CoordinateZ2 );
           SetValue( x, GravityPoint, x:GravityPointX, x:GravityPointY );
           };
    } );
SetFunctionComment( PointCoordinate, "X;ObjectName" );

       /************************************
        ****  FUNCTION: NearestPanelSearch
        ************************************/
MakeFunction( NearestPanelSearch, [],
    {
    ForAll [ y|Facility ]
           {
           SetValue( y, DistancePanel, 1000.0 );
           ForAll [ x|Panel ]
                  {
```

```
                Let [distance Sqrt( ( ( y:GravityPointX - x:GravityPointX )
                                    ^ 2 ) + ( ( y:GravityPointY
                                                - x:GravityPointY )
                                              ^ 2 ) )]
                    If ( distance <= y:DistancePanel )
                      Then {
                           SetValue( y, DistancePanel, distance );
                           SetValue( y, NearestPanel, x );
                           }
                      Else  );

       /************************************
        ****  FUNCTION: MeasurePointSearch
        ************************************/
MakeFunction( MeasurePointSearch, [],
    {
    ForAll [ x|Panel ]
           {
           Let [CC ExcavationArea:MaxDepthArea]
               {
               If {
                  ( CC:CoordinateX1 <= x:GravityPointX ) And ( CC:CoordinateX2
                                                               >=
                                                               x:GravityPointX )
                    And ( CC:CoordinateY1 <= x:GravityPointY )
                    And ( CC:CoordinateY2 >= x:GravityPointY );
                  }
                 Then {
                      SetValue( x, ExcavationDepth, CC:MaxDepth );
                      SetValue( x, MeasurePoint, Center );
                      AppendToList( Panel:Priority, x );
                      }
                 Else {
                      SetValue( x, MeasurePoint, None );
                      SetValue( x, ExcavationDepth, Area1:ExcavationDepth );
                      };
               };
           };
    } );

       /************************************
        ****  FUNCTION: SetID
        ************************************/
MakeFunction( SetID, [Object],
    {
    ForAll [ x|Object ]
           {
           SetValue( x:ID, x );
           };
    } );

       /************************************
        ****  FUNCTION: CheckFacility
        ************************************/
MakeFunction( CheckFacility, [],
    {
    ForAll [ y|Panel ]
           {
```

```
            ForAll [ x|Facility ]
                   {
                   If ( y:ID #= x:NearestPanel )
                     Then {
                          SetValue( y, ExistFacility, x );
                          SetValue( y, FacilityDistance, x:DistancePanel );
                          };
                   };
            };
    } );

       /************************************
        ****  FUNCTION: SetInitialValue
        ************************************/
MakeFunction( SetInitialValue, [],
    {
    PostMessage( "***Set up ID & Input data***" );
    SetID( Panel );
    SetID( ExcavationArea );
    SetID( Facility );
    SetID( Support );
    SetID( Layer );
    SetID( SlipSurface );
    SetInitialValuePanel( );
    SetInitialValueExcavationArea( );
    SetInitialValueFacility( );
    SetInitialValueLayer( );
    SetInitialValueSupport( );
    SetInitialValueSlipSurface( );
    SetInitialValueMeasurePoint( );
    PostMessage( "***For Geometric Reasoning : Coordinate Caliculation***" );
    PointCoordinate( Panel );
    PointCoordinate( ExcavationArea );
    PointCoordinate( Facility );
    PostMessage( "***InputData-Checking***" );
    {
    If ( PostMenu( "Do you wish tocheckthe input-data ?", Yes, No )
           #= Yes )
      Then CheckInputData( );
    };
    } );

       /************************************
        ****  FUNCTION: GetPriorityNumber
        ************************************/
MakeFunction( GetPriorityNumber, [],
    {
    ForAll [ x|Panel ]
           {
           If ( x:MeasurePoint #= Center )
             Then {
                  If ( x:PriorityNumber #= TRUE )
                    Then PostMessage( "The PriorityNumber of " #
                                      x # " is low !!!" )
                    Else PostMessage( "The PriorityNumber of " #
                                      x # " is " # x:PriorityNumber );
                  };
           };
```

```
    } );

        /************************************
         ****  FUNCTION: ZeroInitialize
         ************************************/
MakeFunction( ZeroInitialize, [],
    {
    PostMessage( "*** Initialization***" );
    Initialize( Panel );
    Initialize( ExcavationArea );
    Initialize( Facility );
    DelIns( Inclinometer );
    DelIns( LoadCell );
    DelIns( StressGauge );
    DelIns( EPressGauge );
    DelIns( WPressGauge );
    DelIns( SettleGauge );
    } );

        /************************************
         ****  FUNCTION: MaxExcavationSearch
         ************************************/
MakeFunction( MaxExcavationSearch, [],
    {
    SetValue( ExcavationArea, MaxDepth, 0.0 );
    ForAll [ x|ExcavationArea ]
           {
           If ( x:ExcavationDepth >= ExcavationArea:MaxDepth )
             Then {
                  SetValue( ExcavationArea, MaxDepth, x:ExcavationDepth );
                  SetValue( ExcavationArea, MaxDepthArea, x );
                  }
             Else  );

        /************************************
         ****  FUNCTION: SetMeasurementPoint
         ************************************/
MakeFunction( SetMeasurementPoint, [],
    {
    PostMessage( "***The Deepest Excavtion-Area Search ***" );
    MaxExcavationSearch(  );
    PostMessage( "***The Nearest Facility-Panel Relation Search***" );
    NearestPanelSearch(  );
    PostMessage( "***Measurement Point Search***" );
    MeasurePointSearch(  );
    CheckFacility(  );
    GetOrder(  );
    SetPriorityOrder(  );
    } );

        /************************************
         ****  FUNCTION: GetOrder
         ************************************/
MakeFunction( GetOrder, [],
    {
    ForAll [ x|Panel ]
           {
           SetValue( x:PriorityOrder, 1 );
```

```
            {
            If ( x:MeasurePoint #= Center )
              Then {
                   Let [Priority x:PriorityNumber]
                       {
                       ForAll [ y|Panel ]
                              {
                              If ( Priority < y:PriorityNumber )
                                Then SetValue( x, PriorityOrder,
                                               x:PriorityOrder
                                                 + 1 );
                              };
                       };
                   };
            };
            };
    } );

       /************************************
        ****  FUNCTION: SetInitialValueLayer
        ************************************/
MakeFunction( SetInitialValueLayer, [],
    {
    SetValue( Layer1:NValue, 3.0 );
    SetValue( Layer1:Poisson, 0.5 );
    SetValue( Layer1:SoilKind, Clay );
    SetValue( Layer1:SoilStrengthC, 2.0 );
    SetValue( Layer1:SoilStrengthFai, 10.0 );
    SetValue( Layer1:UnitWeight, 1.8 );
    SetValue( Layer1:WaterPressure, 0.0 );
    SetValue( Layer1:CoordinateX1, 50.0 );
    SetValue( Layer1:CoordinateX2, 150.0 );
    SetValue( Layer1:CoordinateY1, 50.0 );
    SetValue( Layer1:CoordinateY2, 50.0 );
    SetValue( Layer1:CoordinateZ1, 10.0 );
    SetValue( Layer1:CoordinateZ2, 5.0 );
    SetValue( Layer2:NValue, 20.0 );
    SetValue( Layer2:Poisson, 0.3 );
    SetValue( Layer2:SoilKind, Sand );
    SetValue( Layer2:SoilStrengthC, 0.0 );
    SetValue( Layer2:SoilStrengthFai, 30.0 );
    SetValue( Layer2:UnitWeight, 2.0 );
    SetValue( Layer2:WaterPressure, 0.0 );
    SetValue( Layer2:CoordinateX1, 50.0 );
    SetValue( Layer2:CoordinateX2, 150.0 );
    SetValue( Layer2:CoordinateY1, 50.0 );
    SetValue( Layer2:CoordinateY2, 50.0 );
    SetValue( Layer2:CoordinateZ1, 5.0 );
    SetValue( Layer2:CoordinateZ2, 0.0 );
    SetValue( Layer3:NValue, 10.0 );
    SetValue( Layer3:Poisson, 0.5 );
    SetValue( Layer3:SoilKind, Clay );
    SetValue( Layer3:SoilStrengthC, 2.0 );
    SetValue( Layer3:SoilStrengthFai, 10.0 );
    SetValue( Layer3:UnitWeight, 1.8 );
    SetValue( Layer3:WaterPressure, 0.0 );
    SetValue( Layer3:CoordinateX1, 50.0 );
    SetValue( Layer3:CoordinateX2, 150.0 );
```

```
    SetValue( Layer3:CoordinateY1, 50.0 );
    SetValue( Layer3:CoordinateY2, 50.0 );
    SetValue( Layer3:CoordinateZ1, 0.0 );
    SetValue( Layer3:CoordinateZ2, -10.0 );
    SetValue( Layer4:NValue, 30.0 );
    SetValue( Layer4:Poisson, 0.3 );
    SetValue( Layer4:SoilKind, Sand );
    SetValue( Layer4:SoilStrengthC, 2.0 );
    SetValue( Layer4:SoilStrengthFai, 10.0 );
    SetValue( Layer4:UnitWeight, 2.0 );
    SetValue( Layer4:WaterPressure, 20.0 );
    SetValue( Layer4:CoordinateX1, 50.0 );
    SetValue( Layer4:CoordinateX2, 150.0 );
    SetValue( Layer4:CoordinateY1, 50.0 );
    SetValue( Layer4:CoordinateY2, 50.0 );
    SetValue( Layer4:CoordinateZ1, -10.0 );
    SetValue( Layer4:CoordinateZ2, -15.0 );
    SetValue( Layer5:NValue, 50.0 );
    SetValue( Layer5:Poisson, 0.4 );
    SetValue( Layer5:SoilKind, Tertiary );
    SetValue( Layer5:SoilStrengthC, 2.0 );
    SetValue( Layer5:SoilStrengthFai, 10.0 );
    SetValue( Layer5:UnitWeight, 2.2 );
    SetValue( Layer5:WaterPressure, 0.0 );
    SetValue( Layer5:CoordinateX1, 50.0 );
    SetValue( Layer5:CoordinateX2, 150.0 );
    SetValue( Layer5:CoordinateY1, 50.0 );
    SetValue( Layer5:CoordinateY2, 50.0 );
    SetValue( Layer5:CoordinateZ1, -15.0 );
    SetValue( Layer5:CoordinateZ2, -25.0 );
    } );

       /************************************
        ****  FUNCTION: CheckInputData
        ************************************/
MakeFunction( CheckInputData, [],
    {
    ForAll [ x|Panel ]
           {
           PostInputForm( "Please check information about " # x,
                          x, CoordinateX1, "X1-Coordinate:", x, CoordinateX2,
                          "X2-Coordinate:", x, CoordinateY1, "Y1-Coordinate:",
                          x, CoordinateY2, "Y2-Coordinate:", x, CoordinateZ1,
                          "TopElevation(m):", x, CoordinateZ2, "EdgeElevation(m):" );
           };
    ForAll [ x|ExcavationArea ]
           {
           PostInputForm( "Please check information about " # x,
                          x, ExcavationDepth, "ExcavationDepth:",
                          x, CoordinateX1, "X1-Coordinate:", x, CoordinateX2,
                          "X2-Coordinate:", x, CoordinateY1, "Y1-Coordinate:",
                          x, CoordinateY2, "Y2-Coordinate:" );
           };
    ForAll [ x|Facility ]
           {
           PostInputForm( "Please check information about " # x,
                          x, FacilityName, "Kind:", x, ImportanceNumber,
                          "How Important (0~10):", x, CoordinateX1,
```

```
                        "X1-Coordinate:", x, CoordinateX2, "X2-Coordinate:",
                        x, CoordinateY1, "Y1-Coordinate:", x, CoordinateY2,
                        "Y2-Coordinate:" );
        };
    ForAll [ x|Layer ]
        {
        PostInputForm( "Please check information about " # x,
                        x, NValue, "N-Value:", x, Poisson, "Poisson Ratio:",
                        x, SoilKind, Soil?, x, UnitWeight, "UnitWeight(t/m3)?",
                        x, WaterPressure, "PorePressure(t/m2):" );
        };
    ForAll [ x|Support ]
        {
        PostInputForm( "Please check information about " # x,
                        x, InitialStress, "InitialLoad(t/m2)?",
                        x, CoordinateZ1, "Set Elevation ( m)" );
        };
    ForAll [ x|SlipSurface ]
        {
        PostInputForm( "Please check information about " # x,
                        x, MeasurementPoint, "Where is this?",
                        x, SurfaceDepth, "Where is a surface at the Point of Panel?" );
        };
    } );

       /************************************
        ****  FUNCTION: SetInitialValueSupport
        ************************************/
MakeFunction( SetInitialValueSupport, [],
    {
    SetValue( Support1:InitialStress, 200.0 );
    SetValue( Support1:CoordinateX1, 100.0 );
    SetValue( Support1:CoordinateX2, 125.0 );
    SetValue( Support1:CoordinateY1, 50.0 );
    SetValue( Support1:CoordinateY2, 50.0 );
    SetValue( Support1:CoordinateZ1, 5.0 );
    SetValue( Support1:CoordinateZ2, -20.0 );
    SetValue( Support2:InitialStress, 250.0 );
    SetValue( Support2:CoordinateX1, 100.0 );
    SetValue( Support2:CoordinateX2, 120.0 );
    SetValue( Support2:CoordinateY1, 50.0 );
    SetValue( Support2:CoordinateY2, 50.0 );
    SetValue( Support2:CoordinateZ1, 0.0 );
    SetValue( Support2:CoordinateZ2, -20.0 );
    SetValue( Support3:InitialStress, 300.0 );
    SetValue( Support3:CoordinateX1, 100.0 );
    SetValue( Support3:CoordinateX2, 115.0 );
    SetValue( Support3:CoordinateY1, 50.0 );
    SetValue( Support3:CoordinateY2, 50.0 );
    SetValue( Support3:CoordinateZ1, -5.0 );
    SetValue( Support3:CoordinateZ2, -20.0 );
    } );

       /************************************
        ****  FUNCTION: SetInitialValueSlipSurface
        ************************************/
MakeFunction( SetInitialValueSlipSurface, [],
    {
```

```
    SetValue( SlipSurface1:MeasurementPoint, Panel3 );
    SetValue( SlipSurface1:CoordinateX1, 100.0 );
    SetValue( SlipSurface1:CoordinateX2, 150.0 );
    SetValue( SlipSurface1:CoordinateY1, 50.0 );
    SetValue( SlipSurface1:CoordinateY2, 50.0 );
    SetValue( SlipSurface1:CoordinateZ1, -10.0 );
    SetValue( SlipSurface1:CoordinateZ2, -20.0 );
    SetValue( SlipSurface2:MeasurementPoint, Panel4 );
    SetValue( SlipSurface2:CoordinateX1, 50.0 );
    SetValue( SlipSurface2:CoordinateX2, 50.0 );
    SetValue( SlipSurface2:CoordinateY1, 50.0 );
    SetValue( SlipSurface2:CoordinateY2, -50.0 );
    SetValue( SlipSurface2:CoordinateZ1, -20.0 );
    SetValue( SlipSurface2:CoordinateZ2, -20.0 );
    SetValue( SlipSurface3:MeasurementPoint, Panel1 );
    SetValue( SlipSurface3:CoordinateX1, 50.0 );
    SetValue( SlipSurface3:CoordinateX2, -50.0 );
    SetValue( SlipSurface3:CoordinateY1, 50.0 );
    SetValue( SlipSurface3:CoordinateY2, 50.0 );
    SetValue( SlipSurface3:CoordinateZ1, 0.0 );
    SetValue( SlipSurface3:CoordinateZ2, 5.0 );
    } );

       /************************************
        ****  FUNCTION: SetPriorityOrder
        ************************************/
MakeFunction( SetPriorityOrder, [],
    {
    ForAll [ x|MeasurePoint ]
           {};
    ForAll [ x|Panel ]
           {
           If {
              ( x:MeasurePoint #= Center ) And ( x:PriorityOrder
                                                 <= ProjectOutline:NumMeasurePoint );
              }
            Then {
                 PostMessage( " The Priority Order of " # x # " is "
                              # x:PriorityOrder # " . " # " The ExcavationDepth is "
                              # x:ExcavationDepth # " & MeasurementPoint is-- "
                              # x:GravityPointX # "  " # x:GravityPointY );
                 SetValue( MeasurePoint # x:PriorityOrder, PanelName,
                           x );
                 SetValue( MeasurePoint # x:PriorityOrder, CoordinateX1,
                           x:GravityPointX );
                 SetValue( MeasurePoint # x:PriorityOrder, CoordinateY1,
                           x:GravityPointY );
                 SetValue( MeasurePoint # x:PriorityOrder, CoordinateX2,
                           x:GravityPointX );
                 SetValue( MeasurePoint # x:PriorityOrder, CoordinateY2,
                           x:GravityPointY );
                 SetValue( MeasurePoint # x:PriorityOrder, PriorityNumber,
                           x:PriorityOrder );
                 SetValue( MeasurePoint # x:PriorityOrder, PanelEdge,
                           x:CoordinateZ2 );
                 SetValue( MeasurePoint # x:PriorityOrder, FacilityDistance,
                           x:FacilityDistance );
                 SetValue( MeasurePoint # x:PriorityOrder, ExcavationDepth,
```

