A Study on Software Management Approaches: Proposing a Project Support Tool

Jenny Pettersson
Sammanfattning

Förord

Innehållsförteckning

A Study on Software Management Approaches: Proposing a Project Support Tool .................. i
Sammanfattning ................................................................................................................. ii
Förord ................................................................................................................................ iii
Innehållsförteckning ......................................................................................................... iv
Nomenklatur .................................................................................................................. v
Abstract ....................................................................................................................... 6
1 Introduction ................................................................................................................. 6
2 Software Process Management .................................................................................... 6
   2.1 Benchmark Process Models ................................................................................... 7
   2.1.1 ISO 9000/9126/SPICE .................................................................................. 7
   2.1.2 Capability Maturity Model ............................................................................. 8
   2.1.3 Goal/Questio n/Metric ................................................................................ 9
   2.2 Analysis of process models ............................................................................... 10
3 Project Management and Tools ............................................................................... 10
   3.1 Benchmark Project Models ............................................................................... 11
   3.1.1 Gantt-chart ................................................................................................ 11
   3.1.2 Critical Path Method ................................................................................. 11
   3.1.3 Program Evaluation and Review Technique .............................................. 12
   3.2 Analysis of project models .............................................................................. 12
4 Measures and Metrics of the Software Process ...................................................... 13
   4.1 Benchmark Software Metrics .......................................................................... 15
   4.2 A Closer Look at Software Metrics .................................................................. 15
5 Proposing a New Set of Software Metrics ............................................................... 17
   5.1 Implementing the Metrics ............................................................................... 19
6 Introducing a New Software Management Tool .................................................. 19
   6.1 Existing Tools for Software Management ..................................................... 19
   6.2 Deciding on a Management Approach .......................................................... 20
   6.3 Presenting a New Project Management Tool ................................................ 20
       6.3.1 ManagementClient.cs ............................................................................. 21
       6.3.2 DisplayData.cs ....................................................................................... 21
       6.3.3 DisplayPlan.cs ....................................................................................... 21
       6.3.4 InsertData.cs ......................................................................................... 21
       6.3.5 CreatePlan.cs ....................................................................................... 21
       6.3.6 Help.cs ................................................................................................. 22
       6.3.7 KnowledgeBase.cs ............................................................................... 22
       6.3.8 ManagementServer.cs ........................................................................ 22
       6.3.9 ManagementLibrary.cs ....................................................................... 22
   6.4 Analysis of the Suggested Management Tool ................................................ 23
7 Discussion ................................................................................................................ 23
8 Future work .......................................................................................................... 24
9 References ............................................................................................................. 24

Bilagor

Appendix – System Screen Shots .............................................................................. 26
Nomenklatur

CMM = the Capability Maturity Model developed by the SEI institute
FPA = Function Point Analysis, a measurement procedure in software management
Fuzzy logic/theory = the modelling of uncertain and imprecise knowledge or data
GQM = the Goal/Question/Metrics paradigm proposed by Basili et. al.
Metric = measurement attribute designed to give a status indication
A Study on Software Management Approaches: Proposing a Project Support Tool

Jenny Pettersson
University of Trollhättan/Uddevalla
jenny.pettersson@htu.se

Abstract

For many years, the software business has been struggling with problems like schedule overruns, exceeding budgets, and dissatisfied customers. The troubles experienced by many project managers have inspired the development of a number of process- and project models, over the years. Some of the models have contributed significantly to the development of process standards, while others have had less impact. The problems mentioned above have been a motivation for the development of a new project management system, presented in this degree project. The management system measures the project status using a set of adapted metrics, presented in this paper as candidate metrics for software project management. This degree project aims towards an improvement of the software development process measurement and management, and should be considered as a suggestion for project management support.

1 Introduction

The success of a software development procedure relies on how well the process is administrated and controlled. The management of a software process includes many ingredients; it involves the choice of process model, organizing of developing teams, tools, and methods. Scientists and others involved in the software engineering field have contributed with a diversity of methods and models to deal with the process management problems and there is more than one way to view this matter, which is reflected in the reports written by these authors [2, 6, 11, 13, 14, 15]. In the variety of frameworks and models that exist in the field, it can be difficult to keep the methods apart and to identify what they represent. Some methods are ad-hoc and dependent on the in-house methods of the company, while other approaches are more commonly acknowledged and practised, like the ISO standards [4] or the Capability Maturity Model [17, 18].

This degree project is an attempt to bring order to the concepts and to summarize the existing approaches on software management. It is also an investigation on how to manage a software process and what metrics are appropriate to use for project measurement. This paper proposes a new set of software metrics and describes the implementation of them into a new software management tool.

This is the paper disposition; the introduction in chapter one gives a short description the reason for choosing this subject as a degree project, while chapter two will give a more in-depth look of the software process management field, describing benchmark process models in the sub-chapters, followed by a model analysis. Chapter three deals with project management and investigates the models and tools used in this field, followed by a model analysis. Chapter four takes a closer look at the concept of measurement in process management, as well as the most commonly used measures and metrics and their possible application to a software process. In the end of chapter four, a proposal of a new set of software process metrics is presented. In chapter five, a new set of software metrics is created and proposed as suitable for process measurement. Chapter six describes the development of a new software process management tool, which implements the proposed metrics. Chapter seven contains discussion and analysis of the research performed in this degree project, giving the author’s point of view on software process management and lessons learned from this work. Chapter eight concludes this paper with a short description of future development of this work, while chapter nine presents the references used to ensure the scientific quality of this degree project.

2 Software Process Management

Software development processes often struggle with quality and productivity problems, resulting in exceeding cost-estimations or schedule overruns [2, 14, 24]. Difficulties with unreliable software with low performance and late deliveries have often led to frustrated customers. In the 60s and 70s, many software projects failed because of lacks
in management when techniques from other engineering disciplines where applied inefficiently on the software process [24]. Today, a selection of process models and tools are used with various success, some of which will be described in this degree project, for instance the ISO 9000 standards, the Capability Maturity Model, the Critical Path Method and the Gantt-chart.

One of the reasons that methods from the established disciplines have not been successfully applied to software engineering and the cause of the problems that historically occurred, is that there are no standard development processes. The software discipline is considered to be too young as a field for a successful establishment of generic software development [24]. Another reason for the difficulties mentioned above, could be that the products created in the process are abstract and not physical objects like a car or an airplane. This can cause problems in project tracking and supervision. A third cause to this dilemma may be that processes often differ from project to project, which makes it complicated to re-use experiences from earlier processes. The experiences could also quickly be out-of-date due to the emerging of new technologies, and as a result become impossible to re-use [24].

2.1 Benchmark Process Models

A process model can be used in software development measuring with the aim of assessing staff effort and project schedule. The assessment can be expressed in various project-related questions, e.g. How well does the process follow the planning? How does the actual work effort compare to the planned effort? Are milestones met? The process assessment may be carried out in the beginning of the process, during execution of a task or after the task execution. Process models can be either descriptive or predictive; the former can be used to describe the software process, while the latter may be applied to software process success estimation and calculation. Descriptive models can describe both process and product models. In a product model the quality of the software is measured; the most common quality measurement attributes are defined by the ISO 9126 standard, see chapter 2.1.1.

A process model deals with the measurement of the software development procedure elements, and chapter two will describe some of the most known software process models. The description includes the ISO 9000/9126/SPICE, the Capability Maturity Model and the Goal/Question/Metrics paradigm. These models were chosen because they are currently the most recognized and practised methods in the field [14, 24, 25, 27].

