Master Thesis
Software Engineering
Thesis no: MSE-2002-5
March 2002

Improving progress tracking using automated testing techniques

Henrik Bertilsson
Gustav Näsman

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Contact Information:
Authors:

Henrik Bertilsson
E-mail: h.bertilsson@home.se

Gustav Näsman
E-Mail: gustav.nasman@home.se

External advisor:
David Olsson
Ericsson Software Technology AB

University advisor:
Conny Johansson
Department of Software Engineering and Computer Science

Department of
Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

One of the problems when developing software is the difficulty of knowing how much that is done in a project. This is a basic condition to be able to estimate the remaining efforts with increased accuracy over time. To get accurate progress information it is important that the progress tracking is done in an *objective* way and also in an as simple way as possible.

Our focus in this work has been software projects in general but a case study has also been performed at Ericsson Software Technology AB in Ronneby. We have studied the organization and performed interviews with project managers to find out if this really is a problem and if the process can be improved in some way to ease the task of progress tracking. This case study together with our literature study has resulted in a proposition for a solution.

Our approach has been to use automated testing as a way to make progress tracking more objective. The result presented in this thesis is a base for a system that could ease the tracking of progress. In short, the system extracts information from an automated testing process and compares it with estimated figures to be able to automatically generate progress information.

Keywords: Progress tracking, automatic testing, project control.

ACKNOWLEDGEMENTS

We would like to thank all the people that have helped us with the work of this thesis. Special thanks goes to:

- Conny Johansson, our supervisor at Blekinge Institute of Technology.
- David Olsson, our supervisor at Ericsson Software Technology.
- The managers and testing-staff at Ericsson Software Technology for participating in the interviews and observations.

Table of contents

1 INTRODUCTION

This thesis has been written by Henrik Bertilsson and Gustav Näsman during the years of 2001 and 2002. We are both students of the master year at the software engineering program at Blekinge Institute of Technology.

The thesis contains the results from our investigations and analysis regarding the possibility to improve progress tracking by integrating progress tracking with automated testing. This applies to software development in general as well as in a case study of the Ericsson MPC department in Ronneby. From the beginning the topic for the thesis came from Ericsson and at that time it concerned the progress tracking and the related tools for that at the MPC department. Over time it evolved somewhat to especially cover the area of progress tracking integrated with automated testing.

In software development three things are usually considered to make a successful project: it shall be on schedule, within budget and according to requirements [Ben94]. Since we in this thesis are focusing on progress tracking and automated testing aspects we will cover the "on schedule" and the "according to requirements" points. Moreover, since these points are affecting whether or not projects will meet their budgets, this thesis can be of interest even in the "within budget"-viewpoint.

1.1 EPK historical background

What is now called EPK was originally a company called EP-Data, founded in 1981, partially owned by LM Ericsson and Programator. 1992 LM Ericsson became the single owner of EP-data, which lead to that EP-data became a subsidiary of LM Ericsson. In 1993, EP-Data was renamed to EP Consulting Group, which obviously meant that they focused more on pure consulting services. In 1995 the company was renamed to Ericsson Software Technology AB (in short EPK), which is its current name. In 1999, EPK got the responsibility for the production of the charging units that handles the billing.

Now EPK is a pure e-commerce development company focusing on mobile Internet, charging and positioning. EPK is located in Karlskrona, Ronneby, Malmö, Sundbyberg and Kalmar. Together the offices staff about 700 people.

The MPC-department, which is a part of EPK, started to investigate possible products in 1996 and had at that time only a handful of employees. In time the number of employees grew to about 30 and remained so for a couple of years. With the release of MPC 3.0 the size of the department virtually exploded when the need for employees rose. Prior to the release of MPC 3.0 there was two versions of the MPC/MPS-system and now the 4.0-version is in its final stage. The MPC 5.0-version, which also is under development, is in an early analysis- and designphase.

1.2 Products

Ericsson's complete positioning solution is called MPS (Mobile Positioning System). The purpose of the MPS is to provide a mobile network operator with the possibility to offer positioning services to its customers. Positioning is the process of determining the physical (geographical) location of a Mobile Station (MS). MS is a collective name for a Cellular phone or a Personal Digital Assistant (PDA). The MPS can for example be used to answer the question “at what position is a cellular phone right now?”.

There are four major categories of services that can be created with these applications [Pet01]:

- Safety; send help to me, for example 112/911.
- Information; where is the closest “something”, for example yellow pages.
- Tracking, for example fleet & people management (repairmen, friends & family etc.).
- Game; Virtual reality steps into your reality in the street, for example Virtual Paintball.