```
                    x:ExcavationDepth );
                SetValue( MeasurePoint # x:PriorityOrder, FailurePattern,
                    NotDecided );
                SetValue( MeasurePoint # x:PriorityOrder, MeasurementType,
                    NotDecided );
                };
        };
    } );

    /************************************
     ****  FUNCTION: SlipSurfaceSearch
     ************************************/
MakeFunction( SlipSurfaceSearch, [],
    {
    ForAll [ x|MeasurePoint ]
        {
        SetValue( x:SlipSurface, None );
        ForAll [ y|SlipSurface ]
            {
            If ( x:PanelName #= y:MeasurementPoint )
              Then {
                  SetValue( x:SlipSurface, y );
                  SetValue( x:SlipSurfaceDepth, y:SurfaceDepth );
                  };
            };
        };
    } );

    /************************************
     ****  FUNCTION: SlipPositionSearch
     ************************************/
MakeFunction( SlipPositionSearch, [],
    {
    {
    ForAll [ x|MeasurePoint ]
        {
        If ( x:SlipSurface #= None )
          Then {
              {
              ForAll [ y|StateBox ]
                    UpdateImage( y );
              };
              PostMessage( "There is no SlipSurface in " # x
                        # ".  So we need more investigation here" );
              SetValue( x:MeasurementType, NotDecided );
              SetValue( x:FailurePattern, NotDecided );
              SetValue( x:SurfacePosition, None );
              }
          Else {
              Let [p x:PanelName]
                  {
                  If ( x:SlipSurfaceDepth < p:CoordinateZ2 )
                    Then {
                        SetValue( x:SurfacePosition, Under );
                        SetValue( x:MeasurementType, TotalSliding:MeasurementType );
                        SetValue( x:FailurePattern, TotalSliding );
                        {
                        ForAll [ y|StateBox ]
```

```
                                    UpdateImage( y );
                                };
                                PostMessage( " At this " # x # "  " #
                                             x:FailurePattern #
                                             " is predicted." # " So we need the
Measurement "
                                             # x:MeasurementType );
                                }
                           Else {
                                SetValue( x:SurfacePosition, Behind );
                                SetValue( x:MeasurementType, WallFailure:MeasurementType );
                                SetValue( x:FailurePattern, WallFailure );
                                {
                                ForAll [ y|StateBox ]
                                       UpdateImage( y );
                                };
                                PostMessage( " At this " # x # "  " #
                                             x:FailurePattern #
                                             " is predicted." # " So we need the
Measurement "
                                             # x:MeasurementType );
                                };
                          };
                     };
               };
     };
     } );

        /************************************
         ****  FUNCTION: PredictFailurePattern
         ************************************/
MakeFunction( PredictFailurePattern, [],
     {
     PostMessage( "***Checking The Posibility Of Sliding Along SlipSurface***" );
     SlipSurfaceSearch(  );
     SlipPositionSearch(  );
     PostMessage( "***Checking The Posibility Of Heaving & Settlement***" );
     CheckLandSoftness(  );
     PostMessage( "***Checking The Necessity Of GroundInclinometer***" );
     InclinometerPosition(  );
     } );

        /************************************
         ****  FUNCTION: NValueSearch
         ************************************/
MakeFunction( NValueSearch, [],
     {
     ForAll [ x|MeasurePoint ]
            {
            If ( x:FailurePattern #= NotDecided )
              Then PostMessage( x );
            };
     } );

        /************************************
         ****  FUNCTION: NValueCheck
         ************************************/
MakeFunction( NValueCheck, [],
```

```
    {
    ForAll [ x|Layer ]
           {
           If ( ( x:NValue <= 20 ) And ( x:XoilKind #= Clay ) )
             Then SetValue( x:NValueCurve, Low )
             Else SetValue( x:NValueCurve, High );
           };
    } );

       /************************************
        ****  FUNCTION: InclinometerPosition
        ************************************/
MakeFunction( InclinometerPosition, [],
    {
    ForAll [ x|MeasurePoint ]
           {
           {
           SetValue( x, AddNumInclino, 0 );
           };
           If ( x:FacilityDistance < 200 )
             Then {
                  If {
                      ( x:MeasurementType #= Type2 ) Or ( x:MeasurementType
                                                          #=
                                                          Type3 );
                    }
                   Then {
                        {
                        If {
                           x:FacilityDistance < x:ExcavationDepth;
                           }
                          Then {
                               SetValue( x, AddNumInclino, 2 );
                               PostMessage( x # "  needs to be added 2 more
inclinometers." );
                               }
                          Else {
                               SetValue( x, AddNumInclino, 1 );
                               PostMessage( x # "  needs to be added 1 more
inclinometer." );
                               };
                        };
                        };
                  };
           };
    } );

       /************************************
        ****  FUNCTION: SetLayerCondition
        ************************************/
MakeFunction( SetLayerCondition, [],
    {
    EdgeConditionCheck(  );
    NValueCurveCheck(  );
    LayerConditionCheck(  );
    } );

       /************************************
```

```
        ****  FUNCTION: CheckLayerCondition
        ************************************/
MakeFunction( CheckLayerCondition, [],
    {
    ForAll [ x|Panel ]
           {
           If {
              ( x:MeasurePoint #= Center ) And ( x:PriorityOrder
                                                  <= ProjectOutline:NumMeasurePoint );
            }
            Then {
                 SetValue( x, LayerCondition, Bad );
                 SetValue( x, NValueCurve, Low );
                 SetValue( x, EdgeCondition, Good );
                 SetValue( x, EdgePorePressure, Low );
                 ForAll [ y|Layer ]
                        {
                        {
                        If {
                           ( x:ExcavationElevation <= y:CoordinateZ1 )
                             And ( y:SoilKind #= Sand );
                           }
                          Then SetValue( x, LayerCondition, Good );
                        };
                        {
                        If {
                           ( ( x:ExcavationElevation <= y:CoordinateZ1 )
                               And ( y:SoilKind #= Clay ) And
                               ( y:NValue >= 15.0 ) ) Or ( ( x:ExcavationElevation
                                                             <=
                                                             y:CoordinateZ1 )
                                                           And
                                                           ( y:SoilKind
                                                               #=
                                                               Sand )
                                                           And
                                                           ( y:NValue
                                                               >=
                                                               30.0 ) );
                           }
                          Then SetValue( x, NValueCurve, High );
                        };
                        {
                        If {
                           ( x:EdgeElevation > y:CoordinateZ2 )
                             And ( x:EdgeElevation < y:CoordinateZ1 )
                             And ( y:SoilKind #= Clay ) And ( y:NValue
                                                              <
                                                              30.0 );
                           }
                          Then {
                               SetValue( x, EdgeCondition, Bad );
                               Let [k x:EdgeLayer]
                                   If ( ( k:SoilStrengthC * x:PenetrationDepth )
                                          < Layer5:WaterPressure )
                                     Then SetValue( x, EdgePorePressure,
                                                    High );
                               };
```

```
                        };
                        };
                PostMessage( x # "  The LayerCondition  is :  "
                             # x:LayerCondition # ".   The NCurveValue  is :  "
                             # x:NValueCurve # ".   The EdgeCondition  is :  "
                             # x:EdgeCondition # . );
                };
        };
    } );

       /************************************
        ****  FUNCTION: CheckHeaving
        ************************************/
MakeFunction( CheckHeaving, [],
    {
    ForAll [ x|MeasurePoint ]
           {
           If ( x:MeasurementType #= NotDecided )
             Then Let [p x:PanelName]
                    {
                    If ( p:NValueCurve #= Low )
                      Then {
                           If ( p:LayerCondition #= Bad )
                             Then {
                                  SetValue( x, MeasurementType,
                                            Type3 );
                                  SetValue( x, FailurePattern,
                                            Settlement );
                                  PostMessage( " At this " # x
                                               # "  " # x:FailurePattern
                                               # " is predicted."
                                               # " So we need the Measurement "
                                               # x:MeasurementType );
                                  {
                                  If ( p:EdgeCondition #= Bad )
                                    Then {
                                         SetValue( x, Others,
                                                   Heaving );
                                         PostMessage( " At this "
                                                      # x
                                                      # "  "
                                                      # x:Others
                                                      # " is also predicted."
                                                      # " So we need to pay attention
to " );
                                         };
                                  };
                                  }
                             Else {
                                  If ( p:EdgeCondition #= Bad )
                                    Then {
                                         SetValue( x, MeasurementType,
                                                   Type4 );
                                         SetValue( x, FailurePattern,
                                                   Heaving );
                                         PostMessage( " At this "
                                                      # x
                                                      # "  "
```

```
                                                    # x:FailurePattern
                                                    # " is predicted."
                                                    # " So we need the Measurement "
                                                    # x:MeasurementType );
                                          };
                                    };
                               }
                          Else {
                               SetValue( x, MeasurementType, Type1 );
                               SetValue( x, FailurePattern, No );
                               };
                     };
          };
    } );

       /************************************
        ****  FUNCTION: CheckEdgeLayer
        ************************************/
MakeFunction( CheckEdgeLayer, [],
    {
    ForAll [ x|Panel ]
          {
          If {
             ( x:MeasurePoint #= Center ) And ( x:PriorityOrder
                                          <= ProjectOutline:NumMeasurePoint );
             }
            Then {
                 ForAll [ y|Layer ]
                        {
                        If ( ( x:EdgeElevation > y:CoordinateZ2 )
                               And ( x:EdgeElevation < y:CoordinateZ1 ) )
                          Then {
                               SetValue( x, EdgeLayer, y:ID );
                               SetValue( x, EdgeSoil, y:SoilKind );
                               };
                        };
                 {
                 Let [p x:ExcavationDepth / 10.0]
                     {
                     If ( x:PenetrationDepth > p )
                       Then SetValue( x:PenetrationCondition,
                                      Enough )
                       Else SetValue( x:PenetrationCondition,
                                      Short );
                     };
                 PostMessage( x # ":: The EdgeLayer is " # x:EdgeLayer
                                # " and " # x:EdgeSoil );
                 };
                 };
          };
    } );

       /************************************
        ****  FUNCTION: CheckLayers
        ************************************/
MakeFunction( CheckLayers, [],
    {
    PostMessage( "*** The Check Of Panel-Edge Condition *** " );
```

```
    CheckEdgeLayer(  );
    PostMessage( "*** The Check Of Layer Condition Behind Panel ***" );
    CheckLayerCondition(  );
    } );

       /***********************************
        ****  FUNCTION: CheckLandSoftness
        ***********************************/
MakeFunction( CheckLandSoftness, [],
    {
    {
    ForAll [ x|MeasurePoint ]
           {
           SetValue( x:Others, None );
           {
           If  ( x:MeasurementType #= NotDecided )
             Then  ( Let [p x:PanelName]
                         {
                         If  ( p:NValueCurve #= Low )
                           Then {
                                If  ( p:LayerCondition #= Bad )
                                  Then {
                                       SetValue( x, FailurePattern,
                                                 Settlement );
                                       SetValue( x, MeasurementType,
                                                 Settlement:MeasurementType );
                                       PostMessage( " At this " #
                                                    x # "  " #
                                                    x:FailurePattern
                                                    # " is predicted."
                                                    # " So we need the Measurement "
                                                    # x:MeasurementType );
                                       {
                                       If  ( p:EdgeCondition #=
                                             Bad )
                                         Then {
                                              SetValue( x, Others,
                                                        Heaving );
                                              PostMessage( " At this "
                                                           #
                                                           x
                                                           #
                                                           "  "
                                                           #
                                                           x:Others
                                                           #
                                                           " is also predicted."
                                                           #
                                                           " So we need to pay attention
to " );
                                              };
                                       };
                                       }
                                Else {
                                       If  ( p:EdgeCondition #=
                                             Bad )
                                         Then {
                                              SetValue( x, FailurePattern,
```

```
                                        Heaving );
                                    SetValue( x, MeasurementType,
                                        Heaving:MeasurementType );
                                    PostMessage( " At this "
                                                 #
                                                 x
                                                 #
                                                 "  "
                                                 #
                                                 x:FailurePattern
                                                 #
                                                 " is predicted."
                                                 #
                                                 " So we need the Measurement "
                                                 #
                                                 x:MeasurementType );
                                    };
                                };
                            }
                        Else {
                            SetValue( x, MeasurementType, Type1 );
                            SetValue( x, FailurePattern, None );
                            PostMessage( " At this " # x # "  "
                                         # x:FailurePattern
                                         # " is predicted."
                                         # " So we need the Measurement "
                                         # x:MeasurementType );
                            };
                      } )
          Else Let [p x:PanelName]
                   {
                   If ( ( p:NValueCurve #= Low ) And ( p:EdgeCondition
                                                       #=
                                                       Bad ) )
                     Then {
                          SetValue( x:Others, Heaving );
                          PostMessage( " At  " # x # "  " # x:Others
                                       # " is also predicted."
                                       # " So we need to pay attention to " );
                          };
                   };
        };
        };
    };
    {
    ForAll [ y|StateBox ]
           UpdateImage( y );
    };
    } );

       /************************************
        ****  FUNCTION: DecideNumEquips
        ************************************/
MakeFunction( DecideNumEquips, [],
    {
    ForAll [ x|MeasurePoint ]
           {
           Let [y x:MeasurementType]
```

```
                {
                Let [PID x:PanelName]
                    {
                    SetValue( x, NumInclinometer, 1 + x:AddNumInclino );
                    SetValue( x, NumInclinoWall, 1 );
                    SetValue( x, NumInclinoGround, x:AddNumInclino );
                    SetValue( x, NumLoadCell, Support:NumMember );
                    If ( x:MeasurementType #= Type3 )
                      Then {
                           SetValue( x, NumEarthPressureGauge, Floor( ( ( PID:EffDepth
                                                                        /

EPressGauge:Pitch )
                                                                        *
                                                                        2 )
                                                                      -
                                                                      1 ) );
                           SetValue( x, NumWaterPressureGauge, x:NumEarthPressureGauge );
                           SetValue( x, NumStressGauge, Floor( ( PID:EffDepth
                                                               /
                                                               StressGauge:Pitch )
                                                             *
                                                             2 ) );
                           }
                      Else {
                           SetValue( x, NumEarthPressureGauge, 0 );
                           SetValue( x, NumWaterPressureGauge, 0 );
                           SetValue( x, NumStressGauge, 0 );
                           };
                    If ( x:MeasurementType #= Type4 )
                      Then {
                           If ( x:Others #= Heaving )
                             Then SetValue( x, NumSettlementGauge,
                                            2 )
                             Else SetValue( x, NumSettlementGauge,
                                            1 );
                           }
                      Else {
                           If ( x:Others #= Heaving )
                             Then SetValue( x, NumSettlementGauge,
                                            1 )
                             Else SetValue( x, NumSettlementGauge,
                                            0 );
                           };
                    };
                };
            };
    } );

       /************************************
        ****  FUNCTION: MemberCounter
        ************************************/
MakeFunction( MemberCounter, [Object],
    {
    SetValue( Object, NumMember, 0 );
    ForAll [ x|Object ]
           {
           Let [y Object:NumMember + 1]
```

```
            SetValue( Object, NumMember, y );
        };
    } );

    /************************************
     ****  FUNCTION: CountNumEquips
     ************************************/
MakeFunction( CountNumEquips, [],
    {
    SetValue( Inclinometer, NumMember, 0 );
    SetValue( LoadCell, NumMember, 0 );
    SetValue( LoadCell, NumMember, 0 );
    SetValue( EPressGauge, NumMember, 0 );
    SetValue( WPressGauge, NumMember, 0 );
    SetValue( StressGauge, NumMember, 0 );
    SetValue( SettleGauge, NumMember, 0 );
    ForAll [ x|MeasurePoint ]
        {
        SetValue( Inclinometer, NumMember, Inclinometer:NumMember
                                            + x:NumInclinometer );
        SetValue( LoadCell, NumMember, LoadCell:NumMember +
                                        x:NumLoadCell );
        SetValue( EPressGauge, NumMember, EPressGauge:NumMember
                                            + x:NumEarthPressureGauge );
        SetValue( WPressGauge, NumMember, WPressGauge:NumMember
                                            + x:NumWaterPressureGauge );
        SetValue( StressGauge, NumMember, StressGauge:NumMember
                                            + x:NumStressGauge );
        SetValue( SettleGauge, NumMember, SettleGauge:NumMember
                                            + x:NumSettlementGauge );
        };
    } );

    /************************************
     ****  FUNCTION: MakeEquips
     ************************************/
MakeFunction( MakeEquips, [],
    {
    SubMakeEquips( Inclinometer );
    SubMakeEquips( LoadCell );
    SubMakeEquips( EPressGauge );
    SubMakeEquips( WPressGauge );
    SubMakeEquips( StressGauge );
    SubMakeEquips( SettleGauge );
    } );