2.1.1 ISO 9000/9126/SPICE

The ISO 9000 standards, formed by the International Organization for Standardization (ISO), define a series of guidelines aiming towards a quality management system. The framework can be adopted by any organization as an attempt to improve the business procedures. The reason for embracing the ISO 9000 standards may, however, differ among organizations. The reason could be customer-related if the company chooses to follow the wishes of customers and stakeholders, or management-related if the decision to manage the business procedures is based on the ambition to improve process quality. When the reason is management-related, the desired productivity and product quality improvement of is usually reached [25]. Some of the benefits supposedly gained form using the ISO 9000 standards are; reduced costs due to efficient use of resources, and improved, consistent and predictable outcome of the process. A detailed list of the advantages of applying this process model follows [27]:

- Activities essential to achieving a desired result are defined systematically.
- Responsibilities for managing main activities are recognized.
- Analysis and measurement of the capability of main activities are made, as well as identification of interfaces within and between the functions of the organization.
- Focus on resources is gained, supposedly leading to an improvement of the main activities.
- Evaluation of risks, consequences and impacts of activities can be tested on customers, suppliers and other involved partakers.

The ISO/IEC 9126 standard was developed as a method for defining software quality, and its characteristics are believed to be the foundation of software product quality [4]:

- **Functionality** – a set of attributes that concerns the functions and their specified properties.
- **Reliability** – a set of attributes that concerns the capability of the software performance under stated conditions for a stated period of time.
- **Usability** – a set of attributes that concerns the effort needed for usage and the individual evaluation of such use.
• **Efficiency** – a set of attributes that concerns the relationship between the level of software performance and the amount of resources used under stated conditions.

• **Maintainability** – a set of attributes that concerns the effort needed to make specified modifications.

• **Portability** – a set of attributes that concerns the ability of software to be transferred from one environment to another.

The **SPICE model** (Software Process Improvement and Capability dEtermination) was developed from the ISO/IEC 15504 standard, and it aims directly at process improvement. The framework consists of nine parts; see figure 2.1.1 [27]:

<table>
<thead>
<tr>
<th>The 9 parts of SPICE</th>
<th>I Concepts and introductory guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>II</td>
<td>A reference model for processes and process capability</td>
</tr>
<tr>
<td>III</td>
<td>Performing an assessment</td>
</tr>
<tr>
<td>IV</td>
<td>Guide to performing assessments</td>
</tr>
<tr>
<td>V</td>
<td>Assessment model and indicator guidance</td>
</tr>
<tr>
<td>VI</td>
<td>Guide to qualification of assessors</td>
</tr>
<tr>
<td>VII</td>
<td>Guide for use in process improvement</td>
</tr>
<tr>
<td>VIII</td>
<td>Guide for use in determining supplier process capability</td>
</tr>
<tr>
<td>IX</td>
<td>Vocabulary</td>
</tr>
</tbody>
</table>

![Figure 2.1.1](image)

**Figure 2.1.1 The characteristics of the SPICE model**

The ISO standards have reached a fairly high level of approval across Europe, as a method for systematizing the processes of an organization. In the U.S., the standards have not reached the same level of acceptance; the American counterpart for ISO is the Malcolm Baldrige National Quality Award. The award is given annually by the U.S. Department of Commerce to acknowledge companies that stand out in quality management [14]. Studies have been made to analyse whether the ISO model actually improves the software process [4, 25]. Interviews with managers of several European software companies gave that almost 100% of those who adopted the 9000 standards were satisfied and would choose them again. However, Stelzer, Mellis and Herzwurm [25] points to a fact that should be considered in this context; many of the interviewed companies had already reached an optimized maturity level in the development process when they sought certification. For most of the companies, though, using the ISO 9000 has led to improvements with stable processes, organizational structure and engineering standards [25]. Conclusions drawn by Bazzana, Brigliadori, Andersen and Jokela [4] were that the standard is effective but not enough to guarantee quality of the software product. They also say that the 9126 standard is the definite benchmark for quality modelling of software.

### 2.1.2 Capability Maturity Model

The Software Engineering Institute, SEI, is the creator of a software process management model, named the **Capability Maturity Model**, CMM. The CMM consists of a set of practice recommendations, split up into key process areas, intended to measure and improve the maturity of a software development procedure. SEI defines process maturity as the level, to which a specific process is defined, managed, measured, controlled and effective [17, 18, 24]. In an immature organization, the processes are generally improvised and managers often focused on solving immediate crises instead of taking failure precautions. Difficulties in reaching estimated results, leading to an unpredictable outcome of the project, are common in an immature management. The mature organization follows a well-defined process and has taken preventive actions with realistic estimations based on experiences and results from previous projects. The processes of the company are disciplined and controlled. The CMM model consists of five steps, see figure 2.1.2.
sufficient tool to measure the maturity of an organization [24]; (1) it focuses only on project management and not on the aspects of product development, (2) there is no risk analysis in the model and (3) there is no definition of application domain for the model. In some processes it may not be appropriate at all. An important aspect when using the CMM model is that it was initially developed to measure the maturity of companies producing defence-systems software, and the model is therefore not optimised for use in software development in general [24].

2.1.3 Goal/Question/Metric

The Goal/Question/Metric (GQM) paradigm, suggested by Basili et al., is a method to identify measurable software goals for process improvement [1]. The method works from the top-down, starting with identification of goals for process improvement, for example; improve the development process, improve software estimation, improve project tracking, improve software quality or improve company productivity. The goals are then refined into a series of questions. Examples of questions that relate to the goal “improve project tracking” might be;

- What is the overall status of the project?
- What is the earned value\(^1\) of each activity?

From these questions metrics are formulated, by which it is possible to measure the process and confirm that the desired goals are reached. These metrics can be an automated measurement, active data collection or interactive information gathering in co-operation with the user [3].

The authors present the GQM method as a way to formalize the improvement paradigm [1, 2, 3]. This paradigm aims at providing a basis for organization improvement and self-awareness, and it consists of the following steps;

1. Characterize the current project environment
2. Set up goals and refine them into quantifiable questions and metrics for successful project performance
3. Choose the appropriate software project execution model, supporting methods and tools for current project
4. Execute the chosen processes and construct the products, collect and

\(^1\) Earned value is a measurement of the quantity of work completed in the process.
provide an adequate support for process improvement, and that it has not been sufficiently tested or practised by software houses to form a useful model.

The most promising and popular approach to process improvement presented in this paper, appears to be the GQM-method. The method is flexible and adaptive to most processes, by the separation of organization goals and project concerns. The goals and questions formulated when using this method signify the specific business needs and can be used to solve the particular problems of the organization. The method is, supposedly, sufficient in capturing the relevant process issues and building self-awareness in the company, which is a good start to a software process improvement procedure. The method is clearly specified and seems straightforward to apply to any process.

2.2 Analysis of process models

The ISO 9000 model has had some success in developing processes in Europe and it has a good reputation amongst software houses. It appears that the model can be used as a formal method to improve a process, even though some companies seem to be seeking certification merely as a symbol of status. While standardizing the business procedures may considerably enhance the software process, organizational structures and engineering methods, the ISO 9000 standards are not believed to be able to guarantee quality of the software product. For this reason, the 9126 standards are more appropriate as a product model as it defines the quality attributes for software. The SPICE model aims directly at software process improvement by providing a framework for the assessment and should be able to guarantee an enhancement, if properly applied by the manager. It seems that the model is not widely practised and tested yet, however.

