Modeling Software Measurement with OPM and Development of Prototype Tool

Master's Thesis
Peter Svensson

Supervisor
Martin Höst, LTH

Department of Communication Systems

Lund Institute of Technology
Lund University
Modeling Software Measurement with OPM and Development of Prototype Tool

Peter Svensson
Department of Communication Systems
Lund Institute of Technology
Lund University
September, 2004
Abstract

Object-Process Methodology (OPM) is a newly-developed methodology for system modeling. A system is described through two items in OPM, namely its objects and its processes. This approach is widely acknowledged throughout the research community that studies systems, however, it is not that common with modeling frameworks which explicitly recognize this duality.

The main part of the master’s project focused on the creation of an OPM-model describing the data collection, data processing and data analyzing processes used within Ericsson ETH/R&D in Hungary. The purpose of producing the model was that an easily comprehensible diagram structure, such as an OPM model, could help managers understand how the organization works. It would further show who is responsible for what, and aid managers in the decision-making process. The model was produced and shaped by studying the internal documents and by personal interviews.

By implementing parts of the developed model a custom designed prototype tool for representing metrics was created. The reason for producing this tool was to facilitate the creation and analyses of graphs with the motto: if all managers used the same tool for creating uniform graphs, this would allow easy comparison between project.

The valuation of this project shows that before OPM is introduced into the company a major evaluation of its usefulness is required. Also, before making a real version of the prototype tool an appraisal should be performed by the end users.
Acknowledgements

I would like to express my gratitude to the following people for their support and assistance during the making of this master’s thesis. First of all I would like to thank Zoltán Theisz, my supervisor at Ericsson, for his guidance and patience. Whenever I had any questions you would always find time to answer them. Second I would like to thank Martin Höst, for his comments and suggestions on how to improve the thesis. I would also like to thank Domonkos Asztalos and István Kerekes for making it possible for me to come to Hungary and do my master’s project at Ericsson. Further I would like to thank the staff at Software Engineering division, especially Gábor Stikkel and Attila Vida for making me feel welcome. Last but not least I want to thank my Hungarian relatives and Kristian Broman for showing me a good time in Budapest.

Thank you.

Peter Svensson
Stockholm, June 2004
Chapter 1. Introduction

1.1 Background

The Object-Process Methodology (OPM) is a newly developed method for describing systems. It was developed by Dov Dori, Associate Professor at the Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology. The methodology has been developed at least since 1994. OPM has been reworked and new features have been added. In 2000 Mor Peleg and Dov Dori extended OPM to handle real time systems. In 2002 the book *Object-Process Methodology – A Holistic Systems Paradigm* was published by Springer-Verlag. The book describes the different elements that form the OPM paradigm. The building blocks and their fields of use are explained through extensive examples. OPCAT (Object-Process CASE-Tool), a CASE-tool for creating OPM models was released in 2003.

Today the Object-Process Methodology is taught at MIT, Technion - Israel Institute of Technology and University of Rochester. In industry it has experimentally been applied at such organizations as National Semiconductors, Ford Motor Company and Gemcor, a subcontractor for Lockheed in the NASA space shuttle project. OPM has been successfully applied in a number of large-scale projects in USA, Germany and Israel. Application domains include (1) Business Process Reengineering of the technological knowledge base of a hi-tech metal cutting tool manufacturing firm, (2) designing a fully automated $800M semiconductor fabrication (FAB) facility, a re-engineered technological know-how of the world's fourth largest metal cutting tool manufacturer and (3) designing a wafer FAB cluster management system. [1]

1.2 Purpose

Ericsson ETH/R&D is a part of Ericsson Hungary LTD. The R&D department has about 400 employees, and is situated in central Budapest. The thesis work was carried out at Ericsson ETH/RK. It is a subdivision of the R&D branch and has about ten co-workers. The main purpose of this master thesis project is the creation of a model describing the different parts of the software measurement process, using the OPM paradigm. Since the paradigm is developed in the recent years it is not yet clear in which areas it can be applied.

By creating a model of the software measurement process, and evaluating the usefulness of it, an understanding of the OPM and its possible fields of use in the organization is hoped to be achieved. The model serves as a tool for understanding the different parts of the company’s software measurement process and their connections. It can be used by managers to better understand how the organizational processes work, and who is
responsible for what. If the project proves successful the OPM modeling technique might be adapted by other parts of the organization and this project could then serve as an example of how the OPM paradigm could be used for system modeling.

1.3 Working Procedure

The work performed can be divided into three phases. The first would be the modeling phase where the OPM-model of the software measurement process was created. The second phase is the implementation of the created model and the third and last phase of the project is the validation.

The first phase, the creation of the OPM model, was opened by a literature study to gain knowledge of what the OPM paradigm was and how it could be used to describe systems. Requirements written down in internal documents and other information gathered through interviews with employees were then modeled using the skills obtained during the literature study.

The implementation phase of the project began with learning about Microsoft Access. This was done by studying on-line tutorials and the book Access 2002 Bible [2]. Starting out from what was described in the OPM-model some dynamic parts were then implemented into a program, the Software Measurement Presentation Tool.

The validation phase consists of two parts. These are validation of the OPM-model and validation of the presentation tool. The validation of the OPM-model is mainly based on the author’s personal opinions and experiences. The legitimacy of the developed tool is acquired through the creation of various scenarios.

1.4 Outline

The thesis is divided into ten chapters. The first chapter is an introduction. The second one is an explanation of what the Object-Process Methodology is. It can be used as a quick tutorial by anyone who wants to learn the basic concepts of the OPM paradigm. The fourth focuses on the methods used, while chapters 3, 5, 6, 7 and 8 hold results from the project. Chapter 9 contains the evaluations while chapter 10 contains conclusions.
Chapter 2. Object-Process Methodology

This chapter gives the reader an overview of the concepts and building blocks of the Object-Process Methodology. Examples and figures are to a great extent taken from “Object-Process Methodology – A Holistic Systems Paradigm” [3], or from the presentation OPM Tutorials ICEIS [4].

2.1 What is Object-Process Methodology?

Object-Process Methodology (OPM) is a framework for modeling systems. It includes a clear and concise set of symbols that form a language enabling the expression of the system’s building blocks and their relationships.

A system is described through two items in OPM, its objects and its processes. This approach is acknowledged throughout the community that studies systems, yet there are few modeling frameworks that explicitly recognize this duality. One of OPM’s advantages is that it is so easily understood that people with no prior programming experience can understand the models. OPM has another fundamental advantage – it represents the system both graphically and textually. By doing this both sides of the brain are activated and complex systems can easier be understood.

One of the main reasons for the creation of the Object-Process Methodology is that UML, the current standard language for creating models, is sometimes unnecessary complex and software oriented. This calls for a simpler, formal, generic paradigm for systems development.

2.2 Object-Process Diagrams

Diagrams are intuitive and therefore widely used in modeling. It is a practical way of showing how the different parts of a system fit together. In OPM the diagrams are called Object-Process Diagrams (OPDs). They contain symbols for objects, processes and states, which are interconnected with different kind of links. OPM is built of three types of entities: objects, processes and states. Objects exist, and processes transform them by creating, affecting or consuming them. States are used to describe objects and are not stand-alone items.

In the OPD an object is represented by a square and a process is represented by an ellipse. The name of the object or process is recorded inside the corresponding symbol. A state is represented by a rounded rectangle inside an object. Figure 2-1 is an example of a simple OPD showing how the process Marrying affects the object Person. The different building
blocks of an OPD and their visual representation will be discussed in detail in the forthcoming sections.

Figure 2-1 OPD showing how marrying changes the state of a person from single to married

2.3 Object-Process Language

The Object-Process Language is the textual counterpart of the OPD. It is an automatically generated textual description of the system using a subset of natural English. A benefit from using natural language is that the OPL sentences are understandable to people with no previous programming experience. The OPD and OPL contain exactly the same information. From an OPD you can create an OPL paragraph and vice versa.

The Object-Process Language (OPL) serves two goals. One is to convert the OPDs into a natural language text that can be used to express analysis and design results among different stakeholders of a project. It is likely that executives and domain experts on the customer side would prefer reading a text to examining and interpreting OPDs. The second goal of OPL is to provide an infrastructure from which the modeled system can be implemented. The generated OPL text could be used to generate executable code or database schema.

Figure 2-2 shows the OPL sentences that correspond to the OPD in Figure 2-2. The bold words, such as Person, are non-reserved, whereas the non-bold words are reserved words that are part of the sentence structure, and glue the reserved words together.

OPL Paragraph:
Person can be single or married.
Marrying changes Person from single to married.

Figure 2-2 Example of an OPL Paragraph
2.4 Objects and Processes

Objects and processes are in OPM defined as things. Objects exist as static things while processes occur as dynamic things. Objects are relatively stable while processes are transient. Objects cannot be transformed (affected) without processes.

In OPM an object is defined as a thing that has the potential of stable, unconditional physical or mental existence. It is represented by a square with the objects name recorded inside. As can be seen in Figure 2-3, an object can be physical or informatical. A physical object consists of matter. In a broad sense these objects are tangible and can be discovered directly by observation. An informatical object is a piece of informatics. Examples of informatics can be a database, a recipe, an essay, a rule, a password or a file. Shading (or a double frame) is used to distinguish physical objects from informatical ones.

A process in OPM is defined as a pattern of transformations that an object undergoes. Processes can transform objects in three different ways. These are by consuming, affecting or creating them. A process is represented by an ellipse with the name of the process recorded inside. The name of the process should end with “ing” (marrying, manufacturing) so that the compiler understands that it is a process. If the name of the process does not end with “ing”, the word “Process” has to be added in the OPL. For example if you want to name a process “Marriage”, it will appear as “Marriage Process” in the OPL. It works the other way around for objects; they cannot end with “ing”. If you for example want to create an object called “Ceiling” it will be described as “Ceiling Object” in OPL.

In Figure 2-4 and Figure 2-5 the manufacturing process creates (or yields) a product while consuming raw material. The links from the operator and the machine are procedural links that will be explained later on.
2.5 States

OPM defines a state as a situation or a position at which an object can exist for a period of time. A change of an object is an alteration of the state of that object. Examples of states abound. Some examples are: Two states a person can be in are awake or asleep. The weather can be sunny, cloudy, rainy or snowy. A TV can be turned on or off. The states are symbolized by rounded rectangles inside the object box, with their names recorded inside them. A process can change the state of an object. An input state is the state the object has before the occurrence of a process, and an output state is the state it has after. The effect a process has on an object is the change in state that the object undergoes as a result of the process occurrence.
States can be suppressed or expressed. Suppression of states is a way of making the OPDs less complicated. The clarity of the OPD is increased by suppression of states. Expression of states gives more detail to an OPD and shows more exactly what a process does to an object. Figure 2-6 shows three figures describing the same thing. The first figure has its states expressed and the two other have their states suppressed. The middle one is an interim representation which is not valid in OPM.

![Figure 2-6](image)

The first figure shows an OPD with the states expressed. In the second figure the states are suppressed and the links move from the states to the contour of the Person object box. In the last figure the input and output links are superimposed to yield the effect link.