The MPS works as a gateway between the LCS Clients (Location Service Client, applications that are interested in positioning information), and the mobile network. The MPS is divided into two nodes, GMPC and SMPC, and is supposed to provide the LCS Clients with an interface to perform positioning without knowing exactly how the positioning gets done.

The MPC department is developing the gateway (the MPS-box in figure 1.1), and usually *not* the applications that make use of the gateway. The gateway is sold to other companies that use it to develop their own customized applications.

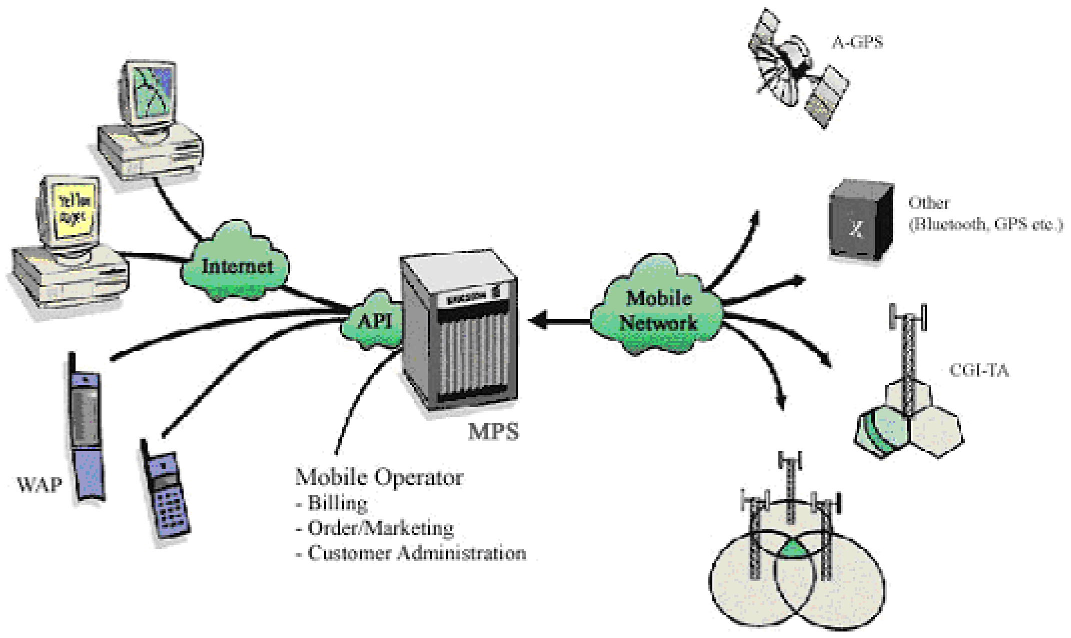


Figure 1.1. The Ericsson Mobile Positioning System (MPS) works as a gateway that provides applications with positioning information independently of the location technique used.

1.3 Hypothesis

Our belief is that the results presented in this thesis will support the following hypothesis:

“The tracking of progress can be improved by integrating progress tracking aspects into the automated testing process.”

1.4 Problem statements

The main problems we are looking at in this thesis are to achieve correct progress tracking and to make progress tracking more efficient.

1.4.1 Correct progress tracking

One difficulty in projects is to track the progress, that is, how much have you done and how much is left to do. This should be done correctly and preferably in a simple way.

If you as a developer do not have access to any good quantitative measures it can be easy or handy just to “guess” how much you have done on a component - that figure is just a *subjective* figure of the *actual progress*. Since it is a guess, the accuracy of this subjective guess is of course not always correct but it depends on different factors, such as the experience of the estimator and his knowledge in the area.

For example [How01]: Two project managers on similar projects are using the earned value concept to track the progress of a project. Both know that the earned value is calculated by the "budgeted time" multiplied by the "percent complete". However, the two project managers gets different results. How can this be the case? The basic problem is that "budgeted time" and "percent complete" are concepts that are hard to define and their value is hard to determine:

- Almost all projects have a budget that changes over time.
- Calculating the "percent complete"-value is perhaps even harder. There are different methods for doing this, referred to as "statusing methods". Most of these models should be applied for certain types of work and for certain situations. Many project managers do not know when to use which method, and also use the methods in an inconsistent way.

1.4.2 Effective progress tracking

It is a fact that the failure by the project management to realize that a project is falling behind schedule is one of the root causes for delayed projects [Pre01]. Therefore it is also hard for the managers to take appropriate actions in time to prohibit the delay. So a more efficient progress tracking process would most certainly help to avoid delays.