The CMM, on the other hand, seems to have had little success as an improvement method for software processes, supposedly as an effect of it not being widely understood by organization managers. While many organizations have been able to reach the lower levels of the model, hardly anyone achieves the higher levels that could lead to process improvement. As mentioned in chapter 2.1.2, there are more problems related to the CMM. Allegedly, the model focuses merely on project management and takes no consideration to product development and quality assurance. The model has also been criticized of been unclear and its domain application not specified, which could mean that in some processes the model is not at all applicable. It seems that the CMM is not specific enough to provide an adequate support for process

3 Project Management and Tools

Chapter two presented process management and its common approaches. This chapter will take a deeper look at the software development procedure and handle important issues of project management. Project management involves several elements, for instance estimation of effort, time and costs, planning of schedules, organization of staff structure and quality assurance of the software product. The software estimation can be performed using several different methods. The analogy method compares the current project to a previous completed project with similar characteristics. This method has the advantages of being based upon past experiences. On the other hand, there may be some concerns if there is no previous project to compare with, or if the data from earlier project are incorrect. The bottom-up method estimates each component and combines the result into an estimation of the entire project. This method provides a detailed and accurate basis for estimations and supports direct project tracking [20]. It may on the other hand be difficult to perform in early stages of the development cycle and has a tendency of being time-consuming [20]. The top-down method starts by estimating the overall structure of a project and then breaking it down to lower-level components and phases. This method may be more applicable to early estimations when only the global properties of the project are known [20]. An advantage of the method is that it considers the system-level activities, like project control and configuration management. It also tends to be faster and easier to implement then the bottom-up method, and also it requires a minimal project detail. There are some disadvantages, however; the method may be less
accurate and it may fail to notice some lower-level components of the project [20].

The project manager is usually responsible for the project planning and organizing, which includes assessing available resources. This may be carried out in the formulation of a project plan with the most critical tasks of the project, the milestones, marked out. The plan usually includes the following parts: introduction, project organization, risk analysis, hardware and software resource requirements, work breakdown, project schedule and monitoring and reporting techniques.

When a software process is assessed this is usually performed on an organizational level, with the purpose of evaluating the processes used in developing projects. For this purpose, one or several of the process models described in chapter two may be used. Project management also have different approaches, techniques and models, which are more or less acknowledged and practised. This chapter will summarise and describe some of the most commonly used methods.

### 3.1 Benchmark Project Models

In the 1950s, large development programs lay the foundation of current project management technologies, with methods like the Critical Path Method, CPM, the Program Evaluation and Review Technique, PERT and the Gantt-schedule [6, 20]. These techniques are well known and have been influential to the approach used by many project managers today [20, 22]. The CPM, PERT, and the Gantt schedule may be used as tools in project planning and reporting. If project tracking and status reporting is performed during project implementation, it can provide the manager with updated information on project schedule, costs and calculations of the earned value. Project tracking can help the manager predict the outcome of the project and give assistance in making necessary decisions [10]. Progress may be displayed by presenting graphs over the trends and indications of the project process. A useful graph could for instance provide an overview of errors per lines of code relatively the design complexity, or defects identified relative the time passed [10].

#### 3.1.1 Gantt-chart

The Gantt chart is a graphical tool for displaying project schedule, developed by a shipbuilder called Henry L. Gantt during the World War I. The model consists of tasks represented as horizontal bars positioned in a time scaled graph, where the length of the bar represents the task duration and the position of the bar denotes start and completion dates [20], see example borrowed from [20] in figure 3.1.1.

Figure 3.1.1 A Gantt chart showing the project schedule

The Gantt chart has been involved in improvement experiments using the fuzzy technology [23, 16], an area widely known and explored in the field of artificial intelligence. In project management, the fuzzy theory has been tested as a way to represent vague numeric values. In one experiment, the authors succeeded to implement further project knowledge details than what was provided by the chart itself, and thus making it an even more useful tool for project management. The authors claimed that this approach would optimize the Gantt chart by decreasing the idle time in the schedule [23].

There have also been experiments performed using Gantt charts for fuzzy scheduling and applying a genetic algorithm to search for multiple solutions [16]. In this experiment, the authors investigated the possibility to calculate the most efficient path of tasks in a project. The conclusion of the test was that the fuzzy theory could be used to improve the task scheduling, but only if the vagueness of activities was not too large.

#### 3.1.2 Critical Path Method

The Critical Path Method, CPM, was developed as a project management method between 1956 and -59, as a method for finding critical paths of a project, using a calculation algorithm. The computation is performed on a network-chart where sets of activities and links form a directed graph, and each activity is defined by two events, start and completion [15], see figure 3.1.2.

Figure 3.1.2 A Critical Path Analysis applied to a project network graph, example borrowed from [13].
The CPM emphasizes the interconnections between activities, their predecessors and successors, and could be used to model all possible situations that may occur in the project. The minimum time to completion in a CPM network is represented by the longest path from start to finish.

The CPM method can be used in three ways [28];

- The enumeration method, which identifies all paths, calculates completion time for all paths and finds the path with the maximum completion time – the critical path.
- The work-on-the-diagram method, which organizes the calculations by using a graphical network representation.
- The two-pass tabular method, which provides a systematic tabular approach. First, a forward pass through the activity list of the project is made to calculate the earliest start and finish times. Then a backward pass is made to calculate slack times from the latest start and finish times. Finally, the critical path is identified by connection of all activities with zero slack time. Investigations have shown that the most common approach to project management is the two-pass tabular method [28].

The CPM method has been tested in various ways, for instance in temporal logic programming [13], fuzzy logic [22] and as an approach to model concurrent projects [12].

### 3.1.3 Program Evaluation and Review Technique

The Program Evaluation and Review Technique, PERT, developed by the U.S. Navy in the 1950s, is intended to assist an estimation of project time-span for the optimistic, pessimistic and most likely scenario. The PERT method introduced the concept of uncertainty in project plans, which have led to a greater managerial understanding of the consequences of schedules overruns [20]. A PERT-chart shows the project activities as a network with all the major steps and precedence relationships between tasks. The model is similar to the CPM method and the two have often been combined, resulting in them being equivalent to each other. The difference between the models is that the CPM focuses on minimizing project costs, while PERT concentrates on optimizing project duration by showing the relationships between activities. The PERT model has successfully been applied in project management as a decision support tool [22, 28]. X-PERT, a graphical project tool created by the authors of [8] provides assistance for development and analysis of PERT diagrams. This tool is intended to aid managers in their decision-making by visualizing the project data.

Several experiments have been performed, for instance in [22], using fuzzy PERT to calculate important project attributes, like the latest and the earliest starting time of each project activity, and multiple critical paths. The reason for using fuzzy networks and numbers in project management is to try to generate reliable information from vague data, and by this create a thorough base for decision-making.

### 3.2 Analysis of project models

The Gantt chart has often been practised as a project scheduling method and has influenced the project management performed by organizations today [20]. The method offers an easy way of structuring the project activities relative the time-span, which gives a project overview. However, some studies have shown that the method may be selected even when it is not believed to provide a useful tool [26]. Further, the authors of this paper find the Gantt chart rather inadequate for software project management, because it lacks modelling possibilities for important project issues, other than time-concerns. The model is not preferable when the manager needs to have a more detailed project overview, with reporting on specific data regarding the development progress.

The CPM is considered a sufficient approach to project management, but it appears to be a rather vague model. There have been some experiments to introduce further constraints and aspects [13] by the use of temporal logic programming. The project network constructed using standard CPM is converted to a mathematical model, which may be used to calculate the critical path. The general opinion about the method seems to be that though it is a sufficient model to use in small, simple projects where the tasks are clearly specified and well known, it lacks the ability to give a detailed project plan. The CPM language is by some considered too weak to model all possible situations of a complex assignment, for instance in design or research projects [13] [12]. Projects for innovation, research and concurrent engineering may not be appropriate for the CPM method, as they could have a rather vague task description.