States can be initial or final. An example of this can be seen in Figure 2-7, where baby is the initial state and elder is the final state of a person. Initial and final states are useful for expressing objects that exhibit lifecycles.

![Figure 2-7](image)

2.6 Structural links

Structural links serves to connect objects to each other. OPM defines the structural link as the graphic symbol that represents a binary structural relation between two objects in an OPD. An arrow with an open head symbolizes a structural link. Along the link a label can be recorded. A structural link with a recorded label is also known as a tagged structural link. A structural link can be unidirectional or bidirectional. A tagged bidirectional link can have one or two labels. The different types of structural links are demonstrated in Figure 2-8.
If a label is not recorded a default OPL phrase will be applied to describe the relationship. For the unidirectional link the phrase is "relates to" and for the bidirectional link it is "are equivalent". An example of unlabeled tags in shown below in Figure 2-9.

Figure 2-9 OPD and OPL for structural relationships with null-tags.

### 2.6.1 Participation Constraints

In the examples above all the links have expressed relations from one object to another. However, in general we may wish to specify a certain number, or a range of numbers, of instances that participate in the relation. Adding a participation constraint does exactly this. The participation constraint is a number or a symbol recorded along the structural link next to an object, which denotes the number of instances of the object that participates in the link. It can be specified as a number, as a symbol or as a range whose limits are numbers and/or symbols. Some different types of participation constraints are demonstrated in Figure 2-10. There are five different symbols for participation constraints in OPM. These are: ?, *, 1, + and m which stand for 0..1, 0..m, 1..1, 1..m and many, respectively.

Figure 2-10 Structural links with participation constraints
2.7 Four fundamental structural relations

Four structural relations are most prevalent. A different OPD symbol has been assigned to each of these fundamental relations. The four relations are:

1) Aggregation-Participation, which denotes the relation between the whole and its parts.
2) Exhibition-Characterization, which denotes the relation between a thing and its features.
3) Generalization-Specialization, which denotes the relation between a general thing and its specifications.
4) Classification-Instantiation, which denotes the relation between a class of a thing and the instances from that class.

2.7.1 Aggregation-Participation

Aggregation-Participation is used to describe the parts the whole is made of. In OPDs the symbol for this relation is a black triangle. The corresponding OPL phrase is “consists of”. Two or more simultaneous relations are represented by a fork in the OPD. An example of how the aggregation symbol can be used is shown in Figure 2-11.

2.7.2 Exhibition-Characterization

The Exhibition-Characterization relation is used to show how a thing can be characterized by another thing. A thing describing another thing is known as a feature. An example of an Exhibition-Characterization relation is how name and address, that are objects in themselves as well, can be used to describe a person. The symbol for the Exhibition-Characterization relation is how name and address, that are objects in themselves as well, can be used to describe a person. The symbol for the Exhibition-Characterization relation is a white triangle with a smaller black triangle inside it. The OPL phrase used to describe this relation is “exhibits”. Figure 2-12 demonstrates how the exhibition-characterization symbol can be used.
The Exhibition-Characterization relation do not have to be objects; they can be processes as well. Figure 2-13 shows some examples of this.

The Exhibition-Characterization relation is also useful when an object has different types of states. Figure 2-14 shows how you should and how you not should (crossed over) model this.
2.7.3 Generalization-Specialization

When we want to distinguish a special group from a general one the Generalization-Specialization relationship is used. Two examples of this relation are how carrot, cucumber and tomato are specializations of the general term vegetable, and how man and woman are special cases of the more general person. In OPDs the relation is symbolized by a white triangle. In OPL the phrase “is a” describes the same thing. In Figure 2-15 Dog, Cat and Fish are specializations of the more general Animal. Shark and Salmon are special cases of the more general term Fish.

The relationship is transitive, which means that if A is a B, and B is a C, then it automatically follows that also A is a C. In the example above this would mean that since Shark is a Fish and Fish is an Animal, it follows that Shark is an Animal. The Generalization-Specialization relationship is normally used to describe connections between objects, but as seen in Figure 2-16 it can also represent relations between processes.
The most prominent benefit gained from using the relationship is the inheritance it induces. In OPM all the features, relations, states and processes used to describe a general thing also describe the special thing.

Figure 2-17 is an OPD of a Camera. Since Analog Camera and Digital Camera are Cameras a Manufacturer and an Image Capturing Process characterize them both, and they both use a Capturing Medium.

2.7.4 Classification-Instantiation

An instance is an incarnation of a particular identifiable member of a class. Instances are in OPDs represented by a white triangle with a black circle (or a black upside-down triangle) inside it. Examples of instances are; “Bach, Beethoven and Brahms are instances of Composers” or “Britney Spears and Christina Aguilera are instances of Pop Singers”. The OPL phrase for instantiation is “is instance of”. Figure 2-18 shows how the films The Godfather, Gone With the Wind and The Matrix are instances of Movie.
Inheritance of features from a class to its instances is exactly the same as inheritance from a super-class to a sub-class in the Generalization-Specialization hierarchy. The only difference is that an instantiation cannot have further specializations in this system, since it is at the bottom of the hierarchy. What is an instance in one system can be a class in another. For example in a national highway system designers may wish to categorize vehicles into three types: cars, busses and trucks. Car would here be an instance of the class vehicle. Now consider a car dealership. Here car would be a class and each individual car would be an instance. This is the “classical” case of instance where each instance is a physical object with its unique identifier.

2.8 Procedural links

Procedural links are typically used to connect objects with processes. There are two main types of procedural links: enabling links and transformation links. The enabling links show which objects need to be present in order for a process to occur. The transformation links are used to show how processes create, affect or consume objects. An invocation link is a direct link between two processes, which is used to denote that one process starts after another one has ended.

2.8.1 Enabling Links

An enabler of a process is an object that enables the process execution but is not consumed by it. The states of the object are the same after the process has completed as they were before it started. An example of an enabler can be a Tennis racket. In order for the process Tennis Playing to occur the racket needs to be present. When the process has ended you still have the racket.

There are two types of enabling links: Agent links and Instrument links. An agent is an intelligent enabler, which can control the process by exercising common sense or goal-oriented considerations. Usually it is a single person. In OPDs the agent link is represented by a line, drawn from the object to the process, which ends with a filled black circle at the periphery of the ellipse representing the process. The corresponding OPL phrase is “handles”.

---

Figure 2-18 The Godfather, Gone With the Wind and The Matrix are instances of Movie.
Instruments are non-human physical or informatical enablers. Machines, tools, robots, controllers, hardware, software, documents and information are examples of instruments. In practice physical instruments usually change because they wear out. However from the systems point of view these changes are often not significant enough to be accounted for. In OPDs the instrument link is represented by a line drawn from the object to the process, ending with a circle touching the periphery of the process. The OPL phrase for representing an instrumental link is “requires”. Figure 2-19 shows examples of both agent links and instrumental links.

2.8.2 Transformation Links

The transformation links are used to show how processes can:
- Create or generate new objects
- Affect objects by changing their states
- Consume or destroy existing objects

Figure 2-20 shows the three types of transformation links. The links are represented by unidirectional or bidirectional arrows.
2.8.3 Invoking Links

By definition any process must transform an object. Sometimes the transformation is not significant and may as well be ignored, but the change in the implicit object invokes another process. The purpose of the invocation link is to model how a process invokes another process without having to model the transformation of the object that the invoking process has brought about. The invocation link can be thought of as an event that marks the end of the process from which it emanates, and at the same time, invokes the process to which it points. Figure 2-21 (a) shows how Process A invokes Process B. In Figure 2-21 (b) the “complete version” is shown, where Process A yields an Interim Object that is immediately consumed by Process B. The invocation link in (a) is just a more convenient way of expressing the same thing that is stated in (b). The invocation link replaces the interim object for the sake of brevity. As can be seen in the figure below, an invoking link is represented by a lightning arrow in OPDs and by the phrase “invokes” in OPL.

Figure 2-21 The same system is modeled in (a) and (b). In (a) with the invocation link and in (b) with an interim object.
2.9 Managing Systems’ Complexity

Complexity management is a tradeoff between two conflicting requirements: completeness and clarity. On one hand, completeness requires that the system should be specified to the fullest extent possible. On the other hand, the need for clarity imposes an upper limit of complexity of a diagram, and does not allow it to be too cluttered or loaded. Object oriented development methods, notably the UML standard addresses the problem of managing systems’ complexity by creating a model for each one of the important aspects of the system – structure, dynamics, state transition, etc.

OPM handles systems’ complexity by introducing a number of abstracting-refining mechanisms. These enable presenting and viewing the system at various levels of detail. Abstracting saves space and reduces complexity, but it comes at the price of completeness. Refining expresses the system in more detailed way at the price of loss of clarity. In OPM the different aspects of the system are presented in one single model. Scaling is the name of the process which can be either abstracting or refining. There are three types of scaling methods: unfolding/folding, in-zooming/out-zooming and state expression/suppression. The state expression/suppression was described in chapter 2.5. The two remaining methods will be presented now.

2.9.1 Unfolding and Folding

Unfolding operates on a thing by revealing a set of lower-level things. The result of unfolding is a tree, the root of which is the unfolded thing. Folding is the opposite of unfolding, and works by hiding the structural relations of a thing.

If we for example would like to describe one of the objects in Figure 2-19 in more detail, but we think that the diagram would become too cluttered if we just added the information into the same diagram, we could unfold the object we want to describe into a new diagram. Figure 2-22 shows how the object House, from diagram SD (in Figure 2-19), is unfolded and described in further detail in diagram SD1. The border around the House object in Figure 2-19 is thicker than the borders around the other objects. This is to symbolize that the House object is unfolded in another diagram.

![Diagram SD1](image)

**Figure 2-22** Example of unfolding. The structural relations, of the House object in diagram SD (Figure 2-19), are revealed in diagram SD1.
2.9.2 In-zooming and Out-zooming

Lower-level things enclosed inside a thing are made visible by in-zooming. Conversely zooming out of a thing makes the things inside it invisible. In-zooming is by default applied to processes, but it can also be applied to objects. In-zooming is useful when you for instance want to describe how a process works in a more detailed manner. Again we can use the building process in Figure 2-19 to show how this works. By zooming into the building process in diagram SD you get diagram SD2, seen in Figure 2-23. The **Building** process in Figure 2-19 has a thicker border than normal processes have, to denote that it can be zoomed into.

![Diagram SD2](image)

**Figure 2-23** The **Building** process from Figure 2-19 is described in further detail through in-zooming.
Chapter 3. An Example of OPM Modeling
The Alarm Mediation Device Example

During a course in Model Driven Architecture (MDA), taught at Ericsson ETH/R&D, the task is to make an xUML model from a requirement specification. The same specification is used here to make an OPM-model of the requirements.

3.1 Introduction

Model driven architecture is used in some projects at Ericsson ETH/R&D. A course has been developed to teach staff the basic concepts of this type of modeling. As a part of this course the assignment to make a model from a certain requirement specification is given. The models are to be created by using the iUML tool, from Kennedy Carter. The tool uses the xUML (eXecutable UML) process, which is a subset of the normal UML standard with support by the action semantics-compliant Action Specification Language (ASL).