    /************************************
     ****  FUNCTION: SubMakeEquips
     ************************************/
MakeFunction( SubMakeEquips, [Object],
    {
    ForAll [ x|Object ]
        DeleteInstance( x );
    For numb [1 Object:NumMember ]
        MakeInstance( Object # numb, Object );
    } );

    /************************************
```

```
        ****  FUNCTION: Test2
        ************************************/
MakeFunction( Test2, [],
    {
    SetValue( MeasureEquip, Initial, 0 );
    ForAll [ x|MeasurePoint ]
           {
           Let [Start MeasureEquip:Initial + 1]
               {
               Let [End MeasureEquip:Initial + x:NumInclinometer]
                   {
                   For numb [Start End ]
                       {
                       SetValue( Inclinometer # numb, ID, x );
                       SetValue( Inclinometer # numb, CoordinateX1,
                                 x:CoordinateX1 );
                       SetValue( Inclinometer # numb, CoordinateY1,
                                 x:CoordinateY1 );
                       };
                   SetValue( MeasureEquip, Initial, End );
                   };
               };
           };
    } );

       /************************************
        ****  FUNCTION: AddInclinoPoint
        ************************************/
MakeFunction( AddInclinoPoint, [],
    {
    ForAll [ x|MeasurePoint ]
           {
           {
           If ( x:AddNumInclino >= 1 )
             Then {
                  Let [ratio Sqrt( 25 / ( ( x:ExDistanceX ^ 2 )
                                        + ( x:ExDistanceY
                                              ^ 2 ) ) )]
                      {
                      SetValue( x, AddInclino1X, x:CoordinateX1
                                                   - ( ratio *
                                                         x:ExDistanceX ) );
                      SetValue( x, AddInclino1Y, x:CoordinateY1
                                                   - ( ratio *
                                                         x:ExDistanceY ) );
                      };
                  }
             Else {
                  SetValue( x, AddInclino1X, None );
                  SetValue( x, AddInclino1Y, None );
                  };
           };
           {
           If ( x:AddNumInclino >= 2 )
             Then {
                  Let [ratio Sqrt( 100 / ( ( x:ExDistanceX ^ 2 )
                                         + ( x:ExDistanceY
                                               ^ 2 ) ) )]
```

```
                    {
                    SetValue( x, AddInclino2X, x:CoordinateX1
                                                - ( ratio *
                                                    x:ExDistanceX ) );
                    SetValue( x, AddInclino2Y, x:CoordinateY1
                                                - ( ratio *
                                                    x:ExDistanceY ) );
                    };
                }
            Else {
                SetValue( x, AddInclino2X, None );
                SetValue( x, AddInclino2Y, None );
                };
        };
        };
    } );

      /************************************
       ****  FUNCTION: PutInclinoID
       ************************************/
MakeFunction( PutInclinoID, [],
    {
    SetValue( MeasureEquip, Initial, 0 );
    ForAll [ x|MeasurePoint ]
          {
          Let [Start MeasureEquip:Initial + 1]
              {
              Let [End MeasureEquip:Initial + x:NumInclinometer]
                  {
                  For numb [Start End ]
                      {
                      SetValue( Inclinometer # numb, ID, x );
                      SetValue( Inclinometer # numb, CoordinateZ1,
                                x:CoordinateZ1 );
                      SetValue( Inclinometer # numb, CoordinateZ2,
                                x:InclinoDepth );
                      {
                      If ( numb #= Start )
                        Then {
                             SetValue( Inclinometer # numb, CoordinateX1,
                                       x:CoordinateX1 );
                             SetValue( Inclinometer # numb, CoordinateY1,
                                       x:CoordinateY1 );
                             };
                      };
                      {
                      If ( numb #= ( Start + 1 ) )
                        Then {
                             SetValue( Inclinometer # numb, CoordinateX1,
                                       x:AddInclino1X );
                             SetValue( Inclinometer # numb, CoordinateY1,
                                       x:AddInclino1Y );
                             };
                      };
                      {
                      If ( numb #= ( Start + 2 ) )
                        Then {
                             SetValue( Inclinometer # numb, CoordinateX1,
```

```
                                    x:AddInclino2X );
                            SetValue( Inclinometer # numb, CoordinateY1,
                                    x:AddInclino2Y );
                            };
                        };
                        };
                    SetValue( MeasureEquip, Initial, End );
                    };
                };
            };
    } );

       /************************************
        ****  FUNCTION: PutLoadCellID
        ************************************/
MakeFunction( PutLoadCellID, [],
    {
    SetValue( MeasureEquip, Initial, 0 );
    ForAll [ x|MeasurePoint ]
           {
           Let [Start MeasureEquip:Initial + 1]
               {
               Let [End MeasureEquip:Initial + x:NumLoadCell]
                   {
                   For numb [Start End ]
                       {
                       SetValue( LoadCell # numb, ID, x );
                       SetValue( LoadCell # numb, CoordinateX1,
                                 x:CoordinateX1 );
                       SetValue( LoadCell # numb, CoordinateY1,
                                 x:CoordinateY1 );
                       {
                       Let [nth numb - Start + 1]
                           {
                           Let [z GetNthElem( x:NSupportZ, nth )]
                               SetValue( LoadCell # numb, CoordinateZ1,
                                         z );
                           SetValue( LoadCell # numb, SupportNo,
                                     nth );
                           };
                       };
                       };
                   SetValue( MeasureEquip, Initial, End );
                   };
               };
           };
    } );

       /************************************
        ****  FUNCTION: EquipLists
        ************************************/
MakeFunction( EquipLists, [],
    {
    ForAll [ x|MeasurePoint ]
           {
           ClearList( x:NSupportZ );
           ClearList( x:NStressGaugeZ );
           ClearList( x:NEPressGaugeZ );
```

```
ClearList( x:NWPressGaugeZ );
{
ForAll [ y|Support ]
       AppendToList( x:NSupportZ, y:CoordinateZ1 );
};
{
Let [n Floor( x:NumStressGauge / 2 )]
    {
    If ( n > 0 )
      Then {
           For nth [1 n ]
               {
               AppendToList( x:NStressGaugeZ, ( x:CoordinateZ1
                                                  -
                                                  1 )
                                                - ( nth
                                                    *
                                                    StressGauge:Pitch ) );
               };
           For nth [1 n ]
               {
               AppendToList( x:NStressGaugeZ, ( x:CoordinateZ1
                                                  -
                                                  1 )
                                                - ( nth
                                                    *
                                                    StressGauge:Pitch ) );
               };
           };
    };
};
{
Let [n Floor( ( x:NumEarthPressureGauge + 1 ) / 2 )]
    {
    If ( n > 0 )
      Then {
           For nth [1 n ]
               {
               AppendToList( x:NEPressGaugeZ, ( x:CoordinateZ1
                                                  -
                                                  1 )
                                                - ( nth
                                                    *
                                                    EPressGauge:Pitch ) );
               AppendToList( x:NWPressGaugeZ, ( x:CoordinateZ1
                                                  -
                                                  1 )
                                                - ( nth
                                                    *
                                                    WPressGauge:Pitch ) );
               };
           For nth [2 n ]
               {
               AppendToList( x:NEPressGaugeZ, ( x:CoordinateZ1
                                                  -
                                                  1 )
                                                - ( nth
                                                    *
```

```
                                                        EPressGauge:Pitch ) );
                        AppendToList( x:NWPressGaugeZ, ( x:CoordinateZ1
                                                           -
                                                           1 )
                                                         - ( nth
                                                             *
                                                             WPressGauge:Pitch ) );
                        };
                    };
                };
            };
            };
    } );

       /************************************
        ****  FUNCTION: PutStressGaugeID
        ************************************/
MakeFunction( PutStressGaugeID, [],
    {
    If ( StressGauge:NumMember > 0 )
      Then {
           SetValue( MeasureEquip, Initial, 0 );
           ForAll [ x|MeasurePoint ]
                  {
                  Let [Start MeasureEquip:Initial + 1]
                      {
                      Let [End MeasureEquip:Initial + x:NumStressGauge]
                          {
                          For numb [Start End ]
                              {
                              SetValue( StressGauge # numb, ID,
                                        x );
                              SetValue( StressGauge # numb, CoordinateX1,
                                        x:CoordinateX1 );
                              SetValue( StressGauge # numb, CoordinateY1,
                                        x:CoordinateY1 );
                              {
                              Let [nth numb - Start + 1]
                                  {
                                  Let [z GetNthElem( x:NStressGaugeZ,
                                                     nth )]
                                    SetValue( StressGauge #
                                                numb, CoordinateZ1,
                                              z );
                                  };
                              };
                              };
                          SetValue( MeasureEquip, Initial, End );
                          };
                      };
                  };
           };
    } );

       /************************************
        ****  FUNCTION: PutPressGaugeID
        ************************************/
MakeFunction( PutPressGaugeID, [],
```

```
    {
    If ( EPressGauge:NumMember > 0 )
      Then {
           SetValue( MeasureEquip, Initial, 0 );
           ForAll [ x|MeasurePoint ]
                  {
                  Let [Start MeasureEquip:Initial + 1]
                      {
                      Let [End MeasureEquip:Initial + x:NumEarthPressureGauge]
                          {
                          For numb [Start End ]
                              {
                              SetValue( EPressGauge # numb, ID,
                                        x );
                              SetValue( EPressGauge # numb, CoordinateX1,
                                        x:CoordinateX1 );
                              SetValue( EPressGauge # numb, CoordinateY1,
                                        x:CoordinateY1 );
                              SetValue( WPressGauge # numb, ID,
                                        x );
                              SetValue( WPressGauge # numb, CoordinateX1,
                                        x:CoordinateX1 );
                              SetValue( WPressGauge # numb, CoordinateY1,
                                        x:CoordinateY1 );
                              {
                              Let [nth numb - Start + 1]
                                  {
                                  Let [z GetNthElem( x:NEPressGaugeZ,
                                                     nth )]
                                      SetValue( EPressGauge #
                                                numb, CoordinateZ1,
                                                z );
                                  Let [z GetNthElem( x:NWPressGaugeZ,
                                                     nth )]
                                      SetValue( WPressGauge #
                                                numb, CoordinateZ1,
                                                z );
                                  };
                              };
                              };
                          SetValue( MeasureEquip, Initial, End );
                          };
                      };
                  };
           };
    } );

       /************************************
        ****  FUNCTION: CheckInclinoDepth
        ************************************/
MakeFunction( CheckInclinoDepth, [],
    {
    ForAll [ x|MeasurePoint ]
           {
           Let [pid x:PanelName]
               {
               If ( x:SurfacePosition #= Under )
                 Then {
```

```
                    If ( ( pid:CoordinateZ2 - x:SlipSurfaceDepth )
                          < 10 )
                      Then SetValue( x:InclinoDepth, x:SlipSurfaceDepth
                                                        - 2 )
                      Else PostMessage( "We need to check the slip position in detail. "
);
                    }
                Else {
                    If ( pid:EdgeSoil #= Clay )
                      Then SetValue( x:InclinoDepth, pid:CoordinateZ2
                                                        - 5 )
                      Else SetValue( x:InclinoDepth, pid:CoordinateZ2 );
                    };
            };
        };
} );

    /************************************
     ****  FUNCTION: PutSettleGaugeID
     ************************************/
MakeFunction( PutSettleGaugeID, [],
    {
    If ( SettleGauge:NumMember > 0 )
      Then {
          SetValue( MeasureEquip, Initial, 0 );
          ForAll [ x|MeasurePoint ]
                {
                Let [Start MeasureEquip:Initial + 1]
                    {
                    Let [End MeasureEquip:Initial + x:NumSettlementGauge]
                        {
                        Let [area ExcavationArea:MaxDepthArea]
                            {
                            For numb [Start End ]
                                {
                                SetValue( SettleGauge # numb,
                                          ID, x );
                                SetValue( SettleGauge # numb,
                                          CoordinateZ1, area:CoordinateZ1 );
                                {
                                If ( numb #= Start )
                                  Then {
                                      SetValue( SettleGauge
                                                  # numb, CoordinateX1,
                                                area:GravityPointX );
                                      SetValue( SettleGauge
                                                  # numb, CoordinateY1,
                                                area:GravityPointY );
                                      };
                                };
                                {
                                If ( numb #= ( Start + 1 ) )
                                  Then {
                                      SetValue( SettleGauge
                                                  # numb, CoordinateX1,
                                                x:AddSettleGaugeX );
                                      SetValue( SettleGauge
                                                  # numb, CoordinateY1,
```

```
                                                    x:AddSettleGaugeY );
                                            };
                                    };
                                    };
                                SetValue( MeasureEquip, Initial,
                                        End );
                                };
                            };
                        };
                    };
    } );

       /************************************
        ****  FUNCTION: AddSettleGaugePoint
        ************************************/
MakeFunction( AddSettleGaugePoint, [],
    {
    ForAll [ x|MeasurePoint ]
           {
           {
           If ( x:Others = 1 )
             Then {
                  Let [ratio Sqrt( ( ( x:ExcavationDepth / 2 )
                                   ^ 2 ) / ( ( x:ExDistanceX
                                                ^ 2 ) + ( x:ExDistanceY
                                                          ^
                                                          2 ) ) )]
                      {
                      SetValue( x, AddSettleGaugeX, x:CoordinateX1
                                                    - ( ratio
                                                        *
                                                        x:ExDistanceX ) );
                      SetValue( x, AddSettleGaugeY, x:CoordinateY1
                                                    - ( ratio
                                                        *
                                                        x:ExDistanceY ) );
                      };
                  }
             Else {
                  SetValue( x, AddSettleGaugeX, None );
                  SetValue( x, AddSettleGaugeY, None );
                  };
           };
           };
    } );

       /************************************
        ****  FUNCTION: LayoutMeasureEquips
        ************************************/
MakeFunction( LayoutMeasureEquips, [],
    {
    PostMessage( "***Deciding The Necessary Number Of Each Equipment***" );
    DecideNumEquips(  );
    CountNumEquips(  );
    MakeEquips(  );
    OutputEquipData1(  );
    PostMessage( "***Caliculating The Setting Point of Each Equipment***" );
```

```
    AddInclinoPoint(  );
    AddSettleGaugePoint(  );
    EquipLists(  );
    CheckInclinoDepth(  );
    PostMessage( "***Layouting Each Equips***" );
    PutInclinoID(  );
    PutLoadCellID(  );
    PutPressGaugeID(  );
    PutStressGaugeID(  );
    PutSettleGaugeID(  );
    OutputEquipData2(  );
    PostMessage( "*** All Set ***" );
    } );

       /************************************
        ****  FUNCTION: DelIns
        ************************************/
MakeFunction( DelIns, [Object],
    {
    ForAll [ x|Object ]
           DeleteInstance( x );
    } );

       /************************************
        ****  FUNCTION: SetInitialValuePanel
        ************************************/
MakeFunction( SetInitialValuePanel, [],
    {
    SetValue( Panel1:CoordinateX1, 0.0 );
    SetValue( Panel1:CoordinateX2, 0.0 );
    SetValue( Panel1:CoordinateY1, 150.0 );
    SetValue( Panel1:CoordinateY2, 0.0 );
    SetValue( Panel1:CoordinateZ1, 10.0 );
    SetValue( Panel1:CoordinateZ2, -13.0 );
    SetValue( Panel1:Thickness, 0.60 );
    SetValue( Panel1:ExistFacility, Null );
    SetValue( Panel2:CoordinateX1, 0.0 );
    SetValue( Panel2:CoordinateX2, 100.0 );
    SetValue( Panel2:CoordinateY1, 0.0 );
    SetValue( Panel2:CoordinateY2, 0.0 );
    SetValue( Panel2:CoordinateZ1, 10.0 );
    SetValue( Panel2:CoordinateZ2, -13.0 );
    SetValue( Panel2:Thickness, 0.60 );
    SetValue( Panel2:ExistFacility, Null );
    SetValue( Panel3:CoordinateX1, 100.0 );
    SetValue( Panel3:CoordinateX2, 100.0 );
    SetValue( Panel3:CoordinateY1, 0.0 );
    SetValue( Panel3:CoordinateY2, 100.0 );
    SetValue( Panel3:CoordinateZ1, 10.0 );
    SetValue( Panel3:CoordinateZ2, -13.0 );
    SetValue( Panel3:Thickness, 0.60 );
    SetValue( Panel3:ExistFacility, Null );
    SetValue( Panel4:CoordinateX1, 100.0 );
    SetValue( Panel4:CoordinateX2, 50.0 );
    SetValue( Panel4:CoordinateY1, 100.0 );
    SetValue( Panel4:CoordinateY2, 100.0 );
    SetValue( Panel4:CoordinateZ1, 10.0 );
    SetValue( Panel4:CoordinateZ2, -13.0 );
```