The PERT method resembles the CPM and the two has often been combined to complement each other [28]. PERT presented the uncertainty concept in project plans, and this has supposedly led to a greater understanding of consequences of schedule slippage, amongst project managers. The opinion about the possible benefits of using the method is somewhat ambivalent. Although, it has been successfully applied to projects as a support
for decision-making [22, 28], it is also accused of being too vague to provide a sufficient tool [12, 22]. The CPM and PERT should perhaps be considered insufficient as project management tool for any project that is not clearly defined [12].

4 Measures and Metrics of the Software Process

The previous two chapters described different models and approaches to software management. This chapter will take a closer look on the specific measurements that may be applied to project management. It will also propose a set of measurement attributes, metrics, suitable for project status tracking based on the research performed in this degree project work.

Measuring the software process is an important ingredient to software management, which may help prevent unexpected results and unsatisfied customers [14, 24]. If the measurement reveals that process quality enhancement is necessary, there are six steps that should be followed [9]. The first step is to identify the characteristics of the project and its environment, through the use of suitable metrics and models. The second step is to define goals for the process improvement. The third is to select an appropriate process model. In the fourth step, the process should be executed to gain feedback. The fifth step is to analyse the retrieved data, while the final step is to record and store the experiences of the entire improvement process. Measurement through observations and generalizations is crucial to the progress of all sciences; it is through the analysis of gathered data that new knowledge is derived [14].

There are four levels of measurement, according to Kan [14]: the nominal scale, the ordinal scale, the interval scale and the ratio scale. In science, the lowest level of measurement is classification of objects into categories [14]. The requirements for using the nominal scale are that the categories are jointly exhaustive and mutually exclusive. The former term means that the classification of categories should be thorough and cover all possible attributes, and the latter term means that an object should only be allowed to belong to one category. When these two requirements are fulfilled it is possible to apply a statistical analysis on the classification of objects to perform the measurement. The ordinal scale provides measurement through order comparison of objects, e.g. the CMM maturity levels of projects. This type of measurement is performed at a higher level than the nominal scale, by not only grouping objects together but also ordering them. An example of an ordinal scale is the Likert scale, which is a five-point scale for measurement. The ordinal scale cannot be used for mathematical operations other than comparison, for instance; A > B, C < A. It is not possible to apply addition, subtraction, division or multiplication to this scale. The interval scale, however, do have potential for applying addition and subtraction operations, which makes it possible to mathematically specify the exact differences between measurement points [14]. This scale requires a well-defined unit of measurement, which is repeatable and set as a standard. If a non-arbitrary zero point can be located on an interval scale, it is a ratio scale. This scale is the highest level of measurement and it is possible to apply all mathematical operations to it, including division and multiplication. This is the most commonly used scale. For both ratio and interval scales, measurement can be expressed in both integer and non-integer data. Example of integer data is frequency counts, e.g. number of defects. The metrics should preferably suit a high level measurement, since it is possible to reduce a higher level to a lower, but not vice versa [14].

The basic types of measures commonly used are ratio, proportion, percentage and rate. The ratio measure is derived by dividing two numbers, see formula 4.1.

**Formula 4.1 The ratio measure**

\[
\frac{\text{Number of defects}}{\text{Lines of source code}}
\]

One of the most common ratio measures in the software business [14] is the one shown by formula 4.2.

**Formula 4.2 The most common software measure**

\[
\frac{\text{Number of people in test group}}{\text{Number of people in development group}}
\]

The proportion measure differs from the ratio in that the numerator in the proportion is a part of the denominator, see formula 4.3.

**Formula 4.3 In a proportion the numerator is also a part of the denominator**

\[
p = \frac{a}{a + b}
\]

A proportion also differs from a ratio in the possibility to form multiple groups, see formula 4.4, whereas the ratio is best used for two groups.
Formula 4.4  A proportion measure applied to multiple categories of a group

\[
\frac{a}{N} + \frac{b}{N} + \frac{c}{N} + \frac{d}{N} + \frac{e}{N} + \frac{f}{N} = 1
\]

The **percentage** measure is a proportion expressed in terms of per hundred units, and is often used (or misused) to report results. It is important to give the measure together with enough contextual information, a fact sometimes ignored in reports. The total amount of test cases should also be large enough to provide a stable measure, at least 30 cases [14]. If the amount of test cases is lesser, it may be more appropriate to use absolute numbers instead. Table 4.1 shows an example of percentage distribution presented in tabular form, as well as the absolute number on the right (N).

**Table 4.1 Percentage distribution measures in tabular form**

<table>
<thead>
<tr>
<th>Type of defect</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Total (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement%</td>
<td>27.5</td>
<td>35.6</td>
<td>36.9</td>
<td>100.0 (83)</td>
</tr>
<tr>
<td>Design%</td>
<td>18.9</td>
<td>49.2</td>
<td>31.9</td>
<td>100.0 (60)</td>
</tr>
<tr>
<td>Code%</td>
<td>75.0</td>
<td>13.6</td>
<td>11.4</td>
<td>100.0 (117)</td>
</tr>
<tr>
<td>Others%</td>
<td>19.4</td>
<td>67.2</td>
<td>13.4</td>
<td>100.0 (101)</td>
</tr>
</tbody>
</table>

The ratio, the proportion and the percentage measure are all static summary measures that provide a cross-sectional view at a specific time [14]. The **rate** measure is often associated with the dynamic changes of interests, which generally is a measure of the change of one quantity (y) per unit of another quantity (x) on which the former depends. Formula 4.5 shows the risk of defects in software, as a function of the number of defects in proportion to the opportunities for errors (OFE). To specify the defect rate per thousand lines of code, or KLOC, (a common declaration) the rate is multiplied by the constant K (1000).

**Formula 4.5** The rate measure showing the risk of defects in software per KLOC

\[
\text{Defect rate} = \frac{\text{Number of defects}}{\text{OFE}} \times K
\]

A specialized defect rate measure is the **Six Sigma**, proposed by Motorola in 1989 to define quality in products and organizations. The symbol \( \sigma \) (sigma) is the Greek letter for standard deviation, which also defines what the Six Sigma measure is, see figure 4.1.

The figure shows that the areas under the curve of normal distribution defined by standard deviations are constant in terms of percentages, regardless of the distribution parameters. The area under the curve as defined by plus minus one standard deviation, or sigma, from the mean is 68.25%. The area outside the six sigma limit 100% - 99.9999998% = 0.00000002%, that is 0.002 per million parts. The purpose of applying this curve to a software process is to show the defect rate in the product, where the goal would be to produce software that falls within the Six Sigma limit. The defect rate would then be sufficiently low to guarantee a high quality product, according to Motorola. Recently, the Six Sigma measure has been applied to management improvement and total quality improvement, and in the software
industry sigma six may be defined in terms of defect level in software product. Six Sigma for a software product quality is defined as 3.4 defects per million lines of code [14].

4.1 Benchmark Software Metrics

Software metrics are descriptive measures of the software development process, or measurement attributes, which may be used for effort prediction or quality assessment [11]. One of the most known and probably most used metric is the lines of code metric (LOC), perhaps also one of the most complex metric, since the result of line counting may vary among different languages and different programmers [14, 20, 21]. In BASIC, PASCAL and C, one line of code can contain several instructions and comments. It has also been described as any line of program text that is not a comment or a blank line, regardless of the number of statements or fragments of statements on the line, including all lines containing program headers, declarations and executable and non-executable statements, by Conte et al. in Software Engineering Metrics and Models (1986) [14].

The LOC metric may be used in different ways, see example of possible application of the metric below.