Today, the project manager or some other person might spend much time to be able to compile the progress figures for a project. This is a common problem, which we have noticed ourselves when doing projects. By introducing other methods or techniques the progress tracking could be faster, more correct and efficient. By other techniques we mean for example the use of automated testing techniques.

1.5 Why is this important?

It is important to know how much of a project that is completed. This is important to be able to know if you are on schedule, and if you are going to finish the project on time. It is a fact that many projects get delayed nowadays [Dus99, Sta94] and this is often not noticed until the end of the project, an example of this is the so called 90% syndrome which is explained in the next subsection. Our belief is that this can indicate that there might be something wrong with the way progress tracking is done.

We have ourselves noticed the difficulty to find good quantifiable metrics for progress tracking when conducting projects. Since it seems that the progress tracking issues can be a main factor in delayed projects we feel that this area is important and worth exploring.

1.5.1 The 90% syndrome

Software projects often have good high-level planning and control capabilities, but because of bad visibility at the low-level (individual level) those efforts at the high-level are more or less performed in vain. One common source of problems can be derived from an over-reliance in individual percent-complete estimates (subjective figures) as indicators of project progress at the low level [Boe81].

Figure 1.5 shows a typical pattern of percent-complete estimates on a software project which ran into such problems. The percent-complete figures are taken from a series of weekly progress reports submitted by a programmer working on a software component which eventually took 12 weeks to complete. By the end of week 5, the programmer estimated in his weekly progress report that the component was 90% complete (typically, this meant that the basic code had been developed and successfully compiled, and that there were just a couple of small bugs to find and one or two small additional features to implement before the program was finished). When the manager of this project saw the weekly progress report he would assign the programmer to a new project after the following week and schedule the component to be handed to the integration team at the same time. This lead to that when the component was not done after the following week the manager had to re-schedule as well as notify the integration team of the delay. This pattern often repeats itself for a couple of weeks and can lead to that people get bitter and don't trust each other. Further, the problems often spreads to other projects through the uncertainties of interface integration and the delays due to components who do not show up on time. This leads to that the entire project control process gets damaged [Boe81].

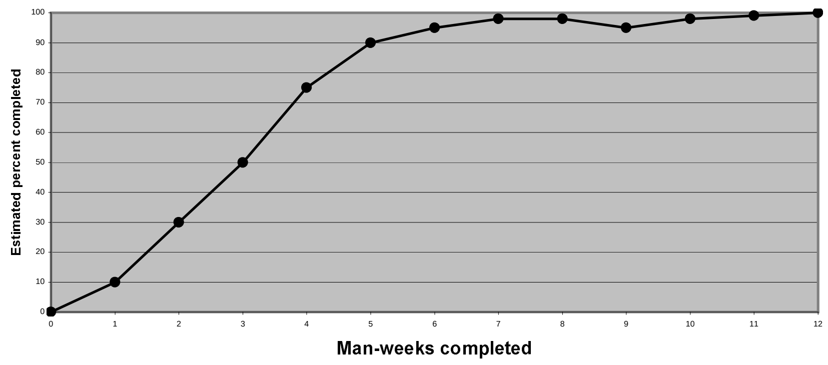


Figure 1.5. A graph showing the possible result of the 90% syndrome.

1.6 Scope

The scope of this thesis is an investigation of the possibility to integrate progress tracking and automated testing. We have chosen to specifically investigate if the progress tracking can be improved by using the test process results as input. This reasoning applies to software development in general as well as to a case study of the MPC department in Ronneby.

We are going to assess and analyze how organizations and in particular the MPC department could integrate some of the progress tracking activities into their testing environment. The focus of our work is the progress tracking during the *implementation part* of the execution phase in the PROPS development model used by Ericsson. It is important to state that this report only concerns the implementation- and test phases of the project cycle due to the fact that testing normally only is performed at those phases. Therefore our proposed solutions and ideas are mainly applicable to those areas of the development process.

The work procedures for the thesis project are defined in more detail in section 2, the method description.

1.7 Outline

This section contains a short description of each chapter in this thesis.

Chapter 1, Introduction

Contains an introduction to the thesis, our hypothesis and the problems that we have been trying to solve.

Chapter 2, Method

Contains information about *how* the work in this thesis was performed.

Chapter 3, Progress tracking

Contains general principles and common techniques regarding progress tracking and also some of the difficulties that you face when tracking progress.

Chapter 4, Testing

Describes general testing techniques as well as automated testing with its advantages and disadvantages. The testing process at the MPC department is also described.

Chapter 5, Integrating progress tracking and automated testing

Describes the advantages and disadvantages with the integration of progress tracking and automated testing.