```
    SetValue( Panel4:Thickness, 0.60 );
    SetValue( Panel4:ExistFacility, Null );
    SetValue( Panel5:CoordinateX1, 50.0 );
    SetValue( Panel5:CoordinateX2, 50.0 );
    SetValue( Panel5:CoordinateY1, 100.0 );
    SetValue( Panel5:CoordinateY2, 150.0 );
    SetValue( Panel5:CoordinateZ1, 10.0 );
    SetValue( Panel5:CoordinateZ2, -13.0 );
    SetValue( Panel5:Thickness, 0.60 );
    SetValue( Panel5:ExistFacility, Null );
    SetValue( Panel6:CoordinateX1, 50.0 );
    SetValue( Panel6:CoordinateX2, 0.0 );
    SetValue( Panel6:CoordinateY1, 150.0 );
    SetValue( Panel6:CoordinateY2, 150.0 );
    SetValue( Panel6:CoordinateZ1, 10.0 );
    SetValue( Panel6:CoordinateZ2, -13.0 );
    SetValue( Panel6:Thickness, 0.60 );
    SetValue( Panel6:ExistFacility, Null );
    } );

       /************************************
        ****  FUNCTION: SetInitialValueExcavationArea
        ************************************/
MakeFunction( SetInitialValueExcavationArea, [],
    {
    SetValue( Area1:CoordinateX1, 0.0 );
    SetValue( Area1:CoordinateX2, 50.0 );
    SetValue( Area1:CoordinateY1, 100.0 );
    SetValue( Area1:CoordinateY2, 150.0 );
    SetValue( Area1:CoordinateZ1, 10.0 );
    SetValue( Area1:CoordinateZ2, -10.0 );
    SetValue( Area1:ExcavationDepth, 10.0 );
    SetValue( Area2:CoordinateX1, 0.0 );
    SetValue( Area2:CoordinateX2, 100.0 );
    SetValue( Area2:CoordinateY1, 0.0 );
    SetValue( Area2:CoordinateY2, 100.0 );
    SetValue( Area2:CoordinateZ1, 10.0 );
    SetValue( Area2:CoordinateZ2, -10.0 );
    SetValue( Area2:ExcavationDepth, 20.0 );
    } );

       /************************************
        ****  FUNCTION: SetInitialValueFacility
        ************************************/
MakeFunction( SetInitialValueFacility, [],
    {
    SetValue( Facility1:CoordinateX1, 120.0 );
    SetValue( Facility1:CoordinateX2, 120.0 );
    SetValue( Facility1:CoordinateY1, 10.0 );
    SetValue( Facility1:CoordinateY2, 20.0 );
    SetValue( Facility1:CoordinateZ1, 0.0 );
    SetValue( Facility1:CoordinateZ2, 0.0 );
    SetValue( Facility1:FacilityName, Hotel );
    SetValue( Facility1:ImportanceNumber, 10.0 );
    SetValue( Facility2:CoordinateX1, -20.0 );
    SetValue( Facility2:CoordinateX2, -40.0 );
    SetValue( Facility2:CoordinateY1, 0.0 );
    SetValue( Facility2:CoordinateY2, 150.0 );
```

```
    SetValue( Facility2:CoordinateZ1, 0.0 );
    SetValue( Facility2:CoordinateZ2, 0.0 );
    SetValue( Facility2:FacilityName, UtilityLine );
    SetValue( Facility2:ImportanceNumber, 8.0 );
    } );

       /***********************************
        ****  FUNCTION: OutputEquipData1
        ***********************************/
MakeFunction( OutputEquipData1, [],
    {
    PostMessage( "*** Initial Design for Measurement Plan(1) ***" );
    ForAll [ x|MeasurePoint ]
           {
           Let [pid x:PanelName]
               {
               PostInputForm( " " # x, x, PriorityNumber, "Importance:",
                              x, PanelName, "Panel:", x, CoordinateX1,
                              "Coordinate-X:", x, CoordinateY1, "Coordinate-Y:",
                              pid, LayerCondition, "LayerCondition:",
                              pid, NValueCurve, "NvalueCurve:", pid,
                              EdgeCondition, "PanelEdgeCondition:",
                              x, SurfacePosition, "Slipsurface:" );
               PostInputForm( NULL # x, x, FailurePattern, "FailurePattern:",
                              x, MeasurementType, MeasurementType,
                              x, NumInclinometer, "Inclinometer:",
                              x, NumLoadCell, "LoadCell:", x, NumStressGauge,
                              "StressGauge:", x, NumEarthPressureGauge,
                              "EarthpressureGauge:", x, NumWaterPressureGauge,
                              "WaterPressureGauge:", x, NumSettlementGauge,
                              "SettlementGauge:" );
               };
           };
    } );

       /***********************************
        ****  FUNCTION: MeasurementPlan
        ***********************************/
MakeFunction( MeasurementPlan, [],
    {
    ZeroInitialize(  );
    SetInitialValue(  );
    SetMeasurementPoint(  );
    CheckLayers(  );
    PredictFailurePattern(  );
    LayoutMeasureEquips(  );
    } );

       /***********************************
        ****  FUNCTION: OutputEquipData2
        ***********************************/
MakeFunction( OutputEquipData2, [],
    {
    If ( PostMenu( "Do you wish to see the result of reasoning ?",
                   Yes, No ) #= Yes )
      Then {
           PostMessage( "*** Initial Design for Measurement Plan(2) ***" );
           {
```

```
ForAll [ x|Inclinometer ]
        {
        Let [m x:ID]
            {
            PostInputForm( " " # x, x, ID, "MeasurePointNo.:",
                           m, PanelName, "Panel:", x, CoordinateX1,
                           "Coordinate-X:", x, CoordinateY1,
                           "Coordinate-Y:", x, CoordinateZ1,
                           "TopElevation(m):", x, CoordinateZ2,
                           "BottomElevation(m):", m, FailurePattern,
                           "FailurePattern:", m, MeasurementType,
                           MeasurementType );
            };
        };
};
{
ForAll [ x|LoadCell ]
        {
        Let [m x:ID]
            {
            PostInputForm( " " # x, x, ID, "MeasurePointNo.:",
                           m, PanelName, "Panel:", x, CoordinateX1,
                           "Coordinate-X:", x, CoordinateY1,
                           "Coordinate-Y:", x, CoordinateZ1,
                           "SetElevation(m):", m, FailurePattern,
                           "FailurePattern:", m, MeasurementType,
                           MeasurementType );
            };
        };
};
{
ForAll [ x|StressGauge ]
        {
        Let [m x:ID]
            {
            PostInputForm( " " # x, x, ID, "MeasurePointNo.:",
                           m, PanelName, "Panel:", x, CoordinateX1,
                           "Coordinate-X:", x, CoordinateY1,
                           "Coordinate-Y:", x, CoordinateZ1,
                           "SetElevation(m):", m, FailurePattern,
                           "FailurePattern:", m, MeasurementType,
                           MeasurementType );
            };
        };
};
{
ForAll [ x|EPressGauge ]
        {
        Let [m x:ID]
            {
            PostInputForm( " " # x, x, ID, "MeasurePointNo.:",
                           m, PanelName, "Panel:", x, CoordinateX1,
                           "Coordinate-X:", x, CoordinateY1,
                           "Coordinate-Y:", x, CoordinateZ1,
                           "SetElevation(m):", m, FailurePattern,
                           "FailurePattern:", m, MeasurementType,
                           MeasurementType );
            };
```

```
            };
        };
        {
        ForAll [ x|WPressGauge ]
            {
            Let [m x:ID]
                {
                PostInputForm( " " # x, x, ID, "MeasurePointNo.:",
                            m, PanelName, "Panel:", x, CoordinateX1,
                            "Coordinate-X:", x, CoordinateY1,
                            "Coordinate-Y:", x, CoordinateZ1,
                            "SetElevation(m):", m, FailurePattern,
                            "FailurePattern:", m, MeasurementType,
                            MeasurementType );
                };
            };
        };
        {
        ForAll [ x|SettleGauge ]
            {
            Let [m x:ID]
                {
                PostInputForm( " " # x, x, ID, "MeasurePointNo.:",
                            m, PanelName, "Panel:", x, CoordinateX1,
                            "Coordinate-X:", x, CoordinateY1,
                            "Coordinate-Y:", x, CoordinateZ1,
                            "SetElevation(m):", m, FailurePattern,
                            "FailurePattern:", m, MeasurementType,
                            MeasurementType );
                };
            };
        };
        };
} );

    /************************************
     ****  FUNCTION: SetInitialValueMeasurePoint
     ************************************/
MakeFunction( SetInitialValueMeasurePoint, [],
    {
    ForAll [ x|MeasurePoint ]
            {
            SetValue( x:FailurePattern, NotDecided );
            SetValue( x:MeasurementType, NotDecided );
            };
    {
    ForAll [ y|StateBox ]
           UpdateImage( y );
    };
    } );



/*********************************************************/
/**          ALL CLASSES ARE SAVED BELOW                **/
/*********************************************************/

    /************************************
```

```
        ****  CLASS: Image
        ************************************/

/************** METHOD: MClear   **************/
MakeMethod( Image, MClear, [],
    {} );

/************** METHOD: MGravityPointX   **************/
MakeMethod( Image, MGravityPointX, [SlotName ],
    {
    Self:GravityPointX = ( ( Self:CoordinateX1 + Self:CoordinateX2 )
                           / 2 );
    GravityPointX;
    } );

/************** METHOD: MGravityPonitX   **************/
MakeMethod( Image, MGravityPonitX, [],
    SetValue( Self:GravityPointX, ( Self:CoordinateX1 + Self:CoordinateX2 )
                                  / 2 ) );

/************** METHOD: MGravityPoint   **************/
MakeMethod( Image, MGravityPoint, [SlotName ],
    {
    Self:GravityPointX = ( ( Self:CoordinateX1 + Self:CoordinateX2 )
                           / 2 );
    Self:GravityPointY = ( ( Self:CoordinateY1 + Self:CoordinateY2 )
                           / 2 );
    AppendToList( Self:GravityPoint, Self:GravityPointX );
    AppendToList( Self:GravityPoint, Self:GravityPointY );
    Self:GravityPoint;
    } );
Image:Width = 150;
Image:Height = 120;
Image:WinType = Make_ImageWindow;
Image:Visible = FALSE;
Image:ShowBorder = TRUE;
Image:Transparent = FALSE;

       /************************************
        ****  CLASS: SlotView
        ************************************/

       /************************************
        ****  CLASS: OutputView
        ************************************/

       /************************************
        ****  CLASS: Meter
        ************************************/
Meter:Draw = Draw_HMeter;
Meter:Update = Update_HMeter;

       /************************************
        ****  CLASS: StateBox
        ************************************/
SetSlotOption( StateBox:AllowableValues, MULTIPLE );
StateBox:Draw = Draw_RectLight;
StateBox:Update = Update_RectLight;
```

```
        /************************************
         ****  CLASS: InputOutputView
         ************************************/

        /************************************
         ****  CLASS: Slider
         ************************************/
Slider:Width = 200;
Slider:Height = 75;
Slider:WinType = Make_SliderWindow;
Slider:Draw = Draw_Slider;
Slider:Update = Update_Slider;

        /************************************
         ****  CLASS: Edit
         ************************************/
Edit:ShowBorder = FALSE;
Edit:Width = 100;
Edit:Height = 25;
Edit:WinType = Make_EditWindow;
Edit:Update = Update_Edit;

        /************************************
         ****  CLASS: Transcript
         ************************************/
Transcript:ProportionalFont = TRUE;
Transcript:WinType = Make_TransWindow;

        /************************************
         ****  CLASS: Button
         ************************************/
Button:ShowBorder = FALSE;
Button:Width = 100;
Button:Height = 25;
Button:WinType = Make_ButtonWindow;
Button:Draw = Draw_Bitmap;

        /************************************
         ****  CLASS: Bitmap
         ************************************/
Bitmap:FitToScreen = FALSE;
Bitmap:Draw = Draw_Bitmap;

        /************************************
         ****  CLASS: Drawing
         ************************************/
Drawing:XLeft = 0;
Drawing:YTop = 0;
Drawing:XRight = 100;
Drawing:YBottom = 100;
Drawing:Draw = Draw_Drawing;
Drawing:Print = Print_Drawing;

        /************************************
         ****  CLASS: Text
         ************************************/
Text:Justification = CENTER;
```

```
Text:ShowBorder = FALSE;
Text:Width = 100;
Text:Height = 25;
Text:WinType = Make_StaticWindow;

        /************************************
         ****  CLASS: LinePlot
         ************************************/

/************** METHOD: MGravityPointX   **************/
MakeMethod( LinePlot, MGravityPointX, [SlotName ],
    {
    Self:GravityPointX = ( ( Self:CoordinateX1 + Self:CoordinateX2 )
                           / 2 );
    GravityPointX;
    } );
LinePlot:AutoScale = FALSE;
LinePlot:Grid = FALSE;
LinePlot:Draw = Draw_LinePlot;
LinePlot:Update = Update_LinePlot;
LinePlot:Print = Print_LinePlot;

        /************************************
         ****  CLASS: member
         ************************************/
MakeClass( member, Root );

/************** METHOD: MGravityPointX   **************/
MakeMethod( member, MGravityPointX, [SlotName ],
    {
    ( Self:CoordinateX1 + Self:CoordinateX2 ) / 2;
    } );

/************** METHOD: Mdistance   **************/
MakeMethod( member, Mdistance, [],
    {
    ForAll [ x|Panel ]
           {
           Sqrt( ( ( Self:GravityPointX - x:GravityPointX ) ^
                     2 ) + ( ( Self:GravityPointY - x:GravityPointY )
                               ^ 2 ) );
           PostMessage( x # " is a Panel " # Sqrt( ( ( Self:GravityPointX
                                                       - x:GravityPointX )
                                                     ^ 2 ) + ( ( Self:GravityPointY
                                                                 -
                                                                 x:GravityPointY )
                                                               ^
                                                               2 ) ) );
           };
    } );

/************** METHOD: MGravityPointY   **************/
MakeMethod( member, MGravityPointY, [],
    {
    ( Self:CoordinateY1 + Self:CoordinateY2 ) / 2;
    } );

/************** METHOD: MMLength   **************/
```

```
MakeMethod( member, MMLength, [],
    {
    Sqrt( ( ( Self:CoordinateX1 - Self:CoordinateX2 ) ^ 2 ) +
            ( ( Self:CoordinateZ1 - Self:CoordinateZ2 ) ^ 2 ) );
    } );

/************** METHOD: MMLength   **************/
MakeMethod( member, MMLength, [],
    {
    Sqrt( ( ( Self:CoordinateX1 - Self:CoordinateX2 ) ^ 2 ) +
            ( ( Self:CoordinateY1 - Self:CoordinateY2 ) ^ 2 ) );
    } );

/************** METHOD: MMeasurePointSearch   **************/
MakeMethod( member, MMeasurePointSearch, [],
    {
    Let [CC ExcavationArea:MaxDepthArea]
        {
        If {
           ( CC:CoordinateX1 <= Self:GravityPointX ) And ( CC:CoordinateX2
                                                           >=
                                                           Self:GravityPointX )
             And ( CC:CoordinateY1 <= Self:GravityPointY ) And
             ( CC:CoordinateY2 >= Self:GravityPointY );
          }
          Then {
               SetValue( Self, ExcavationDepth, CC:MaxDepth );
               PostMessage( " MeasurePoint is Center." );
               }
          Else {
               PostMessage( " MeasurePoint is  None." );
               };
        };
    } );

/************** METHOD: MMDepth   **************/
MakeMethod( member, MMDepth, [],
    {
    Abs( Self:CoordinateZ1 - Self:CoordinateZ2 );
    } );

/************** METHOD: CalNumber   **************/
MakeMethod( member, CalNumber, [],
    {
    MemberCounter( Self );
    } );
MakeSlot( member:CoordinateZ1 );
SetSlotComment( member:CoordinateZ1, "Top Elevation (m)" );
SetSlotOption( member:CoordinateZ1, VALUE_TYPE, NUMBER );
MakeSlot( member:ExcavationNumber );
SetSlotOption( member:ExcavationNumber, VALUE_TYPE, NUMBER );
MakeSlot( member:ExcavationDepth );
SetSlotOption( member:ExcavationDepth, VALUE_TYPE, NUMBER );
MakeSlot( member:CoordinateX1 );
SetSlotComment( member:CoordinateX1, "Holizontal 2 Dimensional Plane

Point1(x1,y1)
```

```
" );
SetSlotOption( member:CoordinateX1, VALUE_TYPE, NUMBER );
MakeSlot( member:CoordinateY1 );
SetSlotComment( member:CoordinateY1, "Holizontal 2 Dimensional Plane

Point1(x1,y1)

" );
SetSlotOption( member:CoordinateY1, VALUE_TYPE, NUMBER );
MakeSlot( member:CoordinateX2 );
SetSlotComment( member:CoordinateX2, "Holizontal 2 Dimensional Plane

Point2(x2,y2)" );
SetSlotOption( member:CoordinateX2, VALUE_TYPE, NUMBER );
MakeSlot( member:CoordinateY2 );
SetSlotComment( member:CoordinateY2, "Holizontal 2 Dimensional Plane