- Count only executable lines
- Count executable lines plus data definitions
- Count executable lines, data definitions and comments
- Count executable lines, data definitions, comments and job control language
- Count lines as physical lines on an input screen
- Count lines as terminated by logical delimiters

Other metrics commonly used are function points and object points, which were created as a result of the difficulties using the LOC-metric [20]. To calculate the function point metric, count external input, inquiries, output, master files and interfaces, adjust this number for complexity and finally sum the results into a Function Point Count (FPC). The major benefit of using the FPC metric over the LOC metric is its independency on the programming language. However, it has been criticized of being hard to use. Counting object points include counting screens and reports, and then adjusting for complexity and summing the results. Object counting has not yet proven to be very useful, compared to the FPC [20].

4.2 A Closer Look at Software Metrics

Software metrics can be divided into three categories; product, process and project. Product metrics measure size, performance and code complexity (e.g. measured in terms of system structure, call paths, or inheritance hierarchies), while process metrics measure improvement and maintenance, and project metrics handles cost, schedule and productivity issues. Some metrics can belong to multiple categories, e.g. software quality metrics. Common product quality metrics include [14]:

- Mean time to failure, MTTF
- Defect density
- Customer problems
- Customer satisfaction

The defect density metric and the MTTF metric are similar but not identical; the former measures the defects relative the software size, and the latter measures the time between failures. There is a difference between the terms defect and failure. The IEEE/ANSI definition of these terms is presented below.

- An error is a human mistake that results in incorrect software.
- The resulting fault is an accidental condition that causes a unit of the system to fail to function as required.
- A defect is an anomaly in a product.
- A failure occurs when a functional unit of a software-related system can no longer perform its required function or cannot perform it within specified limits.

Simplified, the definitions mean that when an error occurs during the development process, a fault is injected in the software. In practise, however, there is no difference between the terms defect and fault. The International Organization for Standardization has established that the principles for product metrics are; (1) data should be collected and reported regularly, (2) the current level of performance on each metric should be identified, (3) corrective actions to the project should be taken if the metric level is increasingly worse or exceeds the target levels, and (4) it is necessary to establish goals for enhancement of the metrics [14].

Process metrics should be used to control that the quality goals of the process are met, and they
should specify how well the development process is performed, as well as how effective the process is at reducing the probability of failures. There are three types of measures to evaluate a development process [24]: time-consumption for the process – for instance total amount of time devoted to the entire process, resources acquired for the process – for instance total effort in person hours, travel expenses or computer resources, or the number of occurrences of one type of event – for instance number of errors detected or requirements changed. Examples of process metrics can be;

- Project size and team activity, such as lines of code and function points
- Metrics for schedule, such as activities completed on time and changes to schedule
- Metrics for requirements specification, such as number of requirements and changes to requirements
- Metrics for software testing, such as percentage of tested SLOC
- Metrics for software quality, such as the fault density
- Metrics for project risks, measuring the confidence that the product being “ready on time”

The type of metrics ultimately selected by the project manager will depend on the goals set for the development process. The goal can be, for instance, to maximize customer satisfaction, to minimize engineering effort and schedule or to minimize product defects. The reason for choosing to maximize the customer satisfaction is generally that the company is trying to win market shares, generally effective in the initial business start-up process [10]. The metrics used in this situation include surveys and interviews, product and defect metrics. The drawback of this approach is that it is not guaranteed to lead to any process improvement. The FURPS+ model, see figure 4, may be used as a way of clearly defining metrics for maximizing customer satisfaction. If the company chooses to aim for minimized effort in the development process, the reason is often the external pressure to improve the business procedures to compete with other companies on the market [10]. The metrics that relate to this approach are calendar time, engineering effort and defects. The third possible choice of strategy, minimizing defects, is often selected to hold or increase market shares when the company has reached a relatively strong market position [10]. Here, the metrics failure analyses by module, cause and severity, size and code coverage, are relevant. This approach may have the disadvantage of fixed defects not being cost effective [10].

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Capabilities</td>
</tr>
<tr>
<td></td>
<td>Generality</td>
</tr>
<tr>
<td></td>
<td>Security</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usability</th>
<th>Human factors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Consistency</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reliability</th>
<th>Frequency/severity of failure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recoverability</td>
</tr>
<tr>
<td></td>
<td>Predictability</td>
</tr>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td></td>
<td>Mean time to failure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Efficiency</td>
</tr>
<tr>
<td></td>
<td>Resource</td>
</tr>
<tr>
<td></td>
<td>Consumption</td>
</tr>
<tr>
<td></td>
<td>Response time</td>
</tr>
</tbody>
</table>

| Supportability | Testability |
|               | Extensibility |
|               | Adaptability |
|               | Maintainability |
|               | Compatibility |
|               | Configurability |
|               | Serviceability |
|               | Installability |
|               | Localizability |

**Figure 4** The FURPS+ model can be used to define relevant measurements

Choosing the minimizing effort and schedule strategy provides a more process-oriented view to project management, and for many software projects it is often the most effective way of meeting both customer and organization needs [10]. Example of applicable metrics with this focus;

- Engineering months by product/component/activity
- Engineering months by defects/enhancement
- Defect and enhancement request counts
- Counts of remaining critical and serious defects
- Calendar time by phase and activity
- Code stability (percent of code changed)
• Size (NCSS, tasks, objects, components)

The approach minimizing defects is purely process-oriented, and it includes the following metrics;

• Count of pre-release defects
• Count of post release defects
• Count of remaining critical and serious defects
• Count of defects sorted by module and cause
• NCSS (Non-Comment Source Statements)
• Percentage of branches covered during test
• Calendar time by phase

To succeed with this strategy there are different techniques possible; one is to use test programs continuously in the process, another is failure analysis. With extensive testing the manager can monitor the defects found and make sure that no new defects are introduced in the product.

5 Proposing a New Set of Software Metrics

This degree project started with an examination of software process and project models, currently practised in the field. The reason for this was to investigate if any of them could be appropriate for creating a new set of software metrics for project measurement. The intention of the metrics proposal was to implement them into a new software management tool; this will be described thoroughly in chapter five. In the model analysis, it was found that the most promising process model of those described was the GQM method, due to the flexible and adaptable character of the model. The GQM method provides a superior support for identifying the improvement areas of the process. The model is also easy to understand and apply to any process, and its practical use is unquestionable in contrast to the other models illustrated. Even though the authors of this paper appreciates the influence the above-mentioned project models, Gantt schedules, CPM or PERT, have had on project management in general, none of them were found suitable for this particular software management proposal. On the contrary, the GQM method by itself was found to be quite advocate for the identification of software process measurements.

In a pre-study to this degree project, the authors created a list of metrics suitable for process management. The metrics list was created using on the GQM approach, starting with goal selection as presented below.

Organizational goals
1) Improve software process
2) Improve software estimation
3) Improve project tracking

From the selected goals, questions were formulated as presented below.

Questions relating to goal number 1
• Does the effort estimation match the actual effort?
• What efforts are required for each activity in the process?
• How much time is required for each activity?
• Where in the process are errors created/discovered/corrected?
• How many requirements are added during the process?

Questions relating to goal number 2
• What is the estimated compared to the actual effort per activity?
• How complex is the software?
• What is the estimated compared to the actual effort/size/schedule per activity?
• What is the estimated compared to the actual skills among the staff?

Questions relating to goal number 3
• What is the status of each activity?
• What is the status of the project?
• What is the earned value per activity?
• What is the actual compared to the estimated time spent per activity?
• What is the actual schedule/effort/size compared to the estimated, per activity?
• How many faults does the product have?