As a minor part of the master project the requirement specification used in the course was modeled using the OPM paradigm. The purpose was to see if some of the UML code could be automatically generated from the OPM model, using the UML generation function in the OPCAT tool (the tool for creating models in OPM). The produced model will be used here to show how requirements can be modeled through the OPM paradigm.

3.2 Requirements

The requirements can be found in Requirement Specification of Alarm Mediation Device R1 (see below). Not all parts of the requirements of the specification are used in the model. Only those elements that were modeled in xUML during the MDA course are accounted for in the OPM model.

3.2.1 Requirement Specification of Alarm Mediation Device R1

Alarm Mediation Device (AMD) is a fault management application. AMD is responsible for informing the network operator about abnormal conditions that affect the operation of a network and its environment. Main task of AMD is to manage alarms reported by network elements and to display them to the operator. Alarms are managed according to the following requirement specification.
If an alarm condition occurs in one of the network elements, a signal is sent by the node to AMD. The alarm signal contains a unique alarm identifier, the IP address of the sender and the number of the problem cause. AMD server appends the new alarm to its active alarm list and notifies the subscribed clients via a user-interface about the change of the size of the alarm list, and backs up the new alarm in log files.

Active alarm list shall be backed into an SQL database.

Clients can poll the active alarm list that is displayed on the clients’ graphical user-interface. Network operator shall see the following information of an alarm: a unique alarm identifier, the IP address of the alarming resource, the problem cause, the raise time, the raise date, the alarm status (e.g. new, acknowledged, cleared), the acknowledging client (ID) and the ceasing client (ID) or network element.

When the operator becomes aware of the situation, he/she acknowledges the alarm via the user-interface (indicating that it is taken into account).

When the alarming condition is ceased the node informs AMD. The cease signal contains the unique alarm identifier. An alarm can be ceased by that node where it raised.

Once the alarm is acknowledged and ceased, it is removed from the active alarm list and backed up in log files and notifies the subscribed clients via the user-interface about the change of the size of the alarm list.

The network operator can cease an alarm manually.

AMD server loads the list of active network elements from the topology database at startup (TOP).

Signals received from nodes that are not listed in TOP shall be discarded.

### 3.3 The Alarm Mediation Device Model

The OPM-model of the alarm mediation device consists of six Object-Process Diagrams and their corresponding Object-Process Language paragraphs. For the sake of brevity not all OPDs and OPLs will be shown here. The entire model can be downloaded from http://www.student.lu.se/~jur98psv/amd.opz. The OPCAT-tool, which is required for viewing the model, can be downloaded from http://dori.technion.ac.il/opm.

Figure 3-1 shows the OPD “SD”. The diagram contains the most significant objects and processes found in the requirement specification. The corresponding OPL paragraphs are found in Figure 3-4.
3.3.1 Objects

The objects modeled based on the requirement specification are: Network Element, Physical Network Element, Network Operator, Active Alarm, Active Alarm List, SQL Database and Log File.

In the requirement specification only one type of network element is mentioned, but during modeling it becomes apparent that you need to make a clear distinction between the Physical Network Element (which sends the signals to create or cease an alarm) and its data representation, Network Element. The Physical Network Element has the attribute PNE IP address which corresponds with the NE IP address of Network Element. Further, the object Network Element has a unique identifier, NE ID which is stored in every created alarm to be able to see which Physical Network Alarm has generated it.

Each Network Operator has been assigned a unique identifier, NwOp ID. Also each Network Operator possesses a GUI on which, for example, changes in the Active Alarm List can be displayed.

In Figure 3-2 the Active Alarm object is unfolded into the OPD “SD1 - Active Alarm unfolded”. The Active Alarm consists of a Unique Alarm Identifier, Senders IP
address, Problem Cause, Raise Time, Raise Date, Acknowledging Client ID and Ceasing Client ID. These attributes are drawn from requirement R3 in the specification. Further the active alarm has two types of states, Acknowledged and Ceased, with values yes and no.

![Figure 3-2 SD1 - Active Alarm unfolded.](image)

When looking at the OPL sentences of some of the objects they are sometimes very similar to the original requirements in the specification. A good example of this is how the requirement R2, which states that “Active alarm list shall be backed into an SQL database”, is matched by the OPL sentence “Active Alarm List is backed up in SQL Database” (see Figure 3-4).

### 3.3.2 Processes

The process Initializing originates from requirement R8 in the specification. The requirement states that “AMD server loads the list of active network elements from the topology database at startup (TOP)”. In the OPD you see how the data representation of the network elements is created by the Initializing process. Only Network Elements stored in the Topology Database are created.

The Alarm Creating process is described in requirement R1 and R9. In the OPD it is shown how an Active Alarm is created. The process is started by a Physical Network Element and requires data from its informatical representation: Network Element.
Figure 3-3, “SD5 - Alarm Creating in-zoomed” shows how an **Active Alarm** is created, backed up in a **Log File**, appended to the **Active Alarm List** and how the **Network Operator** is informed about the changes in the **Active Alarm List**.

![Figure 3-3 SD5 - Alarm Creating in-zoomed](image)

The **Alarm Ceasing** process is modeled from requirements R5 and R7. It shows how an **Active Alarm** can be ceased by the **Network Operator** or a **Physical Network Element**. “SD3 - Alarm Ceasing unfolded” shows how the state **Ceased** of the alarm is changed from **no** to **yes**, and how the **Ceasing Client ID** is added to the **Active Alarm**.

The **Alarm Acknowledging** process is described in requirement R4. It illustrates how the **Active Alarm** is acknowledged by the **Network Operator**. How the state **Acknowledged** is altered from **no** to **yes**, and how the **Acknowledging Client ID** is set for the **Active Alarm** is shown in the OPD “SD2 - Alarm Acknowledging unfolded”.

The **Alarm Clearing** process corresponds to requirement R5. “SD4 - Alarm Clearing in-zoomed” shows that an alarm will be deleted and removed from the **Active Alarm List** once it has been acknowledged and ceased. The deleted **Active Alarm** is backed up in a **Log File** and the **Network Operator** is notified.
The **Polling** process is modeled from requirement R3, which states the information a **Network Operator** shall be able to see for an **Active Alarm**.

---

**OPL Paragraph for SD:**

**Physical Network Element** is physical.

**Physical Network Element** exhibits **PNE IP Address**.

**Physical Network Element** handles **Alarm Creating**.

Following path b, **Physical Network Element** handles **Alarm Ceasing**.

**Network Element** exhibits **NE IP Address** and **NE ID**.

**NE IP Address** and **PNE IP Address** are corresponding.

Many **Network Elements** are represented in **Topology Database**.

**Network Operator** is physical.

**Network Operator** exhibits **NwOp ID**.

**Network Operator** consists of **GUI**.

**Network Operator** handles **Alarm Acknowledging** or **Polling**.

**Network Operator** handles **Initializing**.

Following path a, **Network Operator** handles **Alarm Ceasing**.

**Active Alarm List** consists of optional **Active Alarms**.

**Active Alarm** exhibits **Ceased** and **Acknowledged**.

**Ceased** can be **no** or **yes**.

**no** is initial.

**Ceased** triggers **Alarm Clearing** when it enters **yes**.

**Acknowledged** can be **no** or **yes**.

**no** is initial.

**Acknowledged** triggers **Alarm Clearing** when it enters **yes**.

**Active Alarm List** is backed up in **SQL Database**.

**SQL Database** stores **Active Alarm List**.

**Alarm Creating** requires **Network Element**.

**Alarm Creating** yields **Active Alarm**.

**Alarm Ceasing** affects **Active Alarm**.

**Alarm Acknowledging** affects **Active Alarm**.

**Alarm Clearing** requires **yes Acknowledged** and **yes Ceased**.

**Alarm Clearing** consumes **Active Alarm**.

**Initializing** requires **Topology Database**.

**Initializing** yields **Network Element**.

**Polling** requires **Active Alarm List**.

**Polling** affects **GUI**.

---

Figure 3-4 OPL paragraph for "SD"
3.4 Results

As mentioned in the beginning of this chapter the main purpose of making the model of the Alarm Mediation Device was to examine if the OPM-model could be transformed into UML. In the OPCAT tool there is a function for automatic generation of XMI files. XMI, short for XML Metadata Interchange, is intended to provide a standard way for programmers and other users to exchange information about metadata (essentially, information about what a set of data consists of and how it is organized). Specifically, XMI is intended to help programmers using the Unified Modeling Language (UML) with different languages and development tools to exchange their data models with each other. [5]

The created XMI file was imported into Rational Rose (a tool for creating UML models). The resulting UML information consisted of the following items:

- A class diagram, connecting the different objects from the OPM-model.
- A separate class for each of the objects.
- Two activity diagrams. One for the Alarm Clearing process and one for the Alarm Creating process. The one for Alarm Clearing is shown in Figure 3-5.
- Some associations between the classes created from the OPM-object.

![Figure 3-5 Activity diagram for the process Alarm Clearing](image)

---

[5]
Chapter 4. Method

The first phase of the master’s project was the modeling of the software measurement processes at Ericsson’s R&D department. This phase began with a literature study, followed by the modeling of the software measurement processes in OPM, using OPCAT. The second phase of the project was the creation of a tool for presenting information from measurements of the software process. The final phase of the project was the validation.

4.1 Research Questions

There are four main research questions this thesis is trying to answer.

- What would an OPM-model of the software measurement process at Ericsson ETH/R&D look like? The documentation of the collection of metrics today is in textual format. How should these documents be transformed into a model? An example of how this question could be answered is found in chapter six.
- What should a tool for presenting volume and effort data from the projects at Ericsson ETH/R&D look like? A proposal for what such a tool could look like is found in chapter eight where a prototype of such a tool is presented.
- Is the Object-Process Methodology suited for modeling the software measurement processes at Ericsson ETH/R&D? What are the benefits and drawbacks of using this methodology? This is discussed in chapter nine.
- How useful is the developed prototype for presenting data from the software measurement process? An evaluation is conducted in chapter nine.

4.2 Literature Study

The project began with a one and a half week long literature study. Particular attention was given to the book Object-Process Methodology – A Holistic Systems Paradigm [3] which yet is the only published book on the subject. The book summarizes the basic concepts of the OPM paradigm. The methodology is described through extensive examples.

In the article OPM/Web – Object-Process Methodology for Developing Web Applications [6] the OPM is developed further. The main extensions of OPM/Web are adding properties of links to express requirements, extending the zooming and unfolding facilities and adding global data integrity. Since the CASE-tool for modeling OPM systems, OPCAT, does not support OPM/Web non of these features were used for modeling the software measurement process at Ericsson ETH.
OPM/T is an extension of OPM that adds features such as timing constraints, events, conditions, exceptions and control flow constructs. It is described in *Extending the Object-Process Methodology to Handle Real Time Systems* [7].

In order to be able to understand the different aspects of software engineering the book *Software Engineering* [8] was read. This book gives a general description of the processes normally involved in the creation of software.