Point2(x2,y2)" );
SetSlotOption( member:CoordinateY2, VALUE_TYPE, NUMBER );
MakeSlot( member:CoordinateZ2 );
SetSlotComment( member:CoordinateZ2, "Bottom Elevation (m)" );
SetSlotOption( member:CoordinateZ2, VALUE_TYPE, NUMBER );
MakeSlot( member:Point1 );
SetSlotOption( member:Point1, MULTIPLE );
SetSlotOption( member:Point1, VALUE_TYPE, NUMBER );
MakeSlot( member:Point2 );
SetSlotOption( member:Point2, MULTIPLE );
SetSlotOption( member:Point2, VALUE_TYPE, NUMBER );
MakeSlot( member:Point3 );
SetSlotOption( member:Point3, MULTIPLE );
SetSlotOption( member:Point3, VALUE_TYPE, NUMBER );
MakeSlot( member:Point4 );
SetSlotOption( member:Point4, MULTIPLE );
SetSlotOption( member:Point4, VALUE_TYPE, NUMBER );
MakeSlot( member:GravityPointX );
SetSlotOption( member:GravityPointX, VALUE_TYPE, NUMBER );
SetSlotOption( member:GravityPointX, IF_NEEDED, MGravityPointX );
MakeSlot( member:GravityPointY );
SetSlotOption( member:GravityPointY, VALUE_TYPE, NUMBER );
SetSlotOption( member:GravityPointY, IF_NEEDED, MGravityPointY );
MakeSlot( member:GravityPoint );
SetSlotOption( member:GravityPoint, MULTIPLE );
SetSlotOption( member:GravityPoint, VALUE_TYPE, NUMBER );
MakeSlot( member:MemberLength );
SetSlotComment( member:MemberLength, "(m)" );
SetSlotOption( member:MemberLength, VALUE_TYPE, NUMBER );
SetSlotOption( member:MemberLength, IF_NEEDED, MMLength );
MakeSlot( member:MemberDepth );
SetSlotComment( member:MemberDepth, "(m)" );
SetSlotOption( member:MemberDepth, VALUE_TYPE, NUMBER );
SetSlotOption( member:MemberDepth, IF_NEEDED, MMDepth );
MakeSlot( member:ID );
MakeSlot( member:NumMember );
SetSlotOption( member:NumMember, VALUE_TYPE, NUMBER );
SetSlotOption( member:NumMember, IF_NEEDED, CalNumber );

      /************************************
       ****  CLASS: Panel
```

```
        ************************************/
MakeClass( Panel, member );

/************** METHOD: WallCross    **************/
MakeMethod( Panel, WallCross, [],
    {
    Self:Thickness * Self:PanelLength;
    } );

/************** METHOD: WallCross    **************/
MakeMethod( Panel, WallCross, [],
    {
    Self:Thickness * Self:PanelLength;
    } );

/************** METHOD: MMeasurePointSearch    **************/
MakeMethod( Panel, MMeasurePointSearch, [],
    {
    Let [CC ExcavationArea:MaxDepthArea]
        {
        If {
           ( CC:CoordinateX1 <= Self:GravityPointX ) And ( CC:CoordinateX2
                                                           >=
                                                           Self:GravityPointX )
             And ( CC:CoordinateY1 <= Self:GravityPointY ) And
              ( CC:CoordinateY2 >= Self:GravityPointY );
           }
          Then {
               SetValue( Self, ExcavationDepth, CC:MaxDepth );
               PostMessage( " MeasurePoint is Center." );
               }
          Else {
               PostMessage( " MeasurePoint is  None." );
               };
        };
    } );

/************** METHOD: MeasurePoint    **************/
MakeMethod( Panel, MeasurePoint, [],
    ForAll [ X|Panel ]
           If MPointSearch( CoordinationX1, CoordinationY1, X )
             Then ( If MPointSearch( CoordinationX2, CoordinationY2,
                                     X )
                       Then ( SetValue( X, ExcavationDepth, ExcavationArea:MaxDepth )
                                And SetValue( X, MeasurePoint, Center ) )
                       Else 1 + 1 )
             Else 1 + 2 );

/************** METHOD: MLength    **************/
MakeMethod( Panel, MLength, [],
    {
    Self:MemberLength;
    } );

/************** METHOD: MCalPriority    **************/
MakeMethod( Panel, MCalPriority, [],
    {
    ( ( Self:ImportanceNumber * Self:PanelLength ) * ( Self:ExcavationDepth
```

```
                                                        ^ 2 ) )
      / Self:DistanceFacility;
    } );

/************** METHOD: MCalPriority   **************/
MakeMethod( Panel, MCalPriority, [],
    {
    If ( Self:ExistFacility #= Null )
      Then Floor( ( Self:PanelLength * ( Self:ExcavationDepth
                                          ^ 2 ) ) / 100.0 )
      Else {
           Let [x Self:ExistFacility]
               {
               Floor( ( ( x:ImportanceNumber * Self:PanelLength )
                          * ( Self:ExcavationDepth ^ 2 ) ) / x:DistancePanel );
               };
           };
    } );

/************** METHOD: MLength   **************/
MakeMethod( Panel, MLength, [],
    {
    Self:MemberLength;
    } );

/************** METHOD: MCheckFacility   **************/
MakeMethod( Panel, MCheckFacility, [],
    {
    CheckFacility(  );
    } );

/************** METHOD: CalEl   **************/
MakeMethod( Panel, CalEl, [],
    {
    Self:CoordinateZ1 - Self:ExcavationDepth;
    } );

/************** METHOD: CalEdgeEl   **************/
MakeMethod( Panel, CalEdgeEl, [],
    {
    Self:CoordinateZ1 - Self:MemberDepth;
    } );

/************** METHOD: CalPenetration   **************/
MakeMethod( Panel, CalPenetration, [],
    {
    Self:MemberDepth - Self:ExcavationDepth;
    } );

/************** METHOD: CalEff   **************/
MakeMethod( Panel, CalEff, [],
    {
    Self:MemberDepth - ( Self:PenetrationDepth / 2 );
    } );
MakeSlot( Panel:PanelLength );
SetSlotOption( Panel:PanelLength, VALUE_TYPE, NUMBER );
SetSlotOption( Panel:PanelLength, IF_NEEDED, MLength );
MakeSlot( Panel:UnitWeight );
```

```
SetSlotComment( Panel:UnitWeight, "(t/m)" );
MakeSlot( Panel:AllowableMoment );
MakeSlot( Panel:DesignDisplace );
MakeSlot( Panel:DeformationModulus );
MakeSlot( Panel:I );
MakeSlot( Panel:Thickness );
SetSlotComment( Panel:Thickness, "(m)" );
Panel:Thickness = 0.6;
MakeSlot( Panel:MeasurePoint );
SetSlotOption( Panel:MeasurePoint, IF_NEEDED, MMeasurePointSearch );
MakeSlot( Panel:PriorityNumber );
SetSlotOption( Panel:PriorityNumber, VALUE_TYPE, NUMBER );
SetSlotOption( Panel:PriorityNumber, IF_NEEDED, MCalPriority );
MakeSlot( Panel:ExistFacility );
SetSlotOption( Panel:ExistFacility, IF_NEEDED, MCheckFacility );
MakeSlot( Panel:Priority );
SetSlotOption( Panel:Priority, MULTIPLE );
SetValue( Panel:Priority, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4,
Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3,
Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2,
Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1,
Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4,
Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3,
Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2,
Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1,
Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4,
Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3,
Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2,
Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1,
Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4,
Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3,
Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2,
Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1,
Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4,
Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3,
Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2, Panel3, Panel4, Panel1, Panel2,
Panel3, Panel4, Panel1, Panel2, Panel3, Panel4 );
MakeSlot( Panel:PriorityOrder );
MakeSlot( Panel:FacilityDistance );
MakeSlot( Panel:NValueCurve );
SetSlotComment( Panel:NValueCurve, "Maximam Value for Each layer

If all layer behind panel is that Clay < 20, Sand < 30

So this slot value should be written " Low".

If not this is written " High"." );
MakeSlot( Panel:LayerCondition );
MakeSlot( Panel:EdgeCondition );
SetSlotComment( Panel:EdgeCondition, "Bad or Good" );
MakeSlot( Panel:ExcavationElevation );
SetSlotComment( Panel:ExcavationElevation, "T.P. m

" );
SetSlotOption( Panel:ExcavationElevation, VALUE_TYPE, NUMBER );
SetSlotOption( Panel:ExcavationElevation, IF_NEEDED, CalEl );
MakeSlot( Panel:EdgeLayer );
MakeSlot( Panel:EdgeElevation );
```

```
SetSlotComment( Panel:EdgeElevation, "T.P. (m)

" );
SetSlotOption( Panel:EdgeElevation, VALUE_TYPE, NUMBER );
SetSlotOption( Panel:EdgeElevation, IF_NEEDED, CalEdgeEl );
MakeSlot( Panel:PenetrationDepth );
SetSlotOption( Panel:PenetrationDepth, VALUE_TYPE, NUMBER );
SetSlotOption( Panel:PenetrationDepth, IF_NEEDED, CalPenetration );
MakeSlot( Panel:PenetrationCondition );
SetSlotComment( Panel:PenetrationCondition, "Enough or Short" );
MakeSlot( Panel:EdgeSoil );
MakeSlot( Panel:EdgePorePressure );
SetSlotComment( Panel:EdgePorePressure, "Low or High" );
Panel:NumMember = 0;
MakeSlot( Panel:EffDepth );
SetSlotOption( Panel:EffDepth, VALUE_TYPE, NUMBER );
SetSlotOption( Panel:EffDepth, IF_NEEDED, CalEff );

       /************************************
        ****  CLASS: Layer
        ************************************/
MakeClass( Layer, member );

/************** METHOD: MNValueCheck   **************/
MakeMethod( Layer, MNValueCheck, [],
    {
    If ( Self:SoilKind #= Clay )
      Then {
           If ( Self:NValue <= 20.0 )
             Then SetValue( Self:NValueCurve, Low )
             Else SetValue( Self:NValueCurve, High );
           }
      Else {
           If ( Self:NValue <= 30.0 )
             Then SetValue( Self:NValueCurve, Low )
             Else SetValue( Self:NValueCurve, High );
           };
    } );

/************** METHOD: MPosition   **************/
MakeMethod( Layer, MPosition, [],
    {} );
MakeSlot( Layer:LayerDepth );
SetSlotOption( Layer:LayerDepth, IF_NEEDED, MMDepth );
MakeSlot( Layer:UnitWeight );
MakeSlot( Layer:DeformationModulus );
MakeSlot( Layer:NValue );
MakeSlot( Layer:SoilStrengthC );
MakeSlot( Layer:SoilStrengthFai );
MakeSlot( Layer:WaterPressure );
MakeSlot( Layer:Poisson );
MakeSlot( Layer:SoilKind );
MakeSlot( Layer:NValueCurve );
SetSlotOption( Layer:NValueCurve, IF_NEEDED, MNValueCheck );
MakeSlot( Layer:Position );
SetSlotOption( Layer:Position, ALLOWABLE_VALUES, Behind, Edge, Bellow );
SetSlotOption( Layer:Position, IF_NEEDED, MPosition );
```

```
        /************************************
         ****  CLASS: Support
         ************************************/
MakeClass( Support, member );

/************** METHOD: MMLength   **************/
MakeMethod( Support, MMLength, [],
    {
    Sqrt( ( ( Self:CoordinateX1 - Self:CoordinateX2 ) ^ 2 ) +
          ( ( Self:CoordinateZ1 - Self:CoordinateZ2 ) ^ 2 ) );
    } );

/************** METHOD: MMLength   **************/
MakeMethod( Support, MMLength, [],
    {
    Sqrt( ( ( Self:CoordinateX1 - Self:CoordinateX2 ) ^ 2 ) +
          ( ( Self:CoordinateZ1 - Self:CoordinateZ2 ) ^ 2 ) );
    } );
MakeSlot( Support:Length );
SetSlotComment( Support:Length, "(m)" );
SetSlotOption( Support:Length, VALUE_TYPE, NUMBER );
SetSlotOption( Support:Length, IF_NEEDED, MMLength );
MakeSlot( Support:AllowableStrength );
SetSlotComment( Support:AllowableStrength, "(t/m2)" );
SetSlotOption( Support:AllowableStrength, VALUE_TYPE, NUMBER );
MakeSlot( Support:DeformationModulus );
SetSlotComment( Support:DeformationModulus, "(t/m)

" );
SetSlotOption( Support:DeformationModulus, VALUE_TYPE, NUMBER );
MakeSlot( Support:I );
SetSlotComment( Support:I, "(m4)

" );
MakeSlot( Support:Thickness );
SetSlotComment( Support:Thickness, "(m2)

" );
SetSlotOption( Support:Thickness, VALUE_TYPE, NUMBER );
MakeSlot( Support:InitialStress );
SetSlotComment( Support:InitialStress, "(t/m)" );
SetSlotOption( Support:InitialStress, VALUE_TYPE, NUMBER );
Support:NumMember = 3;

        /************************************
         ****  CLASS: MeasureEquip
         ************************************/
MakeClass( MeasureEquip, member );
MakeSlot( MeasureEquip:MeasureFrequency );
MakeSlot( MeasureEquip:MeasurementPointID );
MakeSlot( MeasureEquip:Initial );
SetSlotOption( MeasureEquip:Initial, VALUE_TYPE, NUMBER );
MeasureEquip:Initial = 14;
MakeSlot( MeasureEquip:Pitch );
SetSlotOption( MeasureEquip:Pitch, VALUE_TYPE, NUMBER );

        /************************************
         ****  CLASS: Inclinometer
```

```
        ************************************/
MakeClass( Inclinometer, MeasureEquip );
Inclinometer:NumMember = 6;
Inclinometer:Initial = 0;
MakeSlot( Inclinometer:NumInWall );
MakeSlot( Inclinometer:NumInGround );
MakeSlot( Inclinometer:SetPoint );
SetSlotComment( Inclinometer:SetPoint, "Ground or Wall

" );

       /************************************
        ****  CLASS: LoadCell
        ************************************/
MakeClass( LoadCell, MeasureEquip );
LoadCell:NumMember = 12;
MakeSlot( LoadCell:SupportNo );

       /************************************
        ****  CLASS: StressGauge
        ************************************/
MakeClass( StressGauge, MeasureEquip );
StressGauge:NumMember = 14;
StressGauge:Pitch = 3.0;

       /************************************
        ****  CLASS: EPressGauge
        ************************************/
MakeClass( EPressGauge, MeasureEquip );
EPressGauge:NumMember = 7;
EPressGauge:Pitch = 5.0;

       /************************************
        ****  CLASS: SettleGauge
        ************************************/
MakeClass( SettleGauge, MeasureEquip );
SettleGauge:NumMember = 0;

       /************************************
        ****  CLASS: WPressGauge
        ************************************/
MakeClass( WPressGauge, MeasureEquip );
WPressGauge:NumMember = 7;
WPressGauge:Pitch = 5.0;

       /************************************
        ****  CLASS: ExcavationArea
        ************************************/
MakeClass( ExcavationArea, member );

/************** METHOD: MCalPriority   **************/
MakeMethod( ExcavationArea, MCalPriority, [],
    {
    If ( Self:ExistFacility #= Null )
      Then {
           PostMessage( "This  " # Self:ID # " Dosen't Have a Facility." );
           ( Self:PanelLength * ( Self:ExcavationDepth ^ 2 ) )
             / 100.0;
```

```
        }
      Else {
        Let [x Self:ExistFacility]
            {
            ( ( x:ImportanceNumber * Self:PanelLength ) * ( Self:ExcavationDepth
                                                              ^
                                                              2 ) )
              / x:DistancePanel;
            };
        };
    } );

/************** METHOD: MaxSearch   **************/
MakeMethod( ExcavationArea, MaxSearch, [],
    {
    SetValue( Self, MaxDepth, 0.0 );
    ForAll [ x|ExcavationArea ]
        {
        If ( x:ExcavationDepth >= Self:MaxDepth )
          Then {
               SetValue( ExcavationArea, MaxDepth, x:ExcavationDepth );
               SetValue( ExcavationArea, MaxDepthArea, x );
               PostMessage( x # "  is the Deepest  " # Self:MaxDepth );
               }
          Else PostMessage( x # "  is not Deep " );
        };
    } );
ExcavationArea:CoordinateZ1 = 10.0;
ExcavationArea:CoordinateZ2 = -10.0;
MakeSlot( ExcavationArea:MaxDepthArea );
ExcavationArea:MaxDepthArea = Area2;
SetSlotOption( ExcavationArea:MaxDepthArea, IF_NEEDED, MaxSearch );
MakeSlot( ExcavationArea:MaxDepth );
SetSlotOption( ExcavationArea:MaxDepth, VALUE_TYPE, NUMBER );
ExcavationArea:MaxDepth = 20.0;
SetSlotOption( ExcavationArea:MaxDepth, IF_NEEDED, MaxSearch );