Chapter 6, Proposed solution

Contains a description of our proposed solution, APTS. Includes what input is needed and what output that is possible to create, both in a general case but also at the MPC department in particular.

Chapter 7, Conclusion

Contains our conclusions after writing this thesis.

Chapter 8, Further work

Describes what further work that should be done to improve the result of this thesis.

Chapter 9, References

Contains a list of books and articles that were used when writing this thesis.

Chapter 10, Appendix A

Contains the results of the interviews performed at the MPC department.

1.8 Terms and abbreviations

Execution phase	The phase in the PROPS development model where the implementation is performed.
FOA	First Office Application. After a product is tested it is released to a customer where it is run for the first time in “sharp” mode.
GMPC	Gateway Mobile Positioning Center
LCS Client	Location Service Client
MPC	Mobile Positioning Center. This at the department at Ericsson Software Technology that develops positioning software.
MPS	Mobile Positioning System. The MPS is the Ericsson solution that provides location-based services. It mainly consists of two nodes, the SMPC and the GMPC.
MS	Mobile Station. MS is a collective name for a Cellular phone or a Personal Digital Assistant (PDA).
PROPS	A development model used and developed at Ericsson. Tollgates separate each phase in the model, which works as decision points for whether or not to continue a project.
SMPC	Serving Mobile Positioning Center
WBS	Work Breakdown Structure

1.9 Chapter summary

The main problems we are looking at in this thesis are to achieve correct progress tracking and to make progress tracking more efficient. One problem as we see it is that progress reporting figures can be too subjective, and thereby possibly affect a project in a negative way.

We will investigate if it is possible to improve progress tracking, both in the general case, but also at the MPC department in particular.

2 METHOD

This section describes how we performed our evaluation of the current testing and progress tracking process at the MPC department and how we gained further background knowledge about the domain in general.

Our literature search and study was the initial stage in our work process to be able to find the different hot spots in the current testing and progress techniques where improvements could be made. This was a crucial stage since our prior experience and knowledge in the area were somewhat limited. We performed a small case study at the MPC department to get information about their development process (naturally including progress tracking and testing). To obtain further knowledge, we got opinions from the testing- and managementstaff at the MPC department by performing interviews and participant observations as a small part of the case study. Since the amount of documentation regarding testing at the level of our interest at the MPC department were quite limited, the case study were of a more direct nature, but also partially indirect [Daw00].

Each step in our method is further described in the subsections below.

2.1 Literature study

The literature-search regarding automated testing, the testing process in general and project management continued throughout the entire project. Since there is a large amount of information available in these areas, and there were simply not enough time to read it all, an evaluation and prioritization of the material were performed. The selection process were done by reading abstract-sections of the articles, and if the article seemed interesting to us, a brief evaluation of the article were made to be able to prioritize its importance for our study. Those articles and books did form the base of our theoretical background.

While reading, it has been important to keep our hypothesis in mind to be able to identify key aspects in the documents, which could provide us with valuable input. Another crucial factor has been to read the selected documentation with a critical attitude so we could detect areas where improvements or enhancements could be proposed.

2.2 Interviews and participant observations

To be able to identify essential areas in the testing and progress tracking process where improvements could be made we had to gain further knowledge in the MPC departments development domain and culture. To get this deeper understanding of how the current development process and tools work we combined interviews and participant observations. By simply and solely perform one of the techniques we would not have got as much information as we needed since our focus was divided in two areas – testing and progress tracking.

The purpose of performing so called *in-depth interviews* can be defined as follows: “At the root of in-depth interviewing is an interest in understanding the experience of other people and the meaning they make of that experience.” [Sei98]. Other authors [Bec57] suggest that participant observations are the best way to get a valuable result. Some arguments against interviews are that they can take much time and be very expensive to perform. The best reason for using observations is that it is the superior alternative in this kind of investigation, where we shall get a view of how the testers, developers and managers work as well as to see how the test-tool is used in practice. It would be hard to get this understanding by performing regular interviews – participating makes the understanding easier. A few drawbacks with participant observations is that we as observers can be too subjective – we must remain objective, the collection of data can be quite unsystematic and the observation can affect the users normal behaviour.

Our observations did not focus so much on pure data-collection for statistical purposes. We were more interested in acquiring the participants' own opinions, ideas for improvements and also to get our own view of how development and progress tracking works today.

The conduction of participant observations was rather straightforward. Testers, developers and managers were our subjects for observation. We sat together with them to follow their work and log their actions and other observations. These observations were made when the time was right for the participants and us so it would show relevant results. That is – we observed them when they were doing relevant activities, for example progress reporting.