### 4.3 Working Procedure

The thesis was written at Ericsson ETH in Hungary. The required information was gathered through internal documents, literature studies and interviews with staff members. The developed model is a representation of internal documents describing the ETH’s software measurement process. Most of the internal documents can be found in the appendix. The creation of the model was conducted during a period of a month. The model was created by going through the internal documents one at a time and incorporating them into the OPM model. Minor reviews of the model were held continuously with my supervisor at the company, with an interval of about two days. Major reviews were held every two weeks. At these occasions a line manager and a part time employed Ph D. student with software measurement expertise also looked though the model and voiced their comments. A line manager is a manager who is responsible for staff with a certain type of expertise. When project managers need people for their projects they contact the appropriate line managers and request personnel. Parts of the model contain information that cannot be found in the internal documents. This information was obtained through interviews with persons with good knowledge of the Ericsson ETH organization, mainly my supervisor and the Ph D. student.

When all the documents had been considered the model was put into a baseline, and the implementation phase of the project could begin. The implementation of the model is a MS Access database which contains data regarding effort and volume and various forms for presenting this information. This phase of the project began with getting to know the software, through various online tutorials and the book *Access 2002 Bible* [2]. The creation of this prototype tool took about one month.

The final phase of the project was the validation. It consists of two parts, the validation of the model and the validation of the implementation. The validity of the OPM-model is based on the reviews held at the company. The validity of the implemented tool for presentation of software measurements was conducted through the creation of various scenarios. Since no survey could be carried out with the intended user-group the validity of this part is fairly low.
Chapter 5. Measurement Program at ETH/R&D

The measurement system at ETH/R&D focuses on three different types of measurements. These are effort, code-size and quality. Effort is derived from the Time Reporting Tool (TRT) which is a collection of man-hours spent on different projects. The size of the code is automatically generated by a computer program called Softstat. Quality is measured through trouble reports. The measurement system is described in extensive internal documentation.

5.1 Requirements

The model of the measurement system is mainly created from internal documents. A concise description of the documentation used for creating the OPM-model is presented now.

- **Time Reporting at ETH/R** [int_1]
  *Prepared by Kinga Légrády, No. ETH/R-2002:0008 Uen*
  This document specifies the basic time reporting principles used at ETH/R. It states how the Time Reporting Tool (TRT) works. Responsibilities regarding the reporting and managing the TRT system is also described for assistants, line managers, resource and project controllers, project managers and other staff members.

- **Software Productivity Measurement User’s Guide (Prototype version)** [int_2]
  *Prepared by Zoltán Theisz, No ETH/R-04:000008 Uen*
  The document gives a short description of the current toolset available at ETH/R&D. It contains a detailed description of the level structure of the Time Reporting Tool. Further it defines which artifacts (e.g. writing specifications, documentation or coding) are connected to which disciplines (e.g. function test, design or implementation) for different types of projects. It also describes how the collected data should be presented in different types of charts.

- **Proposal for New TRT Structure** [int_3]
  *Prepared by Zoltán Theisz, ETH/R-04:000009 Uen*
  The document describes what the four different levels of the TRT structure shall contain. It further specifies which stakeholders should collect data from which TRT-levels. In addition, it contains some matrixes showing the relationships between artifacts and disciplines, just like in [int_2].
• **Measurement System at ETH/R** [int_4]
  *Prepared by István Kiszel, No. ETH/R-2000:0089*
  The quality and process measurement systems used within the Research and Development Division, ETH/R, are described in this document. Since this document was written in 2000 it does not give a comprehensive view of the measurement systems.

• **Softstat 2RC** [int_5]
  *Ericsson PowerPoint presentation, 038 13 CAA 139 1257*
  Softstat is a tool which measures the volume and modification grade of software products, and/or collects statistics for the products. The presentation explains how the Softstat software is used to calculate the number of new, unmodified and deleted lines of code for a project.

• **Proposal for CROSBY cost model calculation** [int_6]
  With the new TRT structure the need arises for how to map the time reporting data to the cost modeling factors. This document aims to propose a solution to this. It describes four different kinds of costs, performance costs, non-conformance costs, prevention costs and appraisal costs, and suggests which disciplines should be connected to which type of cost.

• **Project Quality and Method Coordinator** [int_7]
  *Prepared by István Kiszel, No. ETH/R-2002:0105 Uen*
  This document describes the responsibilities and work tasks of the Project Quality and Method Coordinator. It further states the authorities and competence requirements of the Project Quality and Method Coordinator.

• **Main Project Manager** [int_8]
  *Prepared by Antal Bartus, No. ETH/R-03:000025 Uen*
  This document describes the responsibilities and work tasks of the Main Project Manager. It further states the authorities and competence requirements of the Main Project Manager.

• **Product Quality Board (PQB) Template** [int_9]
  *Prepared by Antal Bartus, PowerPoint Presentation, 2003*
  The presentation shows the types of charts and tables that are relevant for measuring quality. Important information is Fault Density, Open Trouble Reports (TRs), External TR Answering performance and Received Customer Service Requests (CSRs).
5.2 Interviews

In addition to the information found in the internal documentation the model contains information gathered from interviews with staff at Ericsson ETH/R&D. The information was gathered from discussions primarily with two employees.

- **Zoltán Theisz**
  Mr. Theisz is a full time employee at Ericsson and is involved with software development. Questions regarding for example the organization and measurement tools used within the R&D division were answered by him.

- **Gábor Stikkel**
  Mr. Stikkel is a Ph.D. student and part time employee at Ericsson ETH/R&D. He has specialized in software measurements. Questions regarding for example which data should be modeled, and which information might be of use for managers were answered by him.
Chapter 6. OPM-model of the Software Measurement Process

The OPM-model contains the most important elements of the software measurement process at Ericsson ETH. In this chapter the model is discussed and explained.

6.1 Introduction

The software measurement process at Ericsson ETH/R&D is measured using mainly three types of data. These are volume, effort and quality data. The volume metrics comes from automated measurements of the number of lines of source code in different projects. The effort metrics are collected using web-based tools, such as the “Time Reporting Tool”. Each employee has to report the number of hours spent working on different disciplines in different projects. The quality metrics are derived from the trouble reports. How the different metrics are collected and organized is described in several internal documents. These documents have been converted into an OPM-model which will be displayed in this chapter.

The developed model is intended to serve as a tool, aiding the understanding of different elements involved in the software measurement process at Ericsson ETH. Parts of the model have been implemented as a program which presents metrics collected in the organization (presented in chapter eight). Since the model is quite extensive only the most central parts will be presented in this thesis. The complete model can be downloaded from http://www.student.lu.se/~jur98psv/msmp.zip. To be able to view the model you will need to download OPCAT from http://dori.technion.ac.il/opm. The model is described in a top-down manner, starting from the top level and working our way down through the different sub-diagrams.

6.2 OPCAT/OPM Inconsistencies

When making the model some inconsistencies were found between the current version of OPCAT and the Object-Process Methodology Paradigm. The most inconvenient of these had to do with states. For example, when modeling the deadlines the original plan was to create something similar to the OPD shown in Figure 2-1. In the current version of OPCAT a design like this is not possible. When trying to connect the deadlines with the different states of Day and Time you get an error message saying that “An object and a state cannot be connected with a featuring-characterization relation”.

It turned out that in OPCAT states and objects cannot be connected with any kind of structural relations, although examples of this in the literature are extensive. (See for example chapter 12, States and Values in Object-Process Methodology [3].) Since the feature of connecting states and objects does not exist in OPCAT the OPM diagrams had to be adapted to the tool. The original design idea of the deadlines structure from figure 6-1 was instead modeled without states, as can be seen in Figure 6-2.

Two other features that are described in the literature which lack support in OPCAT are distributive forks and non-comprehensiveness. Distributive forks are used to express “AND-statements”. (In the examples above the deadlines consists of a day AND a time.) For example an OPL sentence from figure 6-1 should be able to be written like this: **PM Approving Deadline** exhibits **Day** the value of which is **Tue** and **Time** the value of which is 14:00. Since the feature of distributive forks is not included in OPCAT you instead have to write two separate sentences: **PM Approving Deadline** exhibits **Day** the value of which is **Tue**. **PM Approving Deadline** exhibits **Time** the value of which is...
14:00. (p.120, Object-Process Methodology [3].) In the OPD’s however the forks can be displayed properly. Non-comprehensiveness is used to express that an object consists of not only the parts accounted for in the OPD, but also additional ones. (p.142, Object-Process Methodology [3].)

Yet another function that has not been properly implemented in OPCAT is the OR/XOR statements. In the OPDs the XOR (Xorclusive OR) link is properly denoted by a dotted line, as seen in Figure 6-3. When looking at the OPL-sentences there are no sentences containing any kind of OR-statements.

![Figure 6-3 XOR links](image)

6.3 The OPM Model

The creation of the OPM-model was the first large task of the master’s project. The model contains more than seventy Object-Process Diagrams, so for the sake of brevity not all diagrams are shown. The different diagrams will be explained starting at the top level.

6.3.1 SD - Top Level System Diagram

Figure 6-4 show the first OPD of the model. It shows four processes that are important parts of the software measurement processes at Ericsson ETH/R&D, what tools or files are required for these processes to occur and who is responsible for them. After reading through the internal documents it was decided that four different processes were to be presented at the top level. The processes **Reporting**, **Approving** and **Managing** derive from the document *Time Reporting at ETH/R* [int_1]. The process **Analyzing** originates from the documents *Software Productivity Measurement User’s Guide* [int_2] and *Proposal for New TRT Structure*. As more documents were read and incorporated into the model these were mainly integrated in the **Analyzing** part of the model. How the OPDs are connected to the internal documentation will be explained in the sub-diagrams.

Different stakeholders have been added to the OPD to show who is responsible for what process. For example the agent link from Employee to Reporting correlates to the sentence “Each Member has to submit his/her time report using the Time Reporting Tool on a weekly basis” from paragraph 5.1 ETH/R Staff in *Time Reporting at ETH/R*. 
The processes create or affect the objects **TRT Weekly Report** and **TRT Structure**. Where these objects come from is explained in their respective OPDs.

![Figure 6-4 SD - Top Level System Diagram](image)

### 6.3.2 Reporting in-zoomed (SD 6)

This OPD illustrates how the reporting process works. It shows among other things how employees store the data regarding how much time they have spent working in a tool called the WTR-tool (Working Time Report) and how this information later is transferred to the TRT (Time Reporting Tool) from which a time-report is put together every week. The OPD also contains special cases for what happens if an employee is unable to fill out the time-report due to for instance illness.

The design of this OPD is correlated to paragraph 5.1 and 5.2 in the document Time Reporting at ETH/R [int_1]. Paragraph 5.1 states that “Each member has to submit his/her time report using the Time Reporting Tool on a weekly basis. They have to save the data in the Working Time Report Tool, so that the number of working hours can be transferred to the Time Reporting Tool…” As can be seen in the OPD in Figure 6-5 an **Employee** handles the process **Saving Data in WTR Tool** which affects the **WTR Tool**. The **WTR Tool** transfers data to the **TRT Tool**.

We can further read that “… In case of vacation or short business trips, training, a mail has to be sent to the Assistant. This mail has to include all necessary information for their reports… In case of illness, the person has to phone the Assistant and let her know about all the information described in above…” This correlates to the process **Notifying Assistant**, handled by the **Employee**. The process requires that the **Employee** is **not in office**.