      /***********************************
       ****  CLASS: Facility
       ***********************************/
MakeClass( Facility, member );

/************** METHOD: Mdistance   **************/
MakeMethod( Facility, Mdistance, [],
    {
    ForAll [ x|Panel ]
        {
        Sqrt( ( ( Self:GravityPointX - x:GravityPointX ) ^
                  2 ) + ( ( Self:GravityPointY - x:GravityPointY )
                            ^ 2 ) );
        PostMessage( x # " is a Panel " # Sqrt( ( ( Self:GravityPointX
                                                    - x:GravityPointX )
                                                  ^ 2 ) + ( ( Self:GravityPointY
                                                              -
                                                              x:GravityPointY )
                                                            ^
                                                            2 ) ) );
        };
```

```
    } );

/************** METHOD: NearestSearch   **************/
MakeMethod( Facility, NearestSearch, [],
    {
    SetValue( Self, DistancePanel, 1000.0 );
    ForAll [ x|Panel ]
           {
           Let [distance Sqrt( ( ( Self:GravityPointX - x:GravityPointX )
                                 ^ 2 ) + ( ( Self:GravityPointY
                                             - x:GravityPointY )
                                           ^ 2 ) )]
               If ( distance <= Self:DistancePanel )
                 Then {
                      SetValue( Self, DistancePanel, distance );
                      PostMessage( x # "  is the Nearest  " # Self:DistancePanel );
                      }
                 Else PostMessage( x # "  is not Near " );
           };
    } );
MakeSlot( Facility:FacilityName );
MakeSlot( Facility:DistancePanel );
SetSlotOption( Facility:DistancePanel, VALUE_TYPE, NUMBER );
SetSlotOption( Facility:DistancePanel, IF_NEEDED, MDistance );
MakeSlot( Facility:NearestPanel );
SetSlotOption( Facility:NearestPanel, IF_NEEDED, NearestSearch );
MakeSlot( Facility:ImportanceNumber );
SetSlotOption( Facility:ImportanceNumber, VALUE_TYPE, NUMBER );
SetSlotOption( Facility:ImportanceNumber, MINIMUM_VALUE, 1.0 );
SetSlotOption( Facility:ImportanceNumber, MAXIMUM_VALUE, 10.0 );

       /************************************
        ****  CLASS: VisualSurvey
        ************************************/
MakeClass( VisualSurvey, member );

       /************************************
        ****  CLASS: SlipSurface
        ************************************/
MakeClass( SlipSurface, member );

/************** METHOD: CalDistanceX   **************/
MakeMethod( SlipSurface, CalDistanceX, [],
    {
    Let [area ExcavationArea:MaxDepthArea]
        {
        area:GravityPointX - Self:CoordinateX1;
        };
    } );

/************** METHOD: MElevation   **************/
MakeMethod( SlipSurface, MElevation, [],
    {
    ( Self:CoordinateZ1 + Self:CoordinateZ2 ) / 2.0;
    } );
MakeSlot( SlipSurface:MeasurementPoint );
MakeSlot( SlipSurface:SurfaceDepth );
```

```
SetSlotComment( SlipSurface:SurfaceDepth, "This is a elevation of SlipSurface at the point
of the Panel.

(m)

" );
SetSlotOption( SlipSurface:SurfaceDepth, VALUE_TYPE, NUMBER );
SetSlotOption( SlipSurface:SurfaceDepth, IF_NEEDED, MElevation );

        /************************************
         ****  CLASS: MeasurePoint
         ************************************/
MakeClass( MeasurePoint, member );

/************** METHOD: CalDistanceX   **************/
MakeMethod( MeasurePoint, CalDistanceX, [],
    {
    Let [area ExcavationArea:MaxDepthArea]
        {
        area:GravityPointX - Self:CoordinateX1;
        };
    } );

/************** METHOD: CalDistanceY   **************/
MakeMethod( MeasurePoint, CalDistanceY, [],
    {
    Let [area ExcavationArea:MaxDepthArea]
        {
        area:GravityPointY - Self:CoordinateY1;
        };
    } );

/************** METHOD: CalZ   **************/
MakeMethod( MeasurePoint, CalZ, [],
    {
    Self:PanelName:CoordinateZ1;
    } );
SetSlotOption( MeasurePoint:CoordinateZ1, IF_NEEDED, CalZ );
MakeSlot( MeasurePoint:FacilityDistance );
MakeSlot( MeasurePoint:FailurePattern );
SetSlotOption( MeasurePoint:FailurePattern, ALLOWABLE_VALUES, NotDecided, None,
TotalSliding, WallFailure, AnchorFailure, Heaving, Settlement );
MakeSlot( MeasurePoint:MeasurementType );
MakeSlot( MeasurePoint:AddNumInclino );
SetSlotOption( MeasurePoint:AddNumInclino, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:Others );
MakeSlot( MeasurePoint:PanelEdge );
MakeSlot( MeasurePoint:PanelName );
SetSlotOption( MeasurePoint:PanelName, ALLOWABLE_VALUES, Panel1, Panel2, Panel3, Panel4 );
MakeSlot( MeasurePoint:PriorityNumber );
MakeSlot( MeasurePoint:SlipSurface );
MakeSlot( MeasurePoint:SlipSurfaceDepth );
MakeSlot( MeasurePoint:SurfacePosition );
MakeSlot( MeasurePoint:NumInclinometer );
SetSlotOption( MeasurePoint:NumInclinometer, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NumLoadCell );
SetSlotOption( MeasurePoint:NumLoadCell, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NumEarthPressureGauge );
```

```
SetSlotOption( MeasurePoint:NumEarthPressureGauge, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NumWaterPressureGauge );
SetSlotOption( MeasurePoint:NumWaterPressureGauge, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NumSettlementGauge );
SetSlotOption( MeasurePoint:NumSettlementGauge, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NumStressGauge );
SetSlotOption( MeasurePoint:NumStressGauge, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NumInclinoWall );
MakeSlot( MeasurePoint:NumInclinoGround );
MakeSlot( MeasurePoint:AddInclino1X );
MakeSlot( MeasurePoint:AddInclino1Y );
MakeSlot( MeasurePoint:AddInclino2X );
MakeSlot( MeasurePoint:AddInclino2Y );
MakeSlot( MeasurePoint:ExDistanceX );
SetSlotOption( MeasurePoint:ExDistanceX, VALUE_TYPE, NUMBER );
SetSlotOption( MeasurePoint:ExDistanceX, IF_NEEDED, CalDistanceX );
MakeSlot( MeasurePoint:ExDistanceY );
SetSlotOption( MeasurePoint:ExDistanceY, VALUE_TYPE, NUMBER );
SetSlotOption( MeasurePoint:ExDistanceY, IF_NEEDED, CalDistanceY );
MakeSlot( MeasurePoint:NSupportZ );
SetSlotOption( MeasurePoint:NSupportZ, MULTIPLE );
SetSlotOption( MeasurePoint:NSupportZ, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NStressGaugeZ );
SetSlotOption( MeasurePoint:NStressGaugeZ, MULTIPLE );
SetSlotOption( MeasurePoint:NStressGaugeZ, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NEPressGaugeZ );
SetSlotOption( MeasurePoint:NEPressGaugeZ, MULTIPLE );
SetSlotOption( MeasurePoint:NEPressGaugeZ, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:NWPressGaugeZ );
SetSlotOption( MeasurePoint:NWPressGaugeZ, MULTIPLE );
SetSlotOption( MeasurePoint:NWPressGaugeZ, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:InclinoDepth );
SetSlotOption( MeasurePoint:InclinoDepth, VALUE_TYPE, NUMBER );
MakeSlot( MeasurePoint:AddSettleGaugeX );
MakeSlot( MeasurePoint:AddSettleGaugeY );

        /************************************
         ****  CLASS: FailurePattern
         ************************************/
MakeClass( FailurePattern, Root );

/************** METHOD: CalDanger   **************/
MakeMethod( FailurePattern, CalDanger, [],
    ( Self:SafeX - Self:SafeY ) / Self:UncertainZ );
MakeSlot( FailurePattern:MeasurementType );
MakeSlot( FailurePattern:SlipSurface );
MakeSlot( FailurePattern:NValueCurve );
SetSlotComment( FailurePattern:NValueCurve, "Maximam Value for Each layer. Low or High.

If all layer behind panel is that Clay < 20, Sand < 30,

this slot value should be written " Low".



" );
```

```
MakeSlot( FailurePattern:EdgeSoilStrength );
MakeSlot( FailurePattern:PenetrationDepth );
MakeSlot( FailurePattern:PoreWaterPressure );
MakeSlot( FailurePattern:EdgeLayer );
MakeSlot( FailurePattern:LayerCondition );
SetSlotComment( FailurePattern:LayerCondition, "Condition of layers behaind panel.

Bad or Good

If all layers are clay, it is bad. " );
MakeSlot( FailurePattern:EdgeCondition );
SetSlotComment( FailurePattern:EdgeCondition, "Condition of layer of panel edge.

Bad or Good.

If penetrationdepth isn't enough and Nvalue is low, it is Bad." );

        /************************************
         ****  CLASS: TotalSliding
         ************************************/
MakeClass( TotalSliding, FailurePattern );
TotalSliding:MeasurementType = Type2;
TotalSliding:NValueCurve = Low;
TotalSliding:EdgeSoilStrength = NotEnough;
TotalSliding:PenetrationDepth = Enough;
TotalSliding:PoreWaterPressure = Low;

        /************************************
         ****  CLASS: WallFailure
         ************************************/
MakeClass( WallFailure, FailurePattern );
WallFailure:MeasurementType = Type3;
WallFailure:SlipSurface = BehindPanel;
WallFailure:NValueCurve = Low;
WallFailure:EdgeSoilStrength = Enough;
WallFailure:PenetrationDepth = Enough;
WallFailure:PoreWaterPressure = Low;

        /************************************
         ****  CLASS: AnchorFailure
         ************************************/
MakeClass( AnchorFailure, FailurePattern );
AnchorFailure:MeasurementType = Type3;
AnchorFailure:SlipSurface = AboveSupportingPoint;
AnchorFailure:NValueCurve = High;
AnchorFailure:EdgeSoilStrength = Enough;
AnchorFailure:PenetrationDepth = Enough;
AnchorFailure:PoreWaterPressure = Low;

        /************************************
         ****  CLASS: Heaving
         ************************************/
MakeClass( Heaving, FailurePattern );
Heaving:MeasurementType = Type4;
Heaving:SlipSurface = None;
Heaving:NValueCurve = Low;
Heaving:EdgeSoilStrength = NotEnough;
Heaving:PenetrationDepth = Short;
```

```
Heaving:PoreWaterPressure = Low;
Heaving:EdgeLayer = Clay;

        /************************************
         ****  CLASS: Settlement
         ************************************/
MakeClass( Settlement, FailurePattern );
Settlement:MeasurementType = Type3;
Settlement:SlipSurface = NotNecessary;
Settlement:NValueCurve = Low;
Settlement:EdgeSoilStrength = NoRelation;
Settlement:PenetrationDepth = Enough;
Settlement:PoreWaterPressure = NoRelation;
Settlement:EdgeLayer = NoRelation;

        /************************************
         ****  CLASS: ProjectOutline
         ************************************/
MakeClass( ProjectOutline, Root );
MakeSlot( ProjectOutline:NumMeasurePoint );
SetSlotOption( ProjectOutline:NumMeasurePoint, VALUE_TYPE, NUMBER );
ProjectOutline:NumMeasurePoint = 4;

        /************************************
         ****  CLASS: MeasurePlan
         ************************************/
MakeClass( MeasurePlan, Root );

        /************************************
         ****  CLASS: MeasureType
         ************************************/
MakeClass( MeasureType, MeasurePlan );

/************** METHOD: CalNumSettlementGauge   **************/
MakeMethod( MeasureType, CalNumSettlementGauge, [],
    {
    If ( Self:MeasurementType #= Type4 )
      Then {
           If ( Self:Others #= Null )
             Then 1
             Else 2;
           }
      Else {
           If ( Self:Others #= Null )
             Then 0
             Else 1;
           };
    } );

/************** METHOD: CalNum   **************/
MakeMethod( MeasureType, CalNum, [],
    {
    Support:NumMember;
    } );
MakeSlot( MeasureType:NumInclinometer );
SetSlotComment( MeasureType:NumInclinometer, "this needs for every measurementpoint to
analyze the behavior in case.
```

```
Setting depth depends on the edge layer condition. " );
SetSlotOption( MeasureType:NumInclinometer, VALUE_TYPE, NUMBER );
MakeSlot( MeasureType:NumLoadCell );
SetSlotComment( MeasureType:NumLoadCell, "This needs for every measurementpoint to analyze
the behavior in case.

1 to each support must be needed to estimate the earthpressure and external force." );
SetSlotOption( MeasureType:NumLoadCell, VALUE_TYPE, NUMBER );
SetSlotOption( MeasureType:NumLoadCell, IF_NEEDED, CalNum );
MakeSlot( MeasureType:NumStressGauge );
SetSlotComment( MeasureType:NumStressGauge, "This should be put 2 ~ 5m pitch above the
half point of penetration depth, because large moment doesn't occure around the bottom,

and be put on front and back bar same depth to get moments.

" );
SetSlotOption( MeasureType:NumStressGauge, VALUE_TYPE, NUMBER );
MakeSlot( MeasureType:NumSettlementGauge );
SetSlotComment( MeasureType:NumSettlementGauge, "This should be set the center of deepest
excavation area." );
SetSlotOption( MeasureType:NumSettlementGauge, VALUE_TYPE, NUMBER );
MakeSlot( MeasureType:NumEarthPressureGauge );
SetSlotComment( MeasureType:NumEarthPressureGauge, "This is  recommended to put both side
of panel and with porewatterpressure gauge.

It should be put aroud 5 m pitch or 1 to each layer, and not be behind support." );
SetSlotOption( MeasureType:NumEarthPressureGauge, VALUE_TYPE, NUMBER );
MakeSlot( MeasureType:NumWaterPressureGauge );
SetSlotComment( MeasureType:NumWaterPressureGauge, "This should be put with earthpressure
gauge." );
SetSlotOption( MeasureType:NumWaterPressureGauge, VALUE_TYPE, NUMBER );

        /************************************
         ****  CLASS: MeasureManage
         ************************************/
MakeClass( MeasureManage, Root );



/*********************************************************/
/**         ALL INSTANCES ARE SAVED BELOW              **/
/*********************************************************/

        /************************************
         ****  INSTANCE: HolizontalSliding
         ************************************/
MakeInstance( HolizontalSliding, TotalSliding );
HolizontalSliding:SlipSurface = UnderPanel;

        /************************************
         ****  INSTANCE: RotationalSliding
         ************************************/
MakeInstance( RotationalSliding, TotalSliding );
RotationalSliding:SlipSurface = NotNecessary;

        /************************************
         ****  INSTANCE: BulgingDeformation
         ************************************/
```

```
MakeInstance( BulgingDeformation, WallFailure );

        /************************************
         ****  INSTANCE: ShearingDeformation
         ************************************/
MakeInstance( ShearingDeformation, WallFailure );

        /************************************
         ****  INSTANCE: LoadIncrease
         ************************************/
MakeInstance( LoadIncrease, AnchorFailure );

        /************************************
         ****  INSTANCE: LoadDecrease
         ************************************/
MakeInstance( LoadDecrease, AnchorFailure );

        /************************************
         ****  INSTANCE: WallCrack
         ************************************/
MakeInstance( WallCrack, WallFailure );

        /************************************
         ****  INSTANCE: WaterLeakage
         ************************************/
MakeInstance( WaterLeakage, WallFailure );

        /************************************
         ****  INSTANCE: Panel1
         ************************************/
MakeInstance( Panel1, Panel );
Panel1:Thickness = 0.60;
Panel1:CoordinateZ1 = 10.0;
Panel1:ExcavationDepth = 20.0;
Panel1:CoordinateX1 = 0.0;
Panel1:CoordinateY1 = 150.0;
Panel1:CoordinateX2 = 0.0;
Panel1:CoordinateY2 = 0.0;
Panel1:CoordinateZ2 = -13.0;
Panel1:MeasurePoint = Center;
MakeSlot( Panel1:sample );
SetSlotOption( Panel1:sample, MULTIPLE );
SetSlotOption( Panel1:sample, VALUE_TYPE, NUMBER );
SetValue( Panel1:sample, 5, 5 );
Panel1:ExistFacility = Facility2;
Panel1:PriorityOrder = 1;
Panel1:ID = Panel1;
Panel1:FacilityDistance = 30;
Panel1:NValueCurve = High;
Panel1:LayerCondition = Good;
Panel1:EdgeCondition = Good;
Panel1:EdgeLayer = Layer4;
Panel1:EdgePorePressure = Low;
Panel1:EdgeSoil = Sand;
Panel1:PenetrationCondition = Enough;
Panel1:NumMember = 1;