After the observations we analyzed our notes and general observations so findings could be detected.

2.3 Tool recommendations

We have examined the possibility to develop some kind of tool and made recommendations on how such tool could look like and which functions to include. We have used the acquired knowledge from the literature study, interviews and participant observations as a base for the proposed functionality contained in the tool.

2.4 Chapter summary

To gain knowledge about progress tracking and automated testing we have performed a literature study. We have also performed a small case study at the MPC department, where we have interviewed management and testing-staff, and also observed the way they work.

3 PROGRESS TRACKING

This section will cover the general principles and common techniques regarding progress tracking and other areas connected to that topic. The purpose of this section is to visualize some of the problems that managers are dealing with and also to describe some solutions that they use to be able to control software projects. We will also describe how the project tracking works at the MPC department as a small case study as a second part in this section.

3.1 General principles

The subject of progress tracking and project control is a sole management task to try to prevent and detect problems as early as possible in the development process [Ben94]. Naturally it is therefore the management group who has the responsibility to control the current project's progress and results, but the management shall also manage the estimations and scheduling issues. These tasks are closely connected as well as necessary to be able to control the project and hence a mean to ensure a successful project.

One of the classical and most common problems in the area of project management is that the management group is unaware of that there exist any problems at the time when they need to know about them. Often they realize it too late to be able to correct the problem and still reach the deadline. In the worst scenario this leads to an unfinished project.

It is common that bad communication is the single largest factor in delayed projects. One way to avoid this and other problems is to have good and *direct communication* between the development team and the management group [Ben94]. Good communication can both give much better estimations and more information to the management group hence provide much more reliable tracking data. The topic of inter-group communication is a topic that we will not cover further in this thesis, but it is worth mentioning as a good foundation to build upon when managing projects.

The coming sections will cover the main areas in the development process that are connected to and affect the progress tracking activity.

3.1.1 Progress tracking

Almost all projects have a limited time frame. Therefore it is necessary to decompose the project to clearly quantifiable activities to be able to estimate them accurately. Then the activities should be scheduled to be able to monitor the milestones and deadlines that are set within the project's lifetime. Progress tracking is thereafter used to follow-up the schedule to check if the project is on schedule or not. It is up to the *project manager* to keep it under control by regularly measure the progress of the project to avoid large and negative surprises as we mentioned earlier.

3.1.1.1 Work breakdown structure

It is impossible to successfully keep track of a whole project as one large object. Therefore it is necessary to decompose it into smaller activities, which in turn are much more easy to estimate and follow up. To decompose the project it is natural to use the common divide and conquer approach in several steps to finally get small enough *work packages*. This method is referred to as the *work breakdown structure*, in short, the *WBS* [Ben94].

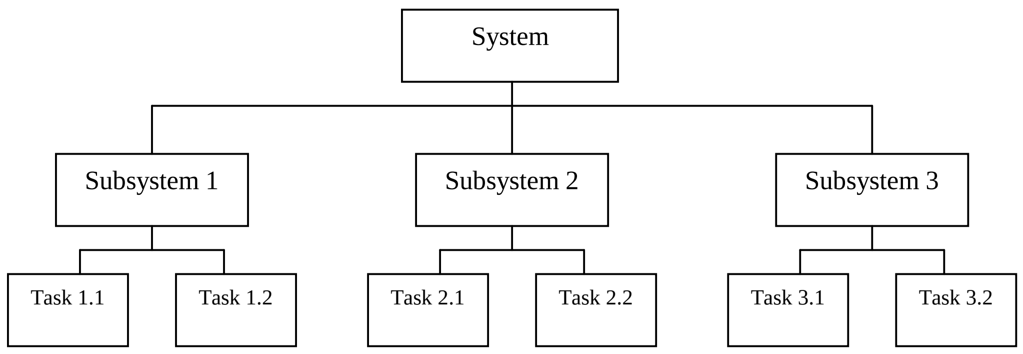


Figure 3.1: An example of a WBS-structure

The result from the WBS-work is a list of tasks containing all the activities to be performed in the project. All the activities in the list are assigned to persons who are responsible for the activity. Each activity also has a status-flag that indicate if it for example has started or is complete. It is important to state to the employees that all work performed in the project has to be part of any of the activities in the task list! This is to ensure that nobody does any “unnecessary” work, such as implementing extra functionality “which may be good to have in the future”.

At this point it is apparent that an updated WBS can be used to indicate the progress of a project. If an activity’s status is marked for example as “started” much longer than it should something is wrong and the delay of the activity is already a fact! This is a reason for why the activities should be kept small – then a delay is not as crucial as they otherwise could be. However, it is not sufficient with a complete WBS-structure. It is also necessary to estimate in detail and develop a schedule to be able to measure the progress in a more exact way.