Paragraph 5.2 in [int_1], states that “Assistant is responsible for sending a reminder mail to project members on first working day of each week, saving the hours in the Working Time Report for those who are not in the office, forwarding the mails to or informing the Resource/Project Controller so that she can prepare the time report for the people not present.” This paragraph is represented by the processes **Reminding Via E-mail** and **Forwarding Mail To Controller** handled by the **Assistant**. Paths in the OPD marked...
with an a represents that the employee is not in office and paths marked with a b represents that the employee is in office.

Paragraph 5.1 also states that “... In case of any problem accessing the tool, the time report should be sent to the Project Manager and to the Resource/Project Controller via e-mail.” In the OPD this corresponds to the process **Send Time-report Via E-mail** which affects the **Project Manager** and the Resource/Project Controller and requires that the **TRT tool** is **offline**. The process is handled by the **Assistant** or the **Employee**. The process **Selecting TRT Structure** relates to paragraph 4 in Time Reporting at ETH/R. The process **Creating TRT Report** is not specifically described in any documentation, but intuitively there has to be a process that creates the report.

![Figure 6-5 SD 6 – Reporting in-zoomed](image)

### 6.3.3 Approving in-zoomed (SD 2)

The diagram shows the different steps a manager has to go through when approving the time-reports of his staff. It states in what has to be done and the deadlines for the different stages. The processes and objects in the OPD concerning the approval of the TRT-reports (Figure 6-6) are related to paragraph 5.3 and 5.4 in [int_1]. The process **Checking If Report Consists Of The Number Of Hours Stated in WTR Tool** is correlated to the sentence “Line managers are responsible for checking whether project members have reported exactly as many hours as the Working Time Report offers on a weekly basis. The deadline for this checking is 18:00 p.m. Monday” in paragraph 5.3.
The process **Approving Internal/Non-productive Activities** comes from the sentence “Line managers are responsible for approving the ‘Internal/Non-productive Activity’ hours on a weekly basis till 18:00 p.m. every Monday”, from the same paragraph. The process **Approving Project Related Activities** corresponds to “Project managers are responsible for approving the hours reported on their projects on a weekly basis...”.

The processes **Checking If All Lines In Report Are Approved** and **Approving TRT Report** are not represented in the documentation. However, interviews with staff showed that this would be an appropriate way of explaining the “approving” processes at the company. The process **Redoing TRT Item** is zoomed into in “SD 2.1 - Redoing TRT Item in-zoomed”. The OPD will not be shown here for the sake of brevity, but the contents of the diagram derive from the sentence “Line Managers are responsible for assuring that all the rejected reports of their people are corrected...” in paragraph 5.3 in [int_1].

---

**Figure 6-6 SD2 - Approving in-zoomed**

---

35
6.3.4 Managing in-zoomed (SD 5)

The OPD managing in-zoomed present some processes that are concerned with managing the TRT-system and who is responsible for these processes. The objects and processes shown in Figure 6-7 mainly correlate to paragraph 5.5 and paragraph 6 in [int_1]. The processes **Freezing, Supervising Time Reporting Process, Closing Time Report Each Month Checking Deadlines & Quality, Initiate Invoicing, Sending Excel-File to Unit Director** and **Registering Hours in SAP** all relate to paragraph 5.5 which states that: Resource and Project Controller is responsible for

- Freezing the projects on weekly basis. The projects are to be frozen till 12:00 a.m. Wednesday
- Supervising the time reporting process.
- Closing the Time Reporting period each month.
- Initiating the invoicing procedure towards the orderer.
- Sending an Excel file containing the hours derived from the Time Reporting Tool to unit directors when invoicing is due.
- Registering the reported hours in SAP for invoicing and monthly closing.
- Checking whether the deadlines are kept, the quality of the time reports is high.

The process **Adding New Project** handled by **ROF** relates to the sentence “**ROF is responsible for adding new projects to the Project List of the Time Reporting Tool…**” in paragraph 6 [int_1]. “**The Project Managers and the Resource/Project Controller are responsible for defining, changing and deleting the project/activity structure...**” is the sentence from which the process **Defining, Changing, Deleting Project/Activity Structure** originates.
6.3.5 Analyzing unfolded (SD 1)

In this diagram processes are displayed which are related to the analyzing of measurement data. The Analyzing part of the model (Figure 6-8) is divided into five different processes. The Cost Modeling process contains information from the document Proposal for CROSBY cost model calculation [int_6]. The process Trouble Report Analysis gets its data from [int_9]. Volume And Effort Analyzing is mainly based on [int_2] and [int_3]. The process Softstat Processing originates from [int_5] and the process Report Creating was created through interviews with staff. Information about who is responsible for which process was obtained from interviews with Mr. Stikkel. The processes are described in more detail in the forthcoming sections.
6.3.6 Cost Modeling in-zoomed (SD 1.2)

The process Cost Modeling is yet to be completed. Since the CROSBY model concept is not yet introduced at ETH/R&D it is not decided how the process of cost modeling will be managed. However, the structure of the CROSBY cost model is already set, showing which types of costs are to be connected to which activities and disciplines. Since only the structure of CROSBY is currently known, only this part has been put into the process Cost Modeling. When the routines regarding CROSBY cost modeling are developed later, these can be easily added to this process. The structure of the cost model is found inside the object CROSBY Cost Model which is unfolded in SD 1.2.1 – CROSBY Cost Model Unfolded (Figure 6-9).

The object CROSBY Cost Model consists of four different types of costs. These are represented by the objects Prevention Cost, Performance Cost, Appraisal Cost and Non-conformance Cost. The different types of costs are presented in the “Crosby Model” section, and described in detail in the “Proposed mapping” section [int_6]. For each type of cost it is shown which types of activities and disciplines are defined. For example there is a relation between the objects Prevention Cost and CW / Man. Disc. It symbolizes that if the activity is “Create Work (CW)” and the discipline is a “Management Discipline” then the type of cost the work shall be related to is “Prevention Cost”. The information regarding which discipline and activity is connected to which type of cost arises from the table “Mapping new TRT structure for CROSBY cost model” in [int_6].
All the six objects which are connected to the different types of costs contain an activity and several disciplines. In paragraph 1.3.4.1 [int_3], the abbreviations of the activities are explained. The abbreviations of the disciplines are explained in paragraph 1.3.2.1 and 1.3.2.2 in the same document. The six objects containing the activities and disciplines are unfolded in SD 1.2.1.1 to 1.2.1.6. The OPM for the object CW / Man. Disc. is shown in Figure 6-10. The OPDs for the remaining five objects follow the same pattern.
6.3.7 Trouble Report Analysis unfolded (SD 1.6)

In Figure 6-11 the structure of a trouble report is displayed. The elements characterizing the trouble report originate from [int_9]. After studying the slides of the document the objects below were decided upon. The objects **FD Goal** and **Sub System** derive from the slide “Fault Density” in the PQB template presentation [int_9], and **Rank** originates for example from the slide “Open TRs”. The three different types of trouble reports: **TR_FT**, **TR_ST/NT** and **TR_60P**, originate from the slide “Fault Density”. The disciplines the different types of trouble reports are related to are: **FT** (Function Test), **ST** (System Test), and **DFU** (Design Follow Up). The information about which type of trouble report is related to which discipline was gathered through interviews.

![Figure 6-11 SD 1.6 - Trouble Report Analysis unfolded](image)

In SD 1.6.1 - Trouble Report Analysis unfolded (Figure 6-12) the different types of analysis that represent the Trouble Report Analysis are displayed. Each type of analysis is represented by a process. The information for each process is found in a slide with a corresponding name in [int_9].

Each process creates some kind of distribution and a chart displaying the created distribution. Since the designs of the processes are very similar only one of them – Examine Open TRs – is presented.
Figure 6-13 shows how a **Line chart** object is created displaying the information of an **Open TR Distribution** object. Figure 6-14 shows how the **Open TR Distribution** is created in more detail. These OPDs have been created to replicate the information found in the slide “Open TRs” in [int_9].

Figure 6-15, Open TRs Distribution unfolded, shows the structure of the object **Open TRs Distribution**. You can see how the data in the distribution is connected to the data in the trouble reports. The way the data will be presented in charts is also shown. The information in this diagram is the author’s idea of how the graph on the slide Open TRs could be created. The distribution is not mentioned in the documentation, but by creating it the construction of graphs found on the slides will be easier to comprehend.
Figure 6-13 SD 1.6.1.3 - Examine Open TRs in-zoomed

Figure 6-14 SD 1.6.1.3.1 Get Open TR Distribution in-zoomed
6.3.8 Volume And Effort Analyzing unfolded (SD 1.7)

The design of the OPD Volume And Effort Analyzing (Figure 6-16) unfolded is similar to the design of Trouble Report Analysis unfolded (SD 1.6.1) in section 6.3.7. The processes in the diagram originate from [int_2]. The process Discipline Distribution Analyzing comes from paragraph 2.5.1.1 in this document. Staffing Profile Analyzing is found in paragraph 2.5.1.2 and Discipline Evolution Analyzing originates from paragraph 2.5.1.3. Code Distribution Analyzing, Code Growth Profile Analyzing and Code Change Profile Analyzing originate from paragraphs 2.5.2.1 – 2.5.2.3. Finally the process Productivity Profile Analyzing is found in paragraph 2.5.3.

The design of the in-zoomed processes and unfolded objects in this diagram are very similar to the ones in section 6.3.7. The in-zoomed processes show approximately the same layout as can be seen in figure 6-13, and the design of the unfolded objects resemble the ones in figure 6-15. All information from these processes and objects has been gathered from the document [int_2] in combination with interviews with the staff.
6.3.9 Softstat Processing unfolded (SD 1.5)

The information regarding Softstat Processing principally comes from the slides in [int_5]. In Figure 6-17 the process Softstat processing is unfolded. The design of the object Weekly Softstat Report was created with the slides in this document as starting point. The objects Configuration Specification and VOB Storage, which are required for the process to be able to occur are not found in the documentation, but were added to the model after interviews.
6.3.10 Report Creating unfolded (SD 1.3)

Figure 6-18 shows the process Report Creating. Discussions with Mr. Theisz and Mr. Stikkel at the company showed that different stakeholders would be interested in different types of reports. This diagram shows these stakeholders. What type of information is connected to which types of reports is shown in the next paragraph.

![Diagram showing Report Creating process](Figure 6-18 SD 1.3 – Report Creating unfolded)

6.3.11 Report unfolded (SD 1.4)

The various stakeholders at the company are interested in different types of information. Figure 6-19 shows the different distributions that the reports are composed of. The information in this OPD is not written down in any documentation, but conversations with staff showed what information might be useful for who.

6.3.12 Report Creating in-zoomed (SD 1.4.1)

This diagram in Figure 6-20 shows how a report is created by selecting a project and a report type. The contents of this OPD are the creation of the author, and can not be found in the documentation.
Figure 6-19 SD 1.4 – Report unfolded

Figure 6-20 SD 1.4.1 – Report Creating in-zoomed
6.3.13 Chart unfolded (SD 1.1)

The OPD in Figure 6-21 describes the different parts various charts. The reason for creating this diagram was that this would help showing which values in a distribution are represented in a graph. The different types of charts are taken from example graphs in [int_2] and [int_9].