From the questions, metrics were extracted as presented below.

Metrics for schedule evaluation
• Estimated compared to actual schedule / activity
• Elapsed time / activity
• Number of person hours to perform an activity
• Estimated compared to actual effort (PH) / activity
• Work effort, LOC / PH
• Earned value (EV) / activity

Metrics for requirements evaluation
• Number of changed or added requirements during process
• Changed requirements / activity

Metrics for project size evaluation
• Estimated compared to actual size of software
• Complexity of code in software
• Estimated LOC compared to LOC written

Metrics for software testing evaluation
• Number of performed tests
• Number of passed tests

Metrics for software quality evaluation
• Number of errors / LOC
• Number of detected errors / activity
• Average time elapsed between error detection and error correction

Metrics for project risks evaluation
• Overall percentage completed work
• Percentage of project time elapsed
• Software re-use

In the pre-study, a minor survey was also conducted at one of the leading software houses in Sweden, which showed that in this particular organization these metrics for software measuring were used;

• Function points
• LOC
• Requirements (number of pages)
• Use cases
• Screen shots/prototypes
• Sequence diagrams
• Number of classes (analysis)
• Number of classes (design code)
• Number of components
• Number of test cases
• Number of test plans
• Number of defects
• Activities per project plan
• Time unit per activity

Based on the research performed in the pre-study and in extensive literature investigation, the authors were able to create a set of metrics that should be appropriate for software process measuring. Inspired by the GQM method, the authors suggest the goals, questions and metrics for software process management and/or improvement, presented below.

Organizational goals
• Improve/manage project tracking
• Improve/manage product quality
• Improve/manage software process

Questions derived from the goals
• Does the effort estimation match the actual effort?
• Does the process follow the schedule?

• Does the product match the requirements?
• Does the product reach the desired quality level?
• Are the requirements accurate?
• Is the process/project well managed?

Metrics derived from the questions
• The number of hours worked in total
• The number of written LOC/PH
• The number of LOC written in total
• The number of requirements changed
• The number of tests performed
• The number of tests passed
• The number of errors/LOC
• Time elapsed to error detection
• Time elapsed to detected error is fixed

Some of the metrics selected are located on the ratio scale level, described in the beginning of chapter 4.1, because of its potential to apply all mathematical operations to the measures. This is also, as stated before, the most common form of measure scale and should therefore be adequate. These metrics are generated by dividing two integers. To create an easily observable picture of the process status, some metrics will be displayed in the percentage-form in the implementation of the metrics. In order to make the software management as thorough as possible, the metrics were selected to cover most of the evaluation areas described in chapter 4.1. The combination of metrics was derived from the evaluation areas schedule, requirements, software testing and software quality. The metric set also reflects the process-oriented view to minimize effort and schedule also described in chapter 4.1.1, by including such metrics as defect counts, calendar time and size measuring. This view on software management was in the earlier in this paper stated to be the most effective view for combining both customer and organization needs, and should therefore be appropriate to use for software process measurement.

In the beginning of chapter 4.1.1 it was stated that there are three possible categories of metrics; product, process and project, and that metrics may belong to multiple categories if they are intended to measure software quality. Product metrics were described as handling size, code complexity and performance, while process metrics handles improvement and maintenance, and project metrics handles cost, schedule and productivity. Since the suggested metrics include scheduling, productivity and product size, it belongs to both the product and the project category. Consequently, this should provide a basis for ensuring high quality in the software process management.
5.1 Implementing the Metrics

To examine the usefulness of the created set of metrics, the authors developed a prototype of a software management system, and implemented the metrics into the system. By testing the metrics in a realistic environment, their effectiveness in software management can be evaluated in a reasonable approach. Chapter six will describe the development of the management system and present a new tool for project administration.

6 Introducing a New Software Management Tool

The investigation of the approaches to software management described in this paper so far has been performed with the aim of developing an application prototype for project management. The set of metrics proposed in the previous chapter was implemented in a management system prototype, intended to provide a useful tool for software measuring and management.

To apply software metrics to a process, the project manager needs assisting tools. A number of methods and models proposals have been presented in the past, including [7, 19, 11]. A short description of existing models and tools will be presented in chapter 6.1.

6.1 Existing Tools for Software Management

There have been several attempts in the past to create metrics programs, for instance by the GTE Government Systems Operation, where a program for measuring, evaluating and reporting on a software process was developed [7]. The program consisted of a knowledge base for prediction of the project performance, support for metrics collection, and methods for monitoring at all management levels and quantifying results. The GTE has a software metrics program, instituted in 1986, which has declared following statement [7]:

“Consistent measures which show trends, assign earned value or identify activity completion shall be collected, analysed and applied to support project management and to improve the software process. Metrics on products and process shall be established and tracked throughout the software life cycle”.

The GTE GSC Software Metrics Program aims at providing a support for project performance prediction, by storing all collected experiences to benefit upcoming projects. The previously gathered experiences can help ensure more accurate estimates and schedules in the future. The authors of [7] concludes in their paper that a metrics program should be planned, monitored, reviewed, supported and improved, and that the metrics collected must be justified by both organizational and process goals. They also say that a metrics program should be flexible and adaptable to changes to the goals.

The authors of [19] have developed a knowledge-based software tool for management of software metrics, called the Software Quality Manager, SQM. The SQM was developed for two reasons: (1) effective management of software measurement knowledge, and (2) utilization of system domain knowledge to software measurement, ranging from requirement specifications to final source code. The authors believes that they have developed a systematic method for management of software metrics-related data, information and knowledge, which was implemented in a management tool. The author’s use of software metrics was according the following approach:

- **Low-level metrics** – numerically characterized attributes like the number of defects or cost-issues
- **High-level metrics** – symbolically characterized attributes like maintainability or quality
- **Measures** – “functions” of metrics, which can be used to assess or predict complex attributes, like cost or quality.

The division of metrics into these categories supposedly removes any possible vagueness by providing a consistent measurement environment.

Another technique frequently studied in the field is fuzzy logic, previously mentioned in chapters’ 3.1.1 and 3.1.2. The authors of [11] suggest application of fuzzy logic to software metric models, to come to terms with the difficulties in retrieving accurate values from vague data. According to the authors, the problem arises when project managers have difficulties in exactly specifying the values of the input metrics and instead must use their anticipations of the development success. This means that an accurate project prediction can be tough to outline at process initiation, when that planning is most critical [11]. The use of fuzzy logic variables would, presumably, serve as accurate and consistent measures at times when precise values are unknown or unclear, an idea that has been often debated in software metric literature. In [11] a case study was performed, investigating the usefulness of fuzzy logic compared to other techniques, such as neural networks or FPA (function point analysis). The results analysis gave that fuzzy logic showed a good performance, but was out-performed in terms of accuracy by the
neural network model. The authors conclude that fuzzy logic certainly has a place in the software metrics field.

6.2 Deciding on a Management Approach

The above description of existing management approaches includes the application of fuzzy logic, an area believed to have potential for generating accurate estimations of the software project success. However, the authors of this paper find the technique somewhat confusing and difficult to get acquainted with. Perhaps it can be useful in project management, but since the gathering of expert information may itself be a problem, one could only imagine the difficulties further introduced to this process when using a complex technique such as the fuzzy logic. Considering this fact, the authors of this paper have chosen not to use this approach for development of the new management tool. The software metrics program described in the previous chapter, developed by the GTE GSC, included a knowledge base for prediction of the project performance. The authors of this paper found this technique to be a particularly useful idea to apply in the new tool, and thus a knowledge base was implemented to provide the system with some rules on how to predict the project outcome with respect to the current project status.