3.1.1.2 Reporting techniques

There exist different levels of progress reporting. For example the developers report to the team leader, who report to the project manager, who in turn reports to his manager etc.

So, how can the project manager ensure that he gets the figures he needs from the developers to be able to compile proper progress reports? The first thing to do is to establish a regular flow of information from the developers. But, how accurate are those figures he gets? They are often very subjective and hence depend on how good the developer is at estimating on how much he has done and how much that is left to do in the current work-task. The manager must do everything possible to avoid the 90%-syndrome (sometimes called 90-50%-syndrome) as we mentioned earlier, or at any case be very observant for such patterns in the progress reports received from the developers [Ben94].

[Ben94] claims that informal talks between the manager and the developers are an excellent way to get good reports of how the progress really is. Often it is good to keep these conversations in an informal environment as well, so the manager’s office is not a very good place. However, this does not *ensure* that the quality of the figures is any better since they are still very subjective. The techniques to get the progress-figures of a project that we describe here are all in some way subjective. As we see it, this can be a potential problem when trying to assess the progress in projects and possibly an area for great improvement. It is this area that we are focusing on in this thesis.

Traditionally there are four more formal techniques for *acquiring* the progress figures [Ben94]:

- Periodic written status reports
- Verbal reports
- Status meetings
- Demos

Status reports

All project members should be obligated to report progress regularly to the project manager or team leader – without any exceptions. The reports are usually delivered on a weekly basis and should besides some natural data - like date, report period, name etc - at least contain these sections:

- Performed activities during the report period
- Planned activities for the next report period
- Problems

It is important that the time for the preparation of these reports do not take too much time, or else they will be considered to be too time-consuming to perform. According to [Ben94] it should take about 20-30 minutes to prepare a status report. The project manager then compiles all reports to a single one to present for the upper management.

Status meetings

Meetings are a good and necessary complement to the status reports. The meeting should take place once a week, to which the key project members participate. Each person (who often is a team-leader or similar) then summarizes the status of his responsibility area in the project. The important thing here is that all the participants shall be able to discuss problems that the others may have – this way all the experience available gets to good use. These discussions can then lead to solutions at the meeting or at least give hints so further discussions can begin after the actual status meeting.

Demos

The main drawback with product demonstrations is that they are very subjective in a special way, because they only demonstrate what the developer wants to been seen. Like we earlier have explained the information should be as objective as possible. Therefore it is a good idea to produce a proper demonstration plan or similar, which clearly describes what should be demonstrated by the developer.

Summary

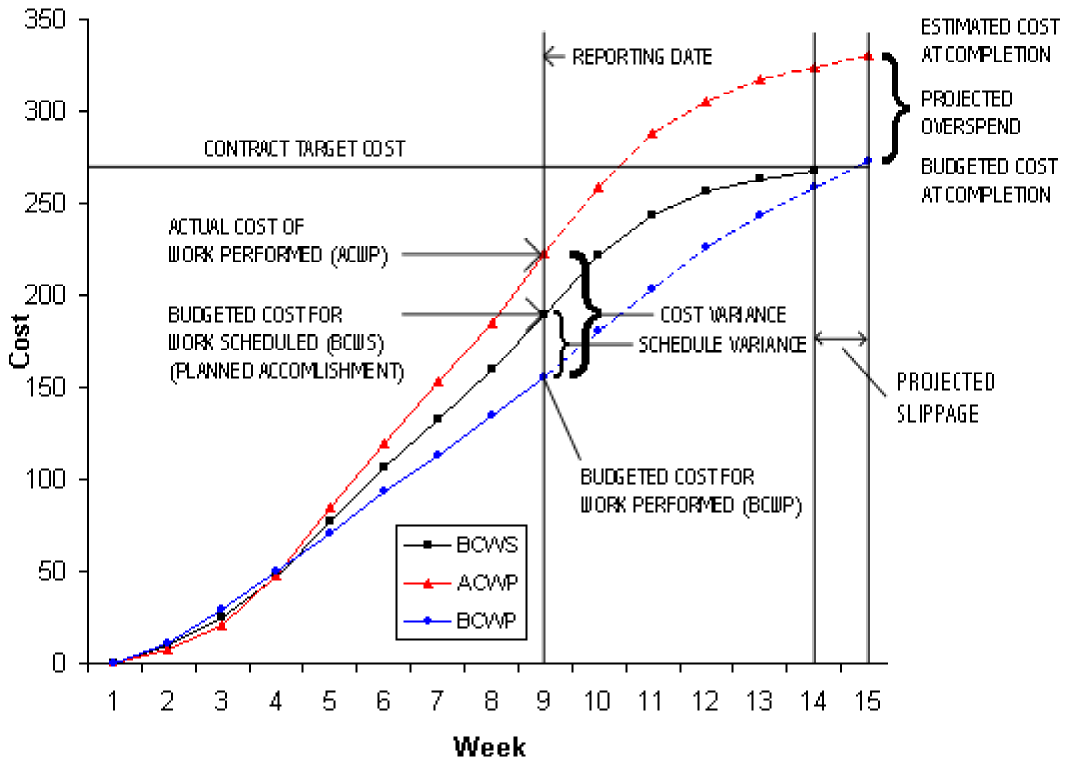
The techniques described above to acquire the progress data of a project can all in some way be too subjective. As we see it this can be a potential problem when trying to assess the progress in projects and possibly an area for great improvement. It is this area that we are focusing on in this thesis.