        /************************************
```

```
        ****  INSTANCE: Panel2
        ************************************/
MakeInstance( Panel2, Panel );
Panel2:Thickness = 0.60;
Panel2:CoordinateZ1 = 10.0;
Panel2:ExcavationDepth = 20.0;
Panel2:CoordinateX1 = 0.0;
Panel2:CoordinateY1 = 0.0;
Panel2:CoordinateX2 = 100.0;
Panel2:CoordinateY2 = 0.0;
Panel2:CoordinateZ2 = -13.0;
Panel2:MeasurePoint = Center;
Panel2:ExistFacility = Null;
Panel2:PriorityOrder = 3;
Panel2:ID = Panel2;
Panel2:NValueCurve = High;
Panel2:LayerCondition = Good;
Panel2:EdgeCondition = Good;
Panel2:EdgeLayer = Layer4;
Panel2:EdgePorePressure = Low;
Panel2:EdgeSoil = Sand;
Panel2:PenetrationCondition = Enough;
Panel2:FacilityDistance = 500.0;
Panel2:NumMember = 1;

       /************************************
        ****  INSTANCE: Panel3
        ************************************/
MakeInstance( Panel3, Panel );
Panel3:Thickness = 0.60;
Panel3:CoordinateZ1 = 10.0;
Panel3:ExcavationDepth = 20.0;
Panel3:CoordinateX1 = 100.0;
Panel3:CoordinateY1 = 0.0;
Panel3:CoordinateX2 = 100.0;
Panel3:CoordinateY2 = 100.0;
Panel3:CoordinateZ2 = -13.0;
Panel3:MeasurePoint = Center;
Panel3:ExistFacility = Facility1;
Panel3:PriorityOrder = 2;
Panel3:ID = Panel3;
Panel3:FacilityDistance = 40.311289;
Panel3:NValueCurve = High;
Panel3:LayerCondition = Good;
Panel3:EdgeCondition = Good;
Panel3:EdgeLayer = Layer4;
Panel3:EdgePorePressure = Low;
Panel3:EdgeSoil = Sand;
Panel3:PenetrationCondition = Enough;
Panel3:NumMember = 1;

       /************************************
        ****  INSTANCE: Panel4
        ************************************/
MakeInstance( Panel4, Panel );
Panel4:Thickness = 0.60;
Panel4:CoordinateZ1 = 10.0;
Panel4:ExcavationDepth = 20.0;
```

```
Panel4:CoordinateX1 = 100.0;
Panel4:CoordinateY1 = 100.0;
Panel4:CoordinateX2 = 50.0;
Panel4:CoordinateY2 = 100.0;
Panel4:CoordinateZ2 = -13.0;
Panel4:MeasurePoint = Center;
Panel4:ExistFacility = Null;
Panel4:PriorityOrder = 4;
Panel4:ID = Panel4;
Panel4:NValueCurve = High;
Panel4:LayerCondition = Good;
Panel4:EdgeCondition = Good;
Panel4:FacilityDistance = 500.00;
Panel4:EdgeLayer = Layer4;
Panel4:NumMember = 1;
Panel4:EdgeSoil = Sand;
Panel4:PenetrationCondition = Enough;
Panel4:EdgePorePressure = Low;

        /************************************
         ****  INSTANCE: Panel5
         ************************************/
MakeInstance( Panel5, Panel );
Panel5:Thickness = 0.60;
Panel5:CoordinateZ1 = 10.0;
Panel5:ExcavationDepth = 10.0;
Panel5:CoordinateX1 = 50.0;
Panel5:CoordinateY1 = 100.0;
Panel5:CoordinateX2 = 50.0;
Panel5:CoordinateY2 = 150.0;
Panel5:CoordinateZ2 = -13.0;
Panel5:MeasurePoint = None;
Panel5:ExistFacility = Null;
Panel5:PriorityOrder = 1;
Panel5:ID = Panel5;
Panel5:NumMember = 1;

        /************************************
         ****  INSTANCE: Panel6
         ************************************/
MakeInstance( Panel6, Panel );
Panel6:Thickness = 0.60;
Panel6:CoordinateZ1 = 10.0;
Panel6:ExcavationDepth = 10.0;
Panel6:CoordinateX1 = 50.0;
Panel6:CoordinateY1 = 150.0;
Panel6:CoordinateX2 = 0.0;
Panel6:CoordinateY2 = 150.0;
Panel6:CoordinateZ2 = -13.0;
Panel6:MeasurePoint = None;
Panel6:ExistFacility = Null;
Panel6:PriorityOrder = 1;
Panel6:ID = Panel6;
Panel6:NumMember = 1;

        /************************************
         ****  INSTANCE: Area2
         ************************************/
```

```
MakeInstance( Area2, ExcavationArea );
Area2:CoordinateZ1 = 10.0;
Area2:ExcavationDepth = 20.0;
Area2:CoordinateX1 = 0.0;
Area2:CoordinateY1 = 0.0;
Area2:CoordinateX2 = 100.0;
Area2:CoordinateY2 = 100.0;
Area2:CoordinateZ2 = -10.0;
Area2:ID = Area2;

        /************************************
         ****  INSTANCE: Area1
         ************************************/
MakeInstance( Area1, ExcavationArea );
Area1:CoordinateZ1 = 10.0;
Area1:ExcavationDepth = 10.0;
Area1:CoordinateX1 = 0.0;
Area1:CoordinateY1 = 100.0;
Area1:CoordinateX2 = 50.0;
Area1:CoordinateY2 = 150.0;
Area1:CoordinateZ2 = -10.0;
Area1:ID = Area1;

        /************************************
         ****  INSTANCE: Facility1
         ************************************/
MakeInstance( Facility1, Facility );
Facility1:FacilityName = Hotel;
Facility1:CoordinateZ1 = 0.0;
Facility1:CoordinateX1 = 120.0;
Facility1:CoordinateY1 = 10.0;
Facility1:CoordinateX2 = 120.0;
Facility1:CoordinateY2 = 20.0;
Facility1:CoordinateZ2 = 0.0;
Facility1:DistancePanel = 40.311289;
Facility1:NearestPanel = Panel3;
Facility1:ImportanceNumber = 10.0;
Facility1:ID = Facility1;

        /************************************
         ****  INSTANCE: Facility2
         ************************************/
MakeInstance( Facility2, Facility );
Facility2:FacilityName = UtilityLine;
Facility2:CoordinateZ1 = 0.0;
Facility2:CoordinateX1 = -20.0;
Facility2:CoordinateY1 = 0.0;
Facility2:CoordinateX2 = -40.0;
Facility2:CoordinateY2 = 150.0;
Facility2:CoordinateZ2 = 0.0;
Facility2:DistancePanel = 30;
Facility2:NearestPanel = Panel1;
Facility2:ImportanceNumber = 8.0;
Facility2:ID = Facility2;

        /************************************
         ****  INSTANCE: Button2
         ************************************/
```

```
MakeInstance( Button2, Button );
Button2:X = -17;
Button2:Y = -1;
Button2:Title = Initialization;
Button2:Visible = TRUE;
Button2:Width = 157;
Button2:Height = 27;
Button2:Action = ZeroInitialize;
 ResetImage (  Button2  );

       /************************************
        ****  INSTANCE: Button4
        ************************************/
MakeInstance( Button4, Button );
Button4:X = 389;
Button4:Y = 339;
Button4:Title = SetMeasurementPoint;
Button4:Visible = TRUE;
Button4:Width = 162;
Button4:Height = 27;
Button4:Action = SetMeasurementPoint;
Button4:FileName = NULL;
Button4:ShowBorder = FALSE;
 ResetImage (  Button4  );

       /************************************
        ****  INSTANCE: Button1
        ************************************/
MakeInstance( Button1, Button );
Button1:X = -19;
Button1:Y = 24;
Button1:Title = SetValue;
Button1:Visible = TRUE;
Button1:Width = 160;
Button1:Height = 27;
Button1:Action = SetInitialValue;
 ResetImage (  Button1  );

       /************************************
        ****  INSTANCE: Layer1
        ************************************/
MakeInstance( Layer1, Layer );
Layer1:NValue = 3.0;
Layer1:Poisson = 0.5;
Layer1:SoilKind = Clay;
Layer1:SoilStrengthC = 2.0;
Layer1:SoilStrengthFai = 10.0;
Layer1:UnitWeight = 1.8;
Layer1:WaterPressure = 0.0;
Layer1:CoordinateX1 = 50.0;
Layer1:CoordinateX2 = 150.0;
Layer1:CoordinateY1 = 50.0;
Layer1:CoordinateY2 = 50.0;
Layer1:CoordinateZ1 = 10.0;
Layer1:CoordinateZ2 = 5.0;
Layer1:ID = Layer1;
SetValue( Layer1:Point1, 50.0, 50.0, 10.0 );
SetValue( Layer1:Point2, 150.0, 50.0, 10.0 );
```

```
SetValue( Layer1:Point3, 150.0, 50.0, 5.0 );
SetValue( Layer1:Point4, 50.0, 50.0, 5.0 );
SetValue( Layer1:GravityPoint, 100, 50 );

        /************************************
         ****  INSTANCE: Layer2
         ************************************/
MakeInstance( Layer2, Layer );
Layer2:NValue = 20.0;
Layer2:Poisson = 0.3;
Layer2:SoilKind = Sand;
Layer2:SoilStrengthC = 0.0;
Layer2:SoilStrengthFai = 30.0;
Layer2:UnitWeight = 2.0;
Layer2:WaterPressure = 0.0;
Layer2:CoordinateX1 = 50.0;
Layer2:CoordinateX2 = 150.0;
Layer2:CoordinateY1 = 50.0;
Layer2:CoordinateY2 = 50.0;
Layer2:CoordinateZ1 = 5.0;
Layer2:CoordinateZ2 = 0.0;
Layer2:ID = Layer2;
Layer2:NValueCurve = Low;
SetValue( Layer2:Point1, 50.0, 50.0, 5.0 );
SetValue( Layer2:Point2, 150.0, 50.0, 5.0 );
SetValue( Layer2:Point3, 150.0, 50.0, 0.0 );
SetValue( Layer2:Point4, 50.0, 50.0, 0.0 );
SetValue( Layer2:GravityPoint, 100, 50 );

        /************************************
         ****  INSTANCE: Layer3
         ************************************/
MakeInstance( Layer3, Layer );
Layer3:NValue = 10.0;
Layer3:Poisson = 0.5;
Layer3:SoilKind = Clay;
Layer3:SoilStrengthC = 2.0;
Layer3:SoilStrengthFai = 10.0;
Layer3:UnitWeight = 1.8;
Layer3:WaterPressure = 0.0;
Layer3:CoordinateX1 = 50.0;
Layer3:CoordinateX2 = 150.0;
Layer3:CoordinateY1 = 50.0;
Layer3:CoordinateY2 = 50.0;
Layer3:CoordinateZ1 = 0.0;
Layer3:CoordinateZ2 = -10.0;
Layer3:ID = Layer3;
Layer3:NValueCurve = Low;
SetValue( Layer3:Point1, 50.0, 50.0, 0.0 );
SetValue( Layer3:Point2, 150.0, 50.0, 0.0 );
SetValue( Layer3:Point3, 150.0, 50.0, -10.0 );
SetValue( Layer3:Point4, 50.0, 50.0, -10.0 );
SetValue( Layer3:GravityPoint, 100, 50 );

        /************************************
         ****  INSTANCE: Layer4
         ************************************/
MakeInstance( Layer4, Layer );
```

```
Layer4:NValue = 30.0;
Layer4:Poisson = 0.3;
Layer4:SoilKind = Sand;
Layer4:SoilStrengthC = 2.0;
Layer4:SoilStrengthFai = 10.0;
Layer4:UnitWeight = 2.0;
Layer4:WaterPressure = 20.0;
Layer4:CoordinateX1 = 50.0;
Layer4:CoordinateX2 = 150.0;
Layer4:CoordinateY1 = 50.0;
Layer4:CoordinateY2 = 50.0;
Layer4:CoordinateZ1 = -10.0;
Layer4:CoordinateZ2 = -15.0;
Layer4:ID = Layer4;
Layer4:NValueCurve = Low;
SetValue( Layer4:Point1, 50.0, 50.0, -10.0 );
SetValue( Layer4:Point2, 150.0, 50.0, -10.0 );
SetValue( Layer4:Point3, 150.0, 50.0, -15.0 );
SetValue( Layer4:Point4, 50.0, 50.0, -15.0 );
SetValue( Layer4:GravityPoint, 100, 50 );

        /************************************
         ****  INSTANCE: Layer5
         ************************************/
MakeInstance( Layer5, Layer );
Layer5:NValue = 50.0;
Layer5:Poisson = 0.4;
Layer5:SoilKind = Tertiary;
Layer5:SoilStrengthC = 2.0;
Layer5:SoilStrengthFai = 10.0;
Layer5:UnitWeight = 2.2;
Layer5:WaterPressure = 0.0;
Layer5:CoordinateX1 = 50.0;
Layer5:CoordinateX2 = 150.0;
Layer5:CoordinateY1 = 50.0;
Layer5:CoordinateY2 = 50.0;
Layer5:CoordinateZ1 = -15.0;
Layer5:CoordinateZ2 = -25.0;
Layer5:ID = Layer5;
Layer5:NValueCurve = High;
SetValue( Layer5:Point1, 50.0, 50.0, -15.0 );
SetValue( Layer5:Point2, 150.0, 50.0, -15.0 );
SetValue( Layer5:Point3, 150.0, 50.0, -25.0 );
SetValue( Layer5:Point4, 50.0, 50.0, -25.0 );
SetValue( Layer5:GravityPoint, 100, 50 );

        /************************************
         ****  INSTANCE: Support1
         ************************************/
MakeInstance( Support1, Support );
Support1:ID = Support1;
Support1:InitialStress = 200.0;
Support1:CoordinateX1 = 100.0;
Support1:CoordinateX2 = 125.0;
Support1:CoordinateY1 = 50.0;
Support1:CoordinateY2 = 50.0;
Support1:CoordinateZ1 = 5.0;
Support1:CoordinateZ2 = -20.0;
```

```
Support1:NumMember = 1;
SetValue( Support1:Point1, 100.0, 50.0, 5.0 );
SetValue( Support1:Point2, 125.0, 50.0, 5.0 );
SetValue( Support1:Point3, 125.0, 50.0, -20.0 );
SetValue( Support1:Point4, 100.0, 50.0, -20.0 );
SetValue( Support1:GravityPoint, 112.500000, 50 );

        /************************************
         ****  INSTANCE: Support2
         ************************************/
MakeInstance( Support2, Support );
Support2:ID = Support2;
Support2:InitialStress = 250.0;
Support2:CoordinateX1 = 100.0;
Support2:CoordinateX2 = 120.0;
Support2:CoordinateY1 = 50.0;
Support2:CoordinateY2 = 50.0;
Support2:CoordinateZ1 = 0.0;
Support2:CoordinateZ2 = -20.0;
Support2:NumMember = 1;
SetValue( Support2:Point1, 100.0, 50.0, 0.0 );
SetValue( Support2:Point2, 120.0, 50.0, 0.0 );
SetValue( Support2:Point3, 120.0, 50.0, -20.0 );
SetValue( Support2:Point4, 100.0, 50.0, -20.0 );
SetValue( Support2:GravityPoint, 110, 50 );

        /************************************
         ****  INSTANCE: Support3
         ************************************/
MakeInstance( Support3, Support );
Support3:ID = Support3;
Support3:InitialStress = 300.0;
Support3:CoordinateX1 = 100.0;
Support3:CoordinateX2 = 115.0;
Support3:CoordinateY1 = 50.0;
Support3:CoordinateY2 = 50.0;
Support3:CoordinateZ1 = -5.0;
Support3:CoordinateZ2 = -20.0;
Support3:NumMember = 1;
SetValue( Support3:Point1, 100.0, 50.0, -5.0 );
SetValue( Support3:Point2, 115.0, 50.0, -5.0 );
SetValue( Support3:Point3, 115.0, 50.0, -20.0 );
SetValue( Support3:Point4, 100.0, 50.0, -20.0 );
SetValue( Support3:GravityPoint, 107.500000, 50 );

        /************************************
         ****  INSTANCE: SlipSurface1
         ************************************/
MakeInstance( SlipSurface1, SlipSurface );
SlipSurface1:ID = SlipSurface1;
SlipSurface1:MeasurementPoint = Panel3;
SlipSurface1:CoordinateX1 = 100.0;
SlipSurface1:CoordinateX2 = 150.0;
SlipSurface1:CoordinateY1 = 50.0;
SlipSurface1:CoordinateY2 = 50.0;
SlipSurface1:CoordinateZ1 = -10.0;
SlipSurface1:CoordinateZ2 = -20.0;
SetValue( SlipSurface1:Point1, 100.0, 50.0, -10.0 );
```

```
SetValue( SlipSurface1:Point2, 150.0, 50.0, -10.0 );
SetValue( SlipSurface1:Point3, 150.0, 50.0, -20.0 );
SetValue( SlipSurface1:Point4, 100.0, 50.0, -20.0 );
SetValue( SlipSurface1:GravityPoint, 125, 50 );