Since this management tool proposal is a prototype to examine the usefulness of the above-suggested metric set, the authors decided that an automated collection of measurements is not necessary to perform the experiment. Currently, the project manager will manually feed the metric data to the system. This functionality may be further developed and automated in future work. In development of this management tool, the authors decided to select the .NET environment and the C# language, because of its possibilities to quickly create a full-feathered system. The prototype was created as a distributed client/server solution to represent a useful application suitable for real software projects, where the clients may be situated on several computers. A full description of the project management prototype is presented below.

6.3 Presenting a New Project Management Tool

The project management prototype is a client/server solution, developed using .NET techniques. The distributing of data is performed using a technique called remoting, which will be explained further down in this chapter. The application consists of a three-tier application, see figure 6, with client, server and a database used to store project data. A function call from the client goes through the server, which uses a class library (a dll-file containing all database operations), and the requested data-transaction is performed on the database.

![Figure 6 The three-tier structure of system](image)

The client application displays the current status of the project on start-up, giving the manager a quick overview of the progress. This status is displayed by retrieving both data from the original project plan set up by the manager in the beginning of the project, and current data entered by a developer or the manager. A project prediction is displayed based on the status of the project, which may serve as a useful tip to the manager. The prediction is created by the knowledge base of the system, which resides in a separate class in the client application.

The server is created as a console application and it passes the function calls sent from the client, which are executed by the class library to perform database operations. The function calls are passed using the remoting technique, which consists of proxies, channels and messages. A proxy is a local copy of the remote object, which contains the exact same methods and properties as the original. The proxy uses a channel object to pass function calls. A channel represents the connection to the remote application, and has a formatter object, which converts the method call into a message with a known format. The message is received by the server, which has a mirror image of the client-side channel listening for incoming requests. The channel formatter descrambles the message, using an object called a StackBuilder. This object converts the incoming message back into a method call and invokes the method on the remote object.

The database was created in Microsoft Access and it is stored in the same directory as the server. It is has two tables; ProjectPlan and Statistics. In the first table, the manager stores estimation project data. This table is used as a reference to compare and calculate the progress during the project life span. The second table of the database, Statistics, is used to store data from the on-going work and may be used to report data updates. The stored data is entered by either a developer in the
project or by the manager. The metrics used in the project management are those presented in chapter five, the number of hours worked, the number of written LOC/PH, the number of total LOC written, the number of requirements changed, the number of tests performed, the number of tests passed, the number of errors/LOC, time elapsed to error detection and time elapsed to detected error is fixed.

Screen shots of the system interfaces can be studied in Appendix.

### 6.3.1 ManagementClient.cs

This is the system main class for project data entering and viewing. It displays the project estimations made by the project manager, along with the most recently entered project data. This is shown in data grid controls at the centre of the screen. At the bottom right of the screen, the project status is shown as the percentage to which the current data is coherent with the estimation data. At the bottom left of the screen a prediction about the project progress is displayed, based on the over-all project status. At the right of the screen the percentage of the over-all project status is shown in a progress bar, together with the three metrics selected as the most critical for the project success; LOC written, hours spent and tests passed.

The project data is requested using a proxy to the server, retrieving the data from the database table Statistics.

### 6.3.2 DisplayData.cs

This class displays the project data in a data grid by making a remote call to the class library on the server, retrieving the data from the database table Statistics.

### 6.3.3 DisplayPlan.cs

This class has the same functionality as DisplayData.cs, except it retrieves the data saved as the project estimation plan from the ProjectPlan table in the database. To update the project plan, the user presses the button “Update plan”. This executes a method in the class library by a remote server call. The estimations previously stored in the database will be updated.

### 6.3.4 InsertData.cs

This class contains a data grid where the user can enter project data. When the form is created, the data grid is initialized and a data table is inserted. The table is created with the column names used as metric names in the project, and a data row is inserted. When the user has entered the data and presses button “Insert data”, it is stored in the database. It is also possible to update data already reported for one day, by pressing the button “Update data”. The database storing is performed by the class library. The client passed the data entered in the form to the library, where they are inserted into the database table Statistics. The client receives a indicator string, showing if the operation was a success or not, which is displayed in a messagebox in the client.
6.3.5 CreatePlan.cs

This class has the same functionality as InsertData.cs, except it saves the entries in the table ProjectPlan in the database. This data contain the estimations of project properties set up by the manager in the beginning of the project’s life-span.

6.3.6 Help.cs

This class is a form providing basic help about how to use the management tool. When the user clicks the links on the form, information about the topic selected is displayed in a textbox. The user can choose to get help about how to set up a new project, how to view project data or general information about the project management system. The information is contained in three text files which are located in the client directory; HelpCreateProject.txt, HelpViewProject.txt and HelpManagementSystem.txt.

6.3.7 KnowledgeBase.cs

On client application load, a reference to this class is created by the client, by which several project progress calculations are made. When the class is called by the client, the knowledge base constructor receives the current project data and the estimation data as data sets. The data sets are used to generate a project status list. This list is created by comparison of the current data for each metric, to the estimation data. For each metric, the percentage to which the current project data match the estimation value is calculated and added to an array list, called “statusList”. This is the list sent to the client. A second list containing the metric names used is also sent to the client main class, for display in the main client interface.

Next, a prediction of the project progress is generated. The client calls the knowledge base again, and passes the status list received. To calculate the over-all project status, the values of the list are added together and divided by the number of metrics used in the project measurement. There are six rules in the knowledge base. If the project status value is zero percent of the estimations, the prediction generated will say that there has not been any progress yet. If the status value is a 100 percent of the estimations the prediction will say that the project has been successfully executed according to the plan. If the status value lies between zero and 25 percent, the prediction will say that the progress will soon reach a fourth of the estimations. The other three rules are constructed according to the same principle, and generate predictions when the status value lies between 25 and 50 percent, 50 and 75 percent and 75 and 100 percent of the estimations.

6.3.8 ManagementServer.cs

This class passes the method calls made in the client application, to a class library where they are executed. The library contains all database operations necessary for the application. The communication between client and server goes through the HTTP protocol. The HTTP channel uses the SoapFormatter to convert method calls into the SOAP protocol (Simple Object Access Protocol), a web service messaging protocol. Both server and client must register a channel to establish endpoint, which is the location of the application on the network (IP-address and port number). This is performed in the statement: ChannelServices.RegisterChannel(channel). The channel registration also establishes a listener thread, which handles the remote calls. This is performed in the statement: RemotingConfiguration.RegisterWellKnownServiceType.

6.3.9 ManagementLibrary.cs

This class contains the database connection and all database operations used by the client. The data is passed to the class in the remote method-calls and the database is manipulated using SQL-statements. The class library uses the OleDbDataAdapter object to create the SQL-commands, and the data is returned as datasets and displayed in the main client interface. The methods of this class are:

- **SaveProjectPlan** – receives the data entered as project estimations, which is stored in the database table ProjectPlan. The method returns an indicator to the client, to show if the operation was successful or not.
- **UpdateProjectPlan** – receives the data entered as project estimations, which is updated in the database table ProjectPlan. The method returns a string to the client, indicating if the operation was successful.
- **UpdateProjectEntries** – receives the data entered during project execution, which is updated in the database table Statistics. The method returns a string to the client, indicating if the operation was successful.
- **ShowEstimations** – extracts all data containing the project estimations from database table ProjectPlan and returns it to the client as a dataset.
- **SaveEntries** – receives the data entered during project execution, which is inserted in the database table Statistics. The method returns a string to the
client, indicating if the operation was successful.