3.1.1.3 Earned value

As we earlier have stated there are methods for tracking the progress that is too subjective. Those methods provide the project manager with progress data that can be good, but not necessarily. It is desirable to use a better and more quantitative method, and such methods do exist – one is called earned value analysis [Pre01].

By using earned value analysis it is possible to assess the progress, status and the probable final cost of a large software project. A prerequisite to use the model is that each work package (task) must have an assigned value, this is called the task's earned value. That value shall include all overhead costs associated with that task so when all tasks are summarized the total cost of the project is visible. At the beginning of the project the accumulated earned value for the completed tasks in the project are plotted against time according to the schedule of when the different tasks should be completed. The curve that is generated shows the budgeted cost of work scheduled (BCWS) and can be used for performance measurement since it is against this curve the progress is assessed [Mcd93].

At regular intervals, for example in each month, the earned values for all *completed tasks* are summarized to give the budgeted cost for the work performed (BCWP). The actual costs for all the completed tasks are also collected from the local time-accounting system to get the actual cost of work performed (ACWP). Comparison between these values (BCWS, BCWP, ACWP) can then be used to give a valuable overview of the project's progress, predict overspends and show project slippage [Mcd93].



When all the figures have been collected it is possible to calculate and estimate different values that are useful for the project manager as progress indicators.

Some examples and explanations:

Budget at completion, BAC = $\sum(BCWS_k)$ for all tasks k

Schedule performance index, SPI = BCWP/BCWS

Schedule variance, SV = BCWP - BCWS

Percent scheduled for completion = BCWS/BAC

Percent complete = BCWP/BAC

Cost performance index, CPI = BCWP/ACWP

Cost variance, CV = BCWP/ACWP

- SPI is an indication of the efficiency that the project is using its available resources. If the SPI value is close to 1.0 it indicates that the project is quite efficient.
- The percent scheduled for completion shows how many percent of the project that should be done, while the percent complete shows how much that actually is complete at the current time.
- A CPI value close to 1.0 strongly indicates that the project is within the set budget and CV is an absolute value of how much the project is over or under the budget at the current time.

The values above indicate that earned value can give the project manager a way to foresee scheduling problems before they otherwise would be visible. Hence this is a good system to use if the project manager wants to take corrective actions before the serious problems actually occur [Ben94].

It is worth mentioning that when using this system it is good to keep the activities relatively small since it is only allowed (in the general case) to include *completed* tasks' earned value when calculating the progress. If the activities are too large the visible progress will be negatively affected due to the fact that they take much longer to complete and thus they cannot be included in the progress for some time.

3.2 Progress tracking at the MPC department

This section describes how the progress tracking process used at the MPC department works. We have also tried to identify some problems and areas for improvement in their process.

3.2.1 Estimation process

The MPC department do not use any algorithmic-based model or method, such as for example function points, when estimating their projects. Instead they compare components in the current project with components in previous projects that have been of similar size and complexity. By doing so they get an estimate, which hopefully is as good as it can be. This way of estimating is somewhat similar to the method called "estimating by analogy" in combination with "expert judgement" [Boe81]. The key thing with those methods is that it is the right people (the experts) that do the estimations and that there exists historical data from previous projects that can be used as a foundation for the new estimations.

So the estimation process at the MPC department is quite simple and works as the base for the schedule and thus also for the progress tracking.

3.2.2 Progress tracking method

During the interviews with managers and other personnel at the MPC department we got a view of how the progress tracking worked.