![Figure 6-21 SD 1.1 – Chart unfolded]

6.3.14 Employee unfolded (SD 4)

The diagram in Figure 6-22 shows the different stakeholders of the organization. The objects Assistant, Line Manager, Project Manager and Resource & Project Controller originate from paragraphs 5.2, 5.3, 5.4, and 5.5 respectively of the document Time reporting at ETH/R.

The object Quality Coord. derives from [int_7] and paragraph 5 in [int_4]. The Software Measurement Responsible arises from paragraph 1.1 in [int_3]. The Quality Manager and Configuration Manager arise from paragraph 1.3.2.1 in the same document. The Main Project Manager is described in [int_8].
Figure 6-22 SD 4 – Employee unfolded

Figure 6-23 illustrates a difference between the Project Manager and the Main Project Manager. The information comes from paragraph “2 Responsibilities” in [int_8].

Figure 6-23 SD 4.1 – Main Project Manager Unfolded

6.3.15 Tool unfolded (SD 9)

The OPD Tool unfolded (Figure 6-24) was created because Ericsson is planning to upgrade the current TRT Tool for a new, enhanced one called CATS. This information is not found in the internal documentation, but was gathered during interviews.

Figure 6-24 SD 9 - Tool unfolded

6.3.16 Deadline unfolded (SD 3)

All the deadlines modeled in this OPD (Figure 6-25) are found in the document Time Reporting at ETH/R. The **Send Report Deadline** arises from the sentence “Reports have to be sent on the first working day of each week until 14:00 p.m.” in paragraph 5.1. **LM Checking And Approving Deadline** comes from “Line Managers are responsible for checking whether project members have reported exactly as many hours as the Working Time Report offers on a weekly basis. The deadline for this checking is 18:00 p.m.”
Monday.” in paragraph 5.3 [int_1]. The **Reports Corrected Checking Deadline** originates from the sentence “Line Managers are responsible for assuring that all the rejected reports of their people are corrected. The deadline for this is 10 a.m. Wednesday.” in paragraph 5.3.

**PM Approving Deadline** comes from “Project managers are responsible for approving the hours reported on their projects on a weekly basis. The deadline for approving the hours is 14:00 p.m. Tuesday”, in paragraph 5.3. Finally the Assistant Mail Sending Deadline originates from the sentence “Assistant is responsible for sending a reminder mail to project members on the first working day of each week.”, in paragraph 5.2.

![Figure 6-25 SD 3 – Deadline unfolded](image)

### 6.3.17 TRT Weekly Report unfolded (SD 8)

The design of the TRT report is not specifically written down in the internal documentation. However, interviews with Mr. Theisz showed that it would be suitable to model the TRT report in the manner shown in Figure 6-26.

The objects that a **TRT Item** is composed of are found in for example [int_3]. The object **Project** can be found in paragraph 1.3.1, **Discipline** is found in 1.3.2 and Breakdown Structure comes from paragraph 1.3.3 of this document. **Activity**, **Artifact**, **MHR Type** and **Notefield** come from paragraph 1.3.4.1, 1.3.4.2, 1.3.4.3 and 1.4 respectively. The objects **Week** and **MHR** are data to be entered by employees. These are not found in the documentation but must naturally be represented as a part of the **TRT Item** object.
6.3.18 TRT Structure unfolded (SD 7)

The object **TRT structure** (Figure 6-27) originates from [int_2] and [int_3]. The objects **Project**, **Discipline**, **Breakdown Structure**, **Activity**, **Artifact** and **MHr Type** have already been explained in the previous section (6.3.16). The objects **TR** (Trouble Report) and **CR** (Change Request) comes from paragraph 1.3.3 in [int_3] and paragraph 2.2.1.5 in [int_2].

The level of the TRT tool the different objects belong to is described in paragraph 1.2 in the document [int_3]. In this paragraph it is also explained which stakeholders get data from which levels:

- **Controlling** gets data from Level 1 and Level 4.
- **Project managers** get data from Level 2, Level 3 and Level 4.
- **Quality** people get data from Level 1, Level 2, Level 3 and Level 4.
- **Software measurement** gets data from Level 1, Level 2 and Level 4.
- **Line managers** get data from Level 1 and Level 4.
Figure 6-27 SD 7 – TRT Structure unfolded

SD 7.1 – Activity unfolded – contains the three types of activities, CW (Create/Work), R&I (Review & Inspect) and RW (Rework). These objects are found in paragraph 1.3.4.1 in [int_3].

SD 7.2 – Artifact unfolded – contains some of the different artifacts found in the tables describing the different types of processes in the documents [int_2] and [int_3]. Since the number of artifacts is quite high only a few of them have been added to the model to serve as examples. If the need arises to put all the different types of artifacts in the model this can be done easily later on.

SD 7.3 – Breakdown Structure unfolded – contains two types of breakdown structures, namely WBS (Work Breakdown Structure) and PBS (Product Breakdown Structure). These originate from paragraph 1.3.3 in [int_3].

SD 7.5 – Discipline unfolded – originates from paragraphs 1.3.2.1 and 1.3.2.2 in [int_3]. It describes the different management and development disciplines.

SD 7.6 – MHR Type unfolded – consists of four types of man hours, MHR, MhrA, MhrI and STP described in paragraph 1.3.4.3 in [int_3].
6.3.19 Development Process unfolded (SD 7.4)

Development Process unfolded (Figure 6-28) contains the four types of development processes which are found in [int_3]. Each type of development process is described in a table, explaining which disciplines and artifacts are defined. Management Process is found on page 10, and Software Development Process At R&D is found on page 11 of this document. Mix Development Process and SPEED Development Process are found on pages 14 and 16 respectively.

In this OPD only the disciplines are defined for each type of process. By unfolding a discipline the artifacts which are connected to it are revealed.

Figure 6-29–PrM unfolded – illustrates what one of the unfolded disciplines, PrM (Project Management) looks like. You see that the artifacts Project Planning, Project Tracking, Meeting and OFI are defined for this discipline. The other disciplines unfolded follow the same pattern. As can be seen in figure 6-27 the artifacts have not yet been added to all disciplines since this was considered too time consuming.

Figure 6-28 SD 7.4 – Development Process unfolded
6.3.20 Project unfolded (SD 7.7)

Figure 6-30 shows the object project unfolded. Almost all the information in this OPD was gathered through interviews with Mr. Theisz. However, the objects **BTA** and **Name** are found in the sentence “Level 1 shall specify the project name with BTA number included if it exists…” from paragraph 1.3.1 of the document [int_3]. The objects can also be related to paragraph 2.2.1.1 in [int_2].
Chapter 7. Connection Between Model and Implementation

The Object-Process Diagrams displayed in chapter six served as a business model when creating the Software Measurement Presentation Tool. This chapter shows the connection between the OPM-model and the Microsoft Access program.

7.1 Introduction

The Model-View-Controller (MVC) is a commonly used and powerful architecture for constructing GUIs. It was created by the inventors of Smalltalk at Xerox PARC in the late eighties. After creating the OPM-model it stands clear that the diagrams shaping the implementation follows the MVC design pattern quite nicely.

In the MVC paradigm the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of objects, each specialized for its task. The **view** manages the graphical and/or textual output. The **controller** interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate. Finally, the **model** manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view) and responds to instructions to change state (usually from the controller). [9]

The **model** in the MVC paradigm is principally represented by the OPD shown in Figure 7-4. In the diagram you can see how the data is arranged in the TRT reports. The **view** part is connected to the OPD in Figure 6-21, which shows the different types of charts available. The **controller** part of the paradigm comes from Figure 6-20, which shows how a report is created through the user inputs of selecting a project and a corresponding type of chart.

The Model-View-Controller paradigm states that different parts of the OPM-model can be changed without affecting the functions of the others. A change in for instance the “view-part” does not affect the “model-part”. For example you could exchange the types of charts shown in Figure 6-21 for other types of visual representations and still keep the same data structure in Figure 7-4.
7.2 Using the OPDs for Creating the Implementation

How the charts in the Software Measurements Presentation Tool are connected to the model will be described now. The methodology will be demonstrated by the example of the “Discipline Distribution”.

To show which route to follow in the Object-Process Diagrams arrows have been added to them. In the OPM paradigm there is no natural way to make selections or limit the number of routes that can be taken in an OPD, so adding these arrows can be seen as an extension to the normal OPM.

7.2.1 The Controller

Starting from the object Report we follow the arrows in Figure 7-1 to select the type of distribution we are interested in. In this case this would be the Discipline Distribution.

![Figure 7-1 Selecting the Discipline Distribution from the reports.](image)

The diagram Report Selecting in-zoomed from Figure 6-20 also falls into the controller category. The diagram shows how the user can give input to the program.
7.2.2 The Model

After having selected the appropriate distribution we move on to the object-process diagram “Volume And Effort Analyzing” in Figure 7-2. Here we see that the information from the **Discipline Distribution** will be presented in a **Pie Chart**.

![Diagram](image)

**Figure 7-2** The information from a **Discipline Distribution** is presented in a **Pie Chart**.

In Figure 7-3 the objects **Project**, **Week**, **Discipline** and **MHR** are marked by arrows. These are the objects from a TRT report that are needed to create a Discipline Distribution.

![Diagram](image)

**Figure 7-3** The objects **Project**, **Week**, **Discipline** and **MHR** are needed to create a Discipline Distribution.
In Figure 7-4 the unfolded Discipline Distribution is shown. (1) The OPD demonstrates how a **Discipline Distribution** consists of many **Discipline Distribution Elements**, each of these elements consists of data from **TRT Weekly Reports**. Each **Discipline Distribution Element** consists of two objects, **Effort Spent** and **DD Discipline**. (2) The **Effort Spent** is a summation of **MHrs** (Man Hours) from many TRT Weekly Reports. It is presented as a percentage value, and represented as a **Piece** in a **Pie Chart**. (3) The **DD Disciplines** corresponds to the **Disciplines** found in the TRT Reports. All the Disciplines found for a project in the TRT reports are represented in the Discipline Distribution as well. Each **DD Discipline** is represented as a category in a Pie Chart.

**Figure 7-4** Discipline Distribution unfolded

The SQL-query that creates the graph for the Discipline Distribution in Microsoft Access is presented below.

```sql
SELECT discipline, Sum(Mhrs) AS SumOfMhrs
FROM Discipline_distribution_per_date
WHERE ((ID)=Project() AND (week>=StartWeek("Discipline Distribution") AND week<=StopWeek("Discipline Distribution")))
GROUP BY discipline;
```

**Figure 7-5** SQL-query used to create the graph Discipline Distribution in the Software Measurement Presentation Tool

The first sentence, “SELECT discipline, Sum(Mhrs) AS SumOfMhrs” corresponds to figure 7-4 where “discipline” matches the object **DD Discipline** and “Sum(Mhrs) AS SumOfMhrs” matches the object **Effort Spent**, which is the sum of MHr items.

The “FROM Discipline_distribution_per_date” shows where the data is selected from. “Discipline_distribution_per_date” is a query performed on TRT reports so this part of the SQL-query corresponds to the object **TRT Report** in Figure 7-3.
The “WHERE ((ID)=Project() AND (week>=StartWeek("Discipline Distribution") AND week<=StopWeek("Discipline Distribution")))” shows that you are only interested in the Projects and Weeks (see Figure 7-3) that the user has defined. Project(), StartWeek() and StopWeek() are filters to select the proper interval of investigation.