       /************************************
        ****  INSTANCE: SlipSurface2
        ************************************/
MakeInstance( SlipSurface2, SlipSurface );
SlipSurface2:ID = SlipSurface2;
SlipSurface2:MeasurementPoint = Panel4;
SlipSurface2:CoordinateX1 = 50.0;
SlipSurface2:CoordinateX2 = 50.0;
SlipSurface2:CoordinateY1 = 50.0;
SlipSurface2:CoordinateY2 = -50.0;
SlipSurface2:CoordinateZ1 = -20.0;
SlipSurface2:CoordinateZ2 = -20.0;
SetValue( SlipSurface2:Point1, 50.0, 50.0, -20.0 );
SetValue( SlipSurface2:Point2, 50.0, -50.0, -20.0 );
SetValue( SlipSurface2:Point3, 50.0, -50.0, -20.0 );
SetValue( SlipSurface2:Point4, 50.0, 50.0, -20.0 );
SetValue( SlipSurface2:GravityPoint, 50, 0 );

       /************************************
        ****  INSTANCE: SlipSurface3
        ************************************/
MakeInstance( SlipSurface3, SlipSurface );
SlipSurface3:ID = SlipSurface3;
SlipSurface3:MeasurementPoint = Panel1;
SlipSurface3:CoordinateX1 = 50.0;
SlipSurface3:CoordinateX2 = -50.0;
SlipSurface3:CoordinateY1 = 50.0;
SlipSurface3:CoordinateY2 = 50.0;
SlipSurface3:CoordinateZ1 = 0.0;
SlipSurface3:CoordinateZ2 = 5.0;
SetValue( SlipSurface3:Point1, 50.0, 50.0, 0.0 );
SetValue( SlipSurface3:Point2, -50.0, 50.0, 0.0 );
SetValue( SlipSurface3:Point3, -50.0, 50.0, 5.0 );
SetValue( SlipSurface3:Point4, 50.0, 50.0, 5.0 );
SetValue( SlipSurface3:GravityPoint, 0, 50 );

       /************************************
        ****  INSTANCE: Button5
        ************************************/
MakeInstance( Button5, Button );
Button5:X = 388;
Button5:Y = 390;
Button5:Title = PredictFailurePattern;
Button5:Visible = TRUE;
Button5:Width = 162;
Button5:Height = 27;
Button5:Action = PredictFailurePattern;
 ResetImage (  Button5  );

       /************************************
        ****  INSTANCE: Button7
        ************************************/
MakeInstance( Button7, Button );
```

```
Button7:X = 389;
Button7:Y = 416;
Button7:Title = LayoutMeasureEquips;
Button7:Visible = TRUE;
Button7:Width = 159;
Button7:Height = 27;
Button7:Action = LayoutMeasureEquips;
 ResetImage (  Button7  );

       /************************************
        ****  INSTANCE: Button9
        ************************************/
MakeInstance( Button9, Button );
Button9:X = 389;
Button9:Y = 364;
Button9:Title = CheckLayers;
Button9:Visible = TRUE;
Button9:Width = 161;
Button9:Height = 27;
Button9:Action = CheckLayers;
 ResetImage (  Button9  );

       /************************************
        ****  INSTANCE: Type1
        ************************************/
MakeInstance( Type1, MeasureType );
Type1:NumInclinometer = 1;
Type1:NumLoadCell = NumSupport;
Type1:NumStressGauge = 0;
Type1:NumSettlementGauge = 0;
Type1:NumEarthPressureGauge = 0;
Type1:NumWaterPressureGauge = 0;

       /************************************
        ****  INSTANCE: Type2
        ************************************/
MakeInstance( Type2, MeasureType );
Type2:NumInclinometer = More1;
Type2:NumLoadCell = NumSupport;
Type2:NumStressGauge = 0;
Type2:NumSettlementGauge = SeeOthers;
Type2:NumEarthPressureGauge = 0;
Type2:NumWaterPressureGauge = 0;

       /************************************
        ****  INSTANCE: Type3
        ************************************/
MakeInstance( Type3, MeasureType );
Type3:NumInclinometer = More1;
Type3:NumLoadCell = NumSupport;
Type3:NumStressGauge = Some;
Type3:NumSettlementGauge = SeeOthers;
Type3:NumEarthPressureGauge = Some;
Type3:NumWaterPressureGauge = Some;

       /************************************
        ****  INSTANCE: Type4
        ************************************/
```

```
MakeInstance( Type4, MeasureType );
Type4:NumInclinometer = 1;
Type4:NumLoadCell = NumSupport;
Type4:NumStressGauge = 0;
Type4:NumSettlementGauge = 1;
Type4:NumEarthPressureGauge = 0;
Type4:NumWaterPressureGauge = Some;

       /************************************
        ****  INSTANCE: Others
        ************************************/
MakeInstance( Others, MeasureType );
Others:NumInclinometer = 0;
Others:NumLoadCell = 0;
Others:NumStressGauge = 0;
Others:NumSettlementGauge = 1;
Others:NumEarthPressureGauge = 0;
Others:NumWaterPressureGauge = 0;

       /************************************
        ****  INSTANCE: Button3
        ************************************/
MakeInstance( Button3, Button );
Button3:X = -16;
Button3:Y = 48;
Button3:Title = MeasurementPlan;
Button3:Visible = TRUE;
Button3:Width = 158;
Button3:Height = 103;
Button3:Action = MeasurementPlan;
 ResetImage (  Button3  );

       /************************************
        ****  INSTANCE: StateBox1
        ************************************/
MakeInstance( StateBox1, StateBox );
StateBox1:X = 140;
StateBox1:Y = -3;
StateBox1:Visible = TRUE;
StateBox1:Title = "M1-FailurePattern";
StateBox1:Value = NotDecided;
StateBox1:Width = 128;
StateBox1:Height = 186;
StateBox1:Owner = MeasurePoint1;
StateBox1:OwnerSlot = FailurePattern;
 ResetImage (  StateBox1  );

       /************************************
        ****  INSTANCE: StateBox2
        ************************************/
MakeInstance( StateBox2, StateBox );
StateBox2:X = 266;
StateBox2:Y = -3;
StateBox2:Visible = TRUE;
StateBox2:Width = 130;
StateBox2:Height = 187;
StateBox2:Title = "M2-FaliurePattern";
StateBox2:Value = NotDecided;
```

```
StateBox2:Owner = MeasurePoint2;
StateBox2:OwnerSlot = FailurePattern;
 ResetImage (  StateBox2  );

       /************************************
        ****  INSTANCE: StateBox3
        ************************************/
MakeInstance( StateBox3, StateBox );
StateBox3:X = 392;
StateBox3:Y = -3;
StateBox3:Visible = TRUE;
StateBox3:Title = "M3-FailurePattern";
StateBox3:Value = NotDecided;
StateBox3:Width = 132;
StateBox3:Height = 187;
StateBox3:Owner = MeasurePoint3;
StateBox3:OwnerSlot = FailurePattern;
 ResetImage (  StateBox3  );

       /************************************
        ****  INSTANCE: StateBox6
        ************************************/
MakeInstance( StateBox6, StateBox );
StateBox6:X = 18;
StateBox6:Y = 313;
StateBox6:Visible = TRUE;
StateBox6:Title = "M1-PanelName";
StateBox6:Width = 126;
StateBox6:Height = 128;
StateBox6:Value = Panel1;
StateBox6:Owner = MeasurePoint1;
StateBox6:OwnerSlot = PanelName;
 ResetImage (  StateBox6  );

       /************************************
        ****  INSTANCE: StateBox7
        ************************************/
MakeInstance( StateBox7, StateBox );
StateBox7:X = 143;
StateBox7:Y = 313;
StateBox7:Visible = TRUE;
StateBox7:Title = "M2-PanelName";
StateBox7:Width = 131;
StateBox7:Height = 128;
StateBox7:Value = Panel3;
StateBox7:Owner = MeasurePoint2;
StateBox7:OwnerSlot = PanelName;
 ResetImage (  StateBox7  );

       /************************************
        ****  INSTANCE: StateBox8
        ************************************/
MakeInstance( StateBox8, StateBox );
StateBox8:X = 271;
StateBox8:Y = 313;
StateBox8:Visible = TRUE;
StateBox8:Title = "M3-PanelName";
StateBox8:Width = 119;
```

```
StateBox8:Height = 129;
StateBox8:Value = Panel2;
StateBox8:Owner = MeasurePoint3;
StateBox8:OwnerSlot = PanelName;
 ResetImage (  StateBox8  );

       /************************************
        ****  INSTANCE: MeasurePoint1
        ************************************/
MakeInstance( MeasurePoint1, MeasurePoint );
MeasurePoint1:PanelName = Panel1;
SetSlotOption( MeasurePoint1:PanelName, IMAGE, StateBox6 );
MeasurePoint1:CoordinateX1 = 0;
MeasurePoint1:CoordinateY1 = 75;
MeasurePoint1:CoordinateX2 = 0;
MeasurePoint1:CoordinateY2 = 75;
MeasurePoint1:PriorityNumber = 1;
MeasurePoint1:PanelEdge = -13.0;
MeasurePoint1:FacilityDistance = 30;
MeasurePoint1:ExcavationDepth = 20.0;
MeasurePoint1:FailurePattern = NotDecided;
SetSlotOption( MeasurePoint1:FailurePattern, IMAGE, StateBox1 );
MeasurePoint1:MeasurementType = Type3;
MeasurePoint1:SlipSurface = SlipSurface3;
MeasurePoint1:SlipSurfaceDepth = 2.500000;
MeasurePoint1:SurfacePosition = Behind;
MeasurePoint1:Others = 1;
MeasurePoint1:AddNumInclino = 1;
MeasurePoint1:NumInclinometer = 2;
MeasurePoint1:NumInclinoWall = 1;
MeasurePoint1:NumInclinoGround = 1;
MeasurePoint1:NumLoadCell = 3;
MeasurePoint1:NumEarthPressureGauge = 7;
MeasurePoint1:NumWaterPressureGauge = 7;
MeasurePoint1:NumStressGauge = 14;
MeasurePoint1:NumSettlementGauge = 0;
MeasurePoint1:AddInclino1X = -4.472150;
MeasurePoint1:AddInclino1Y = 77.236075;
MeasurePoint1:AddInclino2X = None;
MeasurePoint1:AddInclino2Y = None;
MeasurePoint1:AddSettleGaugeX = -8.944250;
MeasurePoint1:AddSettleGaugeY = 79.472125;
SetValue( MeasurePoint1:NSupportZ, 5.0, 0.0, -5.0 );
SetValue( MeasurePoint1:NStressGaugeZ, 6, 3, 0, -3, -6, -9, -12, 6, 3, 0, -3, -6, -9, -12
);
SetValue( MeasurePoint1:NEPressGaugeZ, 4, -1, -6, -11, -1, -6, -11 );
SetValue( MeasurePoint1:NWPressGaugeZ, 4, -1, -6, -11, -1, -6, -11 );
MeasurePoint1:InclinoDepth = -13.0;

       /************************************
        ****  INSTANCE: MeasurePoint3
        ************************************/
MakeInstance( MeasurePoint3, MeasurePoint );
MeasurePoint3:PanelName = Panel2;
SetSlotOption( MeasurePoint3:PanelName, IMAGE, StateBox8 );
MeasurePoint3:CoordinateX1 = 50;
MeasurePoint3:CoordinateY1 = 0;
MeasurePoint3:CoordinateX2 = 50;
```

```
MeasurePoint3:CoordinateY2 = 0;
MeasurePoint3:PriorityNumber = 3;
MeasurePoint3:PanelEdge = -13.0;
MeasurePoint3:FacilityDistance = 500.0;
MeasurePoint3:ExcavationDepth = 20.0;
MeasurePoint3:FailurePattern = NotDecided;
SetSlotOption( MeasurePoint3:FailurePattern, IMAGE, StateBox3 );
MeasurePoint3:MeasurementType = Type1;
MeasurePoint3:SlipSurface = None;
MeasurePoint3:SurfacePosition = None;
MeasurePoint3:Others = 1;
MeasurePoint3:AddNumInclino = 0;
MeasurePoint3:NumInclinometer = 1;
MeasurePoint3:NumInclinoWall = 1;
MeasurePoint3:NumInclinoGround = 0;
MeasurePoint3:NumLoadCell = 3;
MeasurePoint3:NumEarthPressureGauge = 0;
MeasurePoint3:NumWaterPressureGauge = 0;
MeasurePoint3:NumStressGauge = 0;
MeasurePoint3:NumSettlementGauge = 0;
MeasurePoint3:AddInclino1X = None;
MeasurePoint3:AddInclino1Y = None;
MeasurePoint3:AddInclino2X = None;
MeasurePoint3:AddInclino2Y = None;
MeasurePoint3:AddSettleGaugeX = 50;
MeasurePoint3:AddSettleGaugeY = -10;
SetValue( MeasurePoint3:NSupportZ, 5.0, 0.0, -5.0 );
MeasurePoint3:InclinoDepth = -13.0;

       /************************************
        ****  INSTANCE: MeasurePoint2
        ************************************/
MakeInstance( MeasurePoint2, MeasurePoint );
MeasurePoint2:PanelName = Panel3;
SetSlotOption( MeasurePoint2:PanelName, IMAGE, StateBox7 );
MeasurePoint2:CoordinateX1 = 100;
MeasurePoint2:CoordinateY1 = 50;
MeasurePoint2:CoordinateX2 = 100;
MeasurePoint2:CoordinateY2 = 50;
MeasurePoint2:PriorityNumber = 2;
MeasurePoint2:PanelEdge = -13.0;
MeasurePoint2:FacilityDistance = 40.311289;
MeasurePoint2:ExcavationDepth = 20.0;
MeasurePoint2:FailurePattern = NotDecided;
SetSlotOption( MeasurePoint2:FailurePattern, IMAGE, StateBox2 );
MeasurePoint2:MeasurementType = Type2;
MeasurePoint2:SlipSurface = SlipSurface1;
MeasurePoint2:SlipSurfaceDepth = -15;
MeasurePoint2:SurfacePosition = Under;
MeasurePoint2:Others = 1;
MeasurePoint2:AddNumInclino = 1;
MeasurePoint2:NumInclinometer = 2;
MeasurePoint2:NumInclinoWall = 1;
MeasurePoint2:NumInclinoGround = 1;
MeasurePoint2:NumLoadCell = 3;
MeasurePoint2:NumEarthPressureGauge = 0;
MeasurePoint2:NumWaterPressureGauge = 0;
MeasurePoint2:NumStressGauge = 0;
```

```
MeasurePoint2:NumSettlementGauge = 0;
MeasurePoint2:AddInclino1X = 105;
MeasurePoint2:AddInclino1Y = 50;
MeasurePoint2:AddInclino2X = None;
MeasurePoint2:AddInclino2Y = None;
MeasurePoint2:AddSettleGaugeX = 110;
MeasurePoint2:AddSettleGaugeY = 50;
SetValue( MeasurePoint2:NSupportZ, 5.0, 0.0, -5.0 );
MeasurePoint2:InclinoDepth = -17;

       /************************************
        ****  INSTANCE: MeasurePoint4
        ************************************/
MakeInstance( MeasurePoint4, MeasurePoint );
MeasurePoint4:PanelName = Panel4;
MeasurePoint4:CoordinateX1 = 75;
MeasurePoint4:CoordinateY1 = 100;
MeasurePoint4:CoordinateX2 = 75;
MeasurePoint4:CoordinateY2 = 100;
MeasurePoint4:PriorityNumber = 4;
MeasurePoint4:PanelEdge = -13.0;
MeasurePoint4:FacilityDistance = 500.00;
MeasurePoint4:ExcavationDepth = 20.0;
MeasurePoint4:FailurePattern = NotDecided;
MeasurePoint4:MeasurementType = Type2;
MeasurePoint4:SlipSurface = SlipSurface2;
MeasurePoint4:SlipSurfaceDepth = -20;
MeasurePoint4:SurfacePosition = Under;
MeasurePoint4:Others = 1;
MeasurePoint4:AddNumInclino = 0;
MeasurePoint4:NumInclinometer = 1;
MeasurePoint4:NumInclinoWall = 1;
MeasurePoint4:NumInclinoGround = 0;
MeasurePoint4:NumLoadCell = 3;
MeasurePoint4:NumEarthPressureGauge = 0;
MeasurePoint4:NumWaterPressureGauge = 0;
MeasurePoint4:NumStressGauge = 0;
MeasurePoint4:NumSettlementGauge = 0;
MeasurePoint4:AddInclino1X = None;
MeasurePoint4:AddInclino1Y = None;
MeasurePoint4:AddInclino2X = None;
MeasurePoint4:AddInclino2Y = None;
MeasurePoint4:AddSettleGaugeX = 79.472125;
MeasurePoint4:AddSettleGaugeY = 108.944250;
SetValue( MeasurePoint4:NSupportZ, 5.0, 0.0, -5.0 );
MeasurePoint4:InclinoDepth = -22;
```