- **ShowEntries** – extracts all data containing the project statistics from database table *Statistics* and returns it to the client as a dataset.
- **ShowCurrentData** – extracts the most recently entered data from database table *Statistics* and returns it to the client as a dataset.

### 6.4 Analysis of the Suggested Management Tool

The suggested project management tool was created to provide a support in the supervision of software projects. The system is merely a prototype and there are endless possibilities for enhancement. Still, it should be adequate to provide assistance in software project management. The metrics selected and implemented in the system, could easily be changed to suit different organizational goals and reflect specific customer needs. The project prediction generated will most likely be replaced further on with more useful information, such as what resources should gain focus according to their status, and if the should be increased or decreased by the manager to be able to reach the project estimations.

### 7 Discussion

The project management tool was developed from an extensive research into the software management field, starting from scratch by describing benchmark process and project models, which were analysed. The authors then explored the area of measurement and measures, and took a deeper look at software measuring and its practises. From this research a new set of software metrics for measuring the software process was derived and tested in the suggested project management tool. It is the authors believe that the suggested tool should be able to provide an adequate help in supervision of a general software project. The tool is built using .NET techniques and can therefore also provide an approach that is up-to-date with modern software development approaches currently used in the field, unlike the existing project tools described in this degree project. Charts and graphs may very well be useful in decision making, by providing a quick over-view of the project status, but compared to the suggested management tool they have some lacks. While it is possible to have control over a project schedule with the Gantt chart, you cannot get a prediction from it, which the tool suggested in this paper offers. The CPM and the PERT methods have both been criticized of being vague and not providing good-enough tools for software management. The methods described have had a great influence on today’s approach to project management, even though perhaps not as frequently practised anymore. The suggested tool does not build on any of the described models because although they may be sufficient in theory, they were found to hard to apply in practise.

The extensive research of the software engineering field described in this paper has led the authors to understand that process and project management are two very different concepts; in project management the focus lies on determining if the project is a success or a failure, which is decided in terms of the effectiveness of the project. The goal of each software project is to successfully complete the assignment on time and within budget. If the project is poorly managed, the symptoms as described above in the chapter 2 may appear, with late deliveries, exceeded cost-estimations and other problems, that in the end affect the customer satisfaction.

The process management on the other hand concentrates on the maturity level reached in the developing process and its effect on the result of the project, i.e. the software. This means that the concepts of process maturity may not concern the project assessment at all. It is, however, as stated before suggested that project management involves choosing an appropriate process model [5], and a project evaluation should, according to Kan [14], include an assessment of the processes used in the project.

In chapter two, an analysis of the process models described, concluded that while the ISO 9000 standard have had some success assisting organizations in their process improvement, a high quality product is not a guaranteed result of this framework. The 9126 standard were developed to define quality attributes, and the SPICE standard was created particularly for software process improvement. Unfortunately, these models have not been practised enough to be able to certify success. The Capability Maturity Model was in the analysis established as a modest tool in process improvement, also because it is not widely practised in the field. The model was also accused of being hard to understand and of having a too strong focus on project management at the expense of product quality. The CMM may be appropriate to use by an unstructured organization that wishes to improve itself by reaching the lower levels in the model, but the authors have not found any documented success in reaching the higher levels. On the contrary, the higher levels of the model were found unclear and hard to apply in reality. The CMM was originally created for use in defence systems development, and therefore not optimised for software processes in general. The GQM method has generated some documented
success, possibly because of its flexibility, and it also distinguishes between organization goals and project concerns in a way that the other models fail to do. The model was selected as an inspiration in the creation of the set of metrics presented above.

The chapter on measurements and metrics gave a deeper look into project measuring, and served as a pre-study to the creation of the metrics set proposal. It contained a description of common measures and a further investigation of metrics applicable to a software process. In the next chapter a metrics set was proposed as appropriate measures to software process management, and these metrics were also implemented into a project management system prototype. The prototype is a fully operational system and may be used for managing smaller software project, however it needs some further adjustments to be adaptable to real projects.

8 Future work

This degree project has been a summary of existing techniques for software process management with the intention of gaining a greater understanding of the field. Further, the paper is an investigation of the metrics used in the field for software process measurement, resulting in a proposal of a new set of software metrics. Finally, the metrics were implemented in a project management system for, created to assist a project manager during process execution. The system is a prototype and will be developed further.

In a future version of the system, the manager will hopefully be able to adapt the metrics set and maybe even create new measures, to suit different processes and varying business needs. The project prediction will also be enhanced to generate more constructive advices based on the project status. A valuable addition to the system could, for instance, be indications of what metric values should have an increased focus, and by how much they should increase in order to maximize the progress and guarantee project success.

9 References


A Study on Software Management Approaches: Proposing a Project Support Tool


Appendix – System Screen Shots

The Client Main Interface

The Create Plan Interface
The View Plan Interface

The Report Progress Interface
The View Statistics Interface

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Description</th>
<th>HoursSpent</th>
<th>LOCPln</th>
<th>ReqChanges</th>
<th>LOCCommits</th>
<th>TestsPerWeek</th>
<th>TestsPassed</th>
<th>ErrorLOC</th>
<th>-browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2003-02-11</td>
<td>Project kick-off, with team</td>
<td>100</td>
<td>250</td>
<td>0</td>
<td>1234</td>
<td>15</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2003-02-20</td>
<td>More req. changes to system</td>
<td>300</td>
<td>250</td>
<td>0</td>
<td>1234</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2003-06-11</td>
<td>All tests passed, timely</td>
<td>250</td>
<td>250</td>
<td>0</td>
<td>1234</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2003-07-10</td>
<td>Some tests failed</td>
<td>250</td>
<td>150</td>
<td>0</td>
<td>1234</td>
<td>11</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2003-07-20</td>
<td>Test req. changes</td>
<td>200</td>
<td>150</td>
<td>0</td>
<td>1234</td>
<td>11</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2003-09-18</td>
<td>Several tests failed this week</td>
<td>350</td>
<td>150</td>
<td>0</td>
<td>1234</td>
<td>10</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2003-09-19</td>
<td>Test req. changes this week</td>
<td>150</td>
<td>100</td>
<td>2</td>
<td>1234</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2003-09-20</td>
<td>Error fix bug detected</td>
<td>120</td>
<td>50</td>
<td>0</td>
<td>452</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>2003-09-17</td>
<td>Tests passed timely</td>
<td>50</td>
<td>50</td>
<td>0</td>
<td>250</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>2003-09-11</td>
<td>Project start date</td>
<td>40</td>
<td>70</td>
<td>0</td>
<td>150</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
A Study on Software Management Approaches: Proposing a Project Support Tool

The Help Interface

![Help on the Project Manager](image)

<table>
<thead>
<tr>
<th>Help topics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting up a project</td>
<td>When starting up a new project, the project plan needs to be created. Select the menu-option &quot;Create new plan&quot; under &quot;Project/Project plan&quot;, in the ManagementClient application. This will open a form where you can enter the estimations you have made about the project properties. When you press the button &quot;Save plan&quot;, these metrics will be stored in the database. The data you entered is automatically displayed at start-up of the application, making it quick and easy to get an overview of the project properties. To view the project plan, chose the option &quot;Project/View project plan&quot;. This will open the form where you can view the project data you have entered. You can easily update the plan if the project requirements should change during the project execution. Open the form for viewing the project plan as described above and enter the data to be changed. Press the button Update plan.</td>
</tr>
<tr>
<td>Viewing project data</td>
<td></td>
</tr>
<tr>
<td>The project manager</td>
<td></td>
</tr>
</tbody>
</table>

Close Window

The Server Console Window

![Server Console Window](image)

Management server is running. Press enter to exit

Project Management library called...