The "GROUP BY discipline" part of the SQL-sentence corresponds to the object DD Discipline which is represented by a Category in Figure 7-4.

### 7.2.3 The View

Figure 7-2 shows that the Discipline Distribution shall be represented by a pie chart. Figure 7-6 shows the layout of the different charts. In combination with the information in Figure 7-4 we can navigate our way down the diagram to find the objects of current interest.

![Diagram showing the selected type of chart and type of value.](image)

**Figure 7-6** The picture shows the selected type of chart and type of value.

The fact that the graph-type shall be a **Pie Chart**, as illustrated in Figure 7-2, can easily be selected in the Microsoft Access database. By simply adding the SQL-sentence from Figure 7-5 to a graph of type Pie Chart, a diagram looking similar to the one seen below in Figure 7-7 is received.
Figure 7-7 Graph representing the Discipline Distribution
Chapter 8. The Software Measurement Presentation Tool

The Software Measurement Presentation Tool presents information about various projects in different types of graphs. The tool is implemented in Microsoft Access.

8.1 Introduction

All employees at Ericsson ETH have to report how much time they spend working every week. This effort data collection is done through a web based tool called the Time Reporting Tool (TRT). Staff at Ericsson has to report how much time they spend on different projects, and what type of work they perform. Typically an employee has to select a project, a discipline, an artifact and fill out the time spent. An example could be Ericsson Project 1, programming, Java and 20 man hours.

Volume data measurement (lines of code produced) is carried out by the Softstat tool. The measurement is scheduled weekly, and data are put into the software measurement database.

The collected effort and volume data is used by managers to see how their projects are evolving.

The Software Measurement Presentation Tool (from now on referred to as SMPT) uses the information collected to create different types of graphs that may help project managers in their daily work.

8.2 Purpose

There were two main reasons for creating the SMPT. Firstly, it would save time for project managers. Today each project manager has to create their own graphs using Microsoft Excel. Instead of manually creating graphs from the collected data these diagrams can automatically be generated by the SMPT. The second, and most important reason, is that the way things are functioning now each project manager creates his or her own types of diagrams. This makes it hard to compare different projects to each other. By having standard charts, comparisons between projects will become a lot easier.

8.3 Design and Implementation

The SMPT is implemented as a Microsoft Access 2000 database. The reason for choosing this technique is that the data is represented in Microsoft Excel tables. MS Access is fairly easy to learn and use, and has built in functions for creating graphs. Using MS
Access instead of for example accessing a traditional database like MySQL or Oracle via a web-interface, was found to be the best solution since the selected approach would probably be far less time consuming.

The data is stored in MS Excel files that are linked into the database as tables. Some other information about for example what types of charts are available for which projects is stored in other tables. The relevant information is being selected by queries. A query is a request or a search for information contained in a database. The information is presented as graphs on various forms. Some VBA (Visual Basic for Applications) code was used, for example, to automatically update the tables at program startup.

8.3.1 Tables

The following tables are found in the database:

- **DisciplineConv**
  The hours spent working in a project is categorized into different disciplines. These disciplines may vary from project to project. To be able to make comparisons between projects this table is used to map the project specific disciplines into unified ones.

- **ProjChart**
  The table contains the different available chart types for each project.

- **Project_cpp (Linked Excel File)**
  These tables contain data about the amount of new, deleted or unmodified lines of C++ code for a project over different weeks. The Project in the heading above is replaced by the project title for each different project.

- **Project_java (Linked Excel File)**
  These tables contain data about the amount of new, deleted or unmodified lines of Java code for a project over various weeks. The Project in the heading above is replaced by the project title for each different project.

- **Project_TRT (Linked Excel File)**
  These tables contain data from the Time Reporting Tool. The Project in the heading above is replaced with by project title for each different project.

- **Project**
  The table contains a list of the different projects.

- **ProjectWeek**
  The table contains the highest and the lowest week for each project. It is automatically updated at startup.
• **TRT**
  The table contains all the TRT data for all different projects. It is automatically updated at startup.

• **WeekConv (linked Excel file)**
  Is used converting weeks from the format 2003/9 to 2003/09 so that Access shall be able to sort the weeks correctly.

### 8.3.2 Queries

By using queries you can select the data you are interested in from the tables or from other queries. The most frequently used queries can be stored in the Microsoft Office Database. The most significant queries can be found in Appendix A.

Not all SQL-sentences are stored as queries. Some have been entered directly into the property-fields of the graphs. A few examples of such SQL-sentences are found in Appendix B.

### 8.3.3 Forms

The forms are used to display information from tables and queries. The form `frmAnalysis` is the main window that appears at startup. From this form you can access the other forms which contain the graphs. All the other forms exist in two variants. One with data from all the weeks of the project and one with data only from specific time intervals. The second types of forms all have the suffix “Restricted”. (These forms are used to display graphs where the range of weeks have been narrowed down to only cover a specific period of the project.) Altogether there are 19 forms. Two for each chart type and one for the main navigation window.

### 8.3.4 Maintenance

The maintenance of the tool is not very complicated, but in order to be able to update the tool properly and add new projects you need to know which tables need to be updated and how. This is described in Appendix C
8.4 User Manual

At startup the main window (Figure 8-1) is automatically loaded. The window consists of two drop-down menus and a command button. The first drop-down menu contains a list of projects that have data stored in the database (Figure 8-2). The second drop-down menu contains the available chart types for the selected project (Figure 8-3).

By clicking on the command button, with the label “Show Chart”, after selecting both a project and a chart type, a new window will open containing the graph of the selected type for the selected project. By default the graph will contain data from all available weeks for a project.

Figure 8-1 Main window for the SMPT

Figure 8-2 Choosing a project

Figure 8-3 Selecting a chart type
There are six fundamental types of charts. These are Code Change Profile, Code Distribution, Code Growth Profile, Discipline Distribution, Discipline Evolution and Staffing Profile. In addition to these there are three other chart types: Universal Discipline Distribution, Universal Discipline Evolution and Universal Staffing Profile. These three charts are basically the same as the normal charts for Discipline Distribution, Discipline Evolution and Staffing Profile, the only difference being that the project specific disciplines which are displayed in the normal charts are mapped against some standard disciplines here to allow comparisons among projects. The different types of charts are described in more detail below. Since the “Universal” graphs are so similar to their normal counterparts these charts will not be explained.

8.4.1 Code Change Profile

The Code Change Profile shows the modifications carried out on the code volume during the weeks. Its purpose is to help monitoring how consequently the program development advances. A screenshot of a Code Change Profile is shown in Figure 8-4. The diagram utilizes data generated by the Softstat tool.

With the “Language selector” you can select if you want the chart to show the amount of modified Java code or the amount of modified C++ code. On the x-axis the weeks are displayed and on the y-axis you can examine the number of lines of code. You can narrow down the range of weeks, to show only those weeks you are interested in. To do this, simply select a “Start Week” and a “Stop Week” and then click on the “Refresh” button. By doing this a graph will open in a new window containing only information from the currently selected weeks.

Figure 8-4 Code Change Profile
8.4.2 Code Growth Profile

The Code growth profile shows how the code volume written in a particular programming language is changing in the course of the program development (Figure 8-5). The diagram utilizes data generated by the Softstat tool. It is similar to the Code Change Profile, with the difference being that it also contains the number of unmodified lines of code. In the same way as with the Code Change Profile you can narrow down the interval of weeks and select which language you are interested in, Java or C++.

![Code Growth Profile](image)

Figure 8-5 Code Growth Profile

8.4.3 Code Distribution

The Code Distribution shows you how the produced code is distributed among the different programming languages (Figure 8-6). The diagram utilizes data generated by the Softstat tool. By choosing a “Start Week” and a “Stop Week” and clicking the refresh button, a new window will open containing a graph with data only from the selected weeks.
8.4.4 Discipline Distribution

The Discipline Distribution shows the proportion of man hours spent on each discipline in a certain project (Figure 8-7). The diagram utilizes data reported to the TRT. By choosing a “Start Week” and a “Stop Week” and clicking the refresh button, a new window will open containing a graph showing data only from the selected weeks.
8.4.5 Discipline Evolution

The Discipline Evolution (Figure 8-8) uses data from the TRT. It shows how the different disciplines evolve as the project advances. By choosing a “Start Week” and a “Stop Week” and clicking the refresh button, a new window will open containing a graph with data only from the selected weeks.

![Discipline Evolution](image)

Figure 8-8 Discipline Evolution

8.4.6 Staffing Profile

The Staffing Profile shows the weekly effort distribution (Figure 8-9). It can help you keep track of the project’s evolution. Each week is represented by a bar showing the total amount of man hours spent that week. Each bar is divided into smaller parts representing the disciplines the effort was spent on that week. The Staffing profile uses data from the TRT. By choosing a “Start Week” and a “Stop Week” and clicking the refresh button, a new window will open containing a graph with data only from the selected weeks.
Figure 8-9 Staffing Profile
Chapter 9. Evaluation and Validity

The OPM-model of the software measurement process and the implemented tool are evaluated in this chapter and the validity of the drawn conclusions is discussed.

9.1 Evaluation of the OPM model

What the OPM-model should contain was decided by the company, since documents and additional information that should be modeled were selected by them. During the time the model was constructed the contents of the model were continuously monitored by my supervisor. Also meetings with a manager where held every two weeks, and the model was gone through by him as well. That the correct information has been accounted for in the model is therefore fairly safe to say.

In Chapter 4 one of the questions of research is if the Object-Process Methodology is suited for modeling the software measurement processes at Ericsson ETH/R&D. To answer this, the advantages and drawbacks of introducing the methodology are analyzed.

- **Advantages:**
  Using models is an effective way to train large groups to gain an understanding of how a structure works. Models may compress time and/or space, this is one of the greatest advantages of modeling in that it enables us to make connections between the different parts of the system while it helps us to visualize their interactions.
  
  The greatest benefit that could be gained from introducing OPM as a tool for modeling processes at Ericsson ETH/R&D is that it could reduce the time it takes to understand the different processes and their connections. OPM in itself is not a tool for reducing costs or simplifying the software creation process. However if a greater insight of the organization and the connections between different processes could be achieved, through the creation of an OPM model, this might result in ideas on how to improve the existing processes.

- **Drawbacks:**
  Introducing the OPM methodology in the organization is connected with certain costs. There might be a need to assign one or several persons to create the OPM models. The persons making the models should preferably be someone with good knowledge of the organization, and the creation of models might take up valuable time from him/her. Courses would have to be held, to teach the syntax of the OPM paradigm to everyone who might come across the models in their daily work. Often when a model is created it is with the intention to test or simulate a real life system. However, this is not the case here. The developed models are solely used for the purpose of aiding the understanding and comprehension of the organization. It can not be determined easily how much the creation of models for
the software measurement processes would actually contribute to this.

9.1.1 Recommendations

What conclusions can be drawn from weighing the benefits against the disadvantages? The benefits that can be gained from creating OPM models of the software measurement processes is that having the processes written down in a model instead of textual form might make them more easily understood. However having the documentation of the processes in textual form, as the case is today, does not require any knowledge about OPM-modeling from the staff.

The benefits of having the documentation in an OPM-model are not proven, probably many people would actually prefer having the documentation in textual form. The Object Process Language part of the OPM-model could perhaps be an adequate solution for these people. Nevertheless in OPL you can not express yourself as freely as in natural language. Also writing down the requirements in OPL, using OPCAT, is far more time-consuming than using a simple word processor such as for instance MS Word which is used today. Writing OPL-sentences instead of natural language would also require some form of tuition of the staff. At the moment OPCAT is the only existing tool for creation of OPM-models.

A lot of time and money would have to be spent on introducing the OPM methodology. Therefore the recommendation to the company is to not model their processes in OPM, at least not until they are sure that modeling processes in OPM will actually add value to their organization.

9.2 Evaluation of the Software Measurement Presentation Tool

The developed tool is a prototype, therefore the emphasis of the evaluation has been put on examining the functions of the tool, rather than other aspects such as usability or design. To begin with the plan was to evaluate the tool using a survey which was sent to the project managers at the company, but since the reply rate was very low this approach could not be taken. The Software Measurement Presentation Tool was instead evaluated by looking at different scenarios and estimating what tasks the tool could be used for.

- **Scenario 1 – Tracking Project Progress**
  Suppose a project manager wants to see the how his project is evolving. An accurate estimation of this could be received by looking at the changes in code-size. By looking at how the amount of code changes over the weeks the manager could for example determine if the project will finish on time (provided that he knows approximately how many lines of code a project of the size he is responsible for normally contains.). To track how the amount of code changes
over time he could use the “Code Growth Profile” in the SMPT. This diagram shows him the total amount of code for the different weeks. If for example a major setback would occur and big parts of the code had to be deleted and then rewritten this would distinctly show in the Code Growth Profile, and the manager would be able to take appropriate measures.

- **Scenario 2 – Quality Measurement**
  A quality manager (the person responsible for the overall quality of a project) or a project manager might be interested in measuring the quality of a project. This can be done by looking at the number of trouble-reports. A large number of trouble-reports, especially not remedied ones, would suggest a low quality of the software. There is no support for monitoring quality in the current version of the SMPT. This is however a useful function and should be considered for implementation in potential future versions.

- **Scenario 3 – Estimation of Project Resources**
  A project manager might wish to predict how much resources has to be spent on different disciplines. This could be done by looking at how much time has been spent in previous phases of the project. For example, looking at how long time was spent creating the requirement specifications could give you an idea of how much time will have to be spent on the actual design/coding-phase. Or looking at how much time was spent in the design-phase can give an approximation of how much time will be required in the validation-phase. If it looks like more resources will be needed than originally expected the project manager might have to request more personnel. How much effort is spent on different disciplines can be seen in the “Staffing Profile” diagram in the SMPT.

- **Scenario 4 – Estimation of Project Resources 2**
  To tell how much of a budget has been consumed a project manager might want to know the total number of hours spent in the project. If a large part of the budget has been spent early in project, action might have to be taken. It could be a useful tool for managers to see how many hours has been spent totally in a project until today. This function is however not implemented in the current version of the tool.

- **Scenario 5 – Project Progress Tracking**
  By looking at how the time spent on different disciplines varies over time a project manager might tell when it is time to move into the next phase of a project. For example if a manager sees that less and less time is spent on a specific discipline for example writing requirement specifications he can assume that this phase in the lifecycle will soon be completed, and he can then start the next phase of the project. The managers can see how the different disciplines evolve over time in the “Discipline Evolution” function of the SMPT.
9.2.1 Functions supported by the tool

The following functions are supported by the tool:

- The Code Change Profile and the Code Growth Profile shows how the amount of source-code changes over time. These functions can primarily be used for tasks such as estimation of project resources, project tracking, evaluation and progress reporting of ongoing projects.
- The Code Distribution, which shows how the lines of code are distributed among different programming languages, can be used for project specification or evaluation of ongoing projects.
- The functions Discipline Distribution, Discipline Evolution and the Staffing Profile show in different ways how much time has been spent on various disciplines. These functions can mainly be used for estimation of project resources, project tracking and progress reporting.

9.2.2 Functions not supported by the tool

These are a couple of functions that could be of use for project managers but have not been implemented into the tool:

- “Total effort” would show the total number of man-hours spent on a project. It could be useful for estimation of project resources or progress evaluation of projects.
- “Total number of trouble reports” could be used for estimation of project resources, quality planning and quality tracking.
- “Effort for each discipline” would show how much has been spent on each discipline and would mainly be used for estimation of project resources, project tracking and progress reporting.
- “Comment density in source-code” could be useful for quality planning and tracking.

9.2.3 Conclusions

The functions available in the Software Measurement Presentation Tool can all be of some use to the project managers. The least helpful function in this prototype is probably the code distribution, and it can be discussed whether this function should remain if the company chooses to make a web-based version of the tool.

The most important function the program lacks today is the support for trouble reports. The trouble reports are an easy way to measure quality in projects and could definitely be
of use for the managers. However, due to the way the information is organized in files containing data regarding the reports, this function could not be introduced to the tool conveniently.

9.3 Validity Analysis

The validity of the drawn conclusions is rather low. No proper evaluation could be performed, so the drawn conclusions are the opinion of a single person. One of the first steps Ericsson should undertake if they decide to create a web-based tool from the prototype is to perform a thorough evaluation of the Software Measurement Presentation Tool, because the evaluation in this chapter is basically the intuitive thoughts of one man with little prior experience of software engineering and limited knowledge of the organization.
Chapter 10. Conclusions

The objectives of this thesis were the creation of an OPM-model describing the software measurement processes at Ericsson ETH/R&D and the creation of a tool for presenting software measurement data. Both these goals have been met.

From a starting-point in internal documents, an extensive OPM-model describing the data collection, processing and analyzing has been developed to aid managers and other staff in the understanding of the organization.

A prototype tool has been created that presents various types of information in different charts. This tool is developed to aid managers in their work and for creating uniform graphs that can be compare between different projects with easy. The evaluation of this tool is still insufficient, therefore before developing a web-based version of the tool further studies are important.
References

[1] Dov Dori et al., *Object-Process Methodology as an Industry Enterprise Framework*  


http://dori.technion.ac.il/opm/documents/articles/article86.ppt

[5] XMI,  
http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci213944,00.html  


[9] How to use Model-View-Controller (MVC)  
Last checked 2004-05-17
Appendix A

Queries

By using queries you can select the data you are interested in from the tables or from other queries. The most frequently used queries from the Software Measurement Presentation Tool are listed below.

- **Code_measurements**
  Contains the volume data, i.e. the information about the new, deleted and unmodified lines of code for the projects.

- **Discipline_distribution**
  Shows the total amount of man hours per discipline for a project.

- **Discipline_distribution_per_date**
  Shows the total amount of man hours per discipline for a project for each week.

- **qryProjChart**
  Selects the available chart types for the project currently selected in the main window (e.g. the form frmAnalysis)

- **qryUniversalTRT**
  This query is used to transform the “project specific” disciplines into “standard” disciplines, so that different projects can be compared with each other.

- **TRT_original**
  The query merges the TRT information from the different projects. The query is copied into the table TRT every time the program is started.

- **Volume**
  This query merges the information about the volume data for the different project and languages.
Appendix B

SQL-sentences

To create the graphs that are displayed in the different forms you need to select the data that is to be represented. This is done by an SQL-query. Not all queries have been saved as demonstrated in section 8.3.2. Many SQL-sentences have been added directly to the “rowsource” (the field where you enter which data is to be displayed) of the graph objects. Some of the SQL-queries added this way are displayed below.

SQL-query for creating the Code Growth Profile:

```sql
SELECT Code_measurements.week, Sum(Code_measurements.deleted_lines) AS [Deleted Lines], Sum(Code_measurements.unmodified_lines) AS [Unmodified Lines], Sum(Code_measurements.new_modified_lines) AS [New/Modified Lines]
FROM Code_measurements
WHERE (((Code_measurements.project_ID)=Project()))
GROUP BY Code_measurements.week;
```

SQL-query for creating the Staffing Profile:

```sql
TRANSFORM Sum(Discipline_distribution_per_date.Mhrs) AS SumOfMhrs
SELECT Discipline_distribution_per_date.week
FROM Discipline_distribution_per_date
WHERE ((([Time_data].[ID])=Project()))
GROUP BY Discipline_distribution_per_date.week
PIVOT Discipline_distribution_per_date.Discipline;
```

SQL-query for creating the Universal Discipline Distribution:

```sql
SELECT qryUniversalTRT.discipline, Sum(qryUniversalTRT.Mhrs) AS SumOfMhrs
FROM qryUniversalTRT
WHERE (((qryUniversalTRT.week)>=StartWeek("Universal Discipline Distribution") AND (qryUniversalTRT.week)<=StopWeek("Universal Discipline Distribution")))
GROUP BY qryUniversalTRT.discipline;
```

As can be seen in the case of the Staffing Profile some of the SQL-sentences include tags that are not part of the formal SQL-notation. The tags “TRANSFORM” and “PIVOT” are not included in the official SQL-standard. These tags are used to create pivot-tables that are to be displayed in the graphs.
Appendix C – Maintenance of SMPT

To add a new project to the Software Measurement Presentation Tool you have to go through the following steps:

1. Open the Software-Measurement-Presentation Database by opening the file Software-Measurement-Presentation.mbd once again, while the main navigation is still open.

2. From the left hand side menu choose “Tables”. Double click on the table “Projects”. In the “name”-field enter the name of the new project to be added. A unique ID is automatically generated. Remember this ID-number and close the table Projects.

3. Open the table “ProjChart”. In the field “Chart” enter the types of graphs you want to be able to draw for this project. In the field “Project” select the unique ID for your project which was created in step 2. Close the table.

4. Link all the excel files containing data regarding your project to the database. This is done by right-clicking in the database window and choosing “Link Tables…”. Follow the instructions and choose suitable names for your linked tables.

5. If you have linked TRT-information to the database click on “Queries” on the left hand side menu. Open the query “TRT_original” and choose the SQL-view. (SQL-view can be selected by a button just below the File-menu.) In the SQL view add the data for your project. This is done by entering an SQL sentence like the one below at the of the current SQL sentence:

\[
\text{UNION SELECT Name, Department, Week, Duration, } \text{YourProject AS Level1, Level2, Level3, Level4, Actnote FROM YourLinkedTable}
\]

Substitute \text{YourProject} in the example with \textbf{exactly} what you named your project in step 2, and exchange \text{YourLinkedTable} with \textbf{exactly} what you named your TRT-table in step 4. Make sure that the semicolon is correctly placed at the end of the SQL-sentence.

6. If you want to be able to draw “Universal” graphs you need to enter which project specific disciplines shall be mapped against which “universal” disciplines in the table “DisciplineConv” and update the query “qryUniversalTRT” to reflect the changes in the table.

7. If you have linked tables containing volume information you have to create a new query called Volume_YourProject according to the existing structure, where \text{YourProject} is the name of your new project. Then you need to update the queries Volume_week, Volume and Code_measurements to include your new project.