During the development of MPC 4.0 it was meant that they should use a progress reporting tool called CMISC. This was however not used since it was considered to be too complex and time consuming to be used regularly as a progress reporting tool, and since the AGRESSO-system is used for the salary system it would be twice the work to fill in both. One thing that the project manager specifically pointed out was that it must be easy to perform progress tracking or otherwise it will not be performed “at all”. Instead of the CMISC tool the manager got reports from the different teams every week on paper. The reports consisted of information from the development teams that the project manager then had to enter in digital form. The sometimes hard read reports usually were complemented with an oral report from the team leader.

However the CMISC-tool was used for the test reports from the different test-sections. The different test-reports were categorized on A, B or C-level, depending on their degree of affect on the systems functionality. In CMISC each developer could see all the current defects, the tester’s description of the problem and the current status of the defect.

Every month a report is compiled to the higher management in where the progress is visualized on a higher perspective on the entire project. Refer to appendix A to see an interview with a representative from the higher management.

3.2.3 Areas for improvement

Here we will shortly describe some areas in the progress tracking process at the MPC department that can be improved. The ideas for these improvements have been discovered from the case study, see appendix A for details. Note that this thesis concerns the progress tracking during the implementation and test phases, and hence our proposed changes will not solve all problems – but hopefully some of them!

1) The weekly reports that the manager gets by paper should be removed and replaced by an easy to use application (web or regular application) that can be used by the team leaders each week when the progress will be reported. All the time that the manager devotes on paperwork could and should be spent on other more important things in the project.
2) The correctness of the progress tracking could be improved. Since two out of three project managers think that there are flaws in the progress reporting this is an area for improvement [Ols02].
3) It is important for the entire staff to be aware of the status of the project. Currently it is only possible to view the *nightly build (see section 4.3.2.2)* results on a web page which is a really good thing to have for the developers. To get an understanding of the current situation in the entire project it would be good to have a similar thing for the progress tracking.
4) Valuable information like *earned value,* various graphs etc. that the higher management wants in their reports should be able to be calculated more easily and efficiently.

3.3 Chapter summary

Sometimes project managers discover problems too late to be able to fully correct the defects in time. Progress tracking is an important management activity that is used to detect these problems at an earlier point in the development process so they can be corrected before they cause more damage. To be able to track the progress it is necessary to decompose the project into smaller work packages, a work breakdown structure, which in turn are easier to keep track of. There are several ways for the developers to report the progress to the management. This can be done by written reports, meetings, demos etc. The common drawback with these traditional methods is that they can be too subjective. However there are methods that can help to prohibit this, for example earned value analysis.

4 TESTING

In this section we discuss classical test issues such as defect classification, different levels of testing etc. A short introduction to automated testing, the advantages and things to think about when automating testing is also discussed. Finally, we describe how the testing process at the MPC department works.

4.1 Classical test issues

It should be noted that the objective of software testing is to “execute a program with the intent of finding defects”. This can be compared with the intention to “execute a program to show that it does what it is supposed to do”. However, the latter description is an impossible task, since testing can only show the presence of defects, not their absence [Mar91]. Therefore, the objective of testing must be changed from an absolute proof to a “suitably convincing” demonstration. What is meant with suitably convincing depends on the context, for example if it concerns a computer game or a nuclear reactor [Bei84].

4.1.1 Defect classification

A “bug” or defect in software can mean many things, such as a mistake in interpreting a requirement or a syntax error in the code etc.

According to IEEE Standard 729, the following terms should be used:

- *Error:* When a human makes a mistake when doing some software activity. For example, a designer might misunderstand a requirement and create a faulty design. A single error can generate several faults.
- *Fault:* This is the result of the error.
- *Failure:* When the system behaves as it is not supposed to.

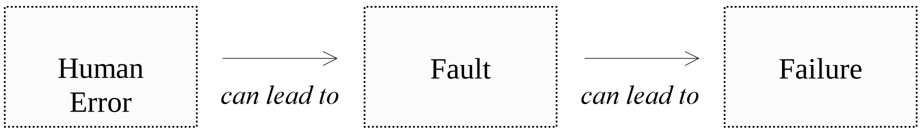


Figure 4.1. The relationship between errors, faults and failures.

A fault can be seen from an inside view of the system, as seen by the developers. A failure on the other hand takes an outside view, a problem that the user sees. Not every fault corresponds to a failure. The fault might be in code that is never executed or a particular state that is never entered. Then the fault will never cause the software to fail [Pfl01].

In this thesis, we will refer to errors, faults and failures as *defects* to simplify things.

4.1.2 The V-model

The V-model is a variation of the waterfall-model, which demonstrates how the testing activities are related to analysis and design [Pfl01]. Each development activity has a corresponding testing activity [Few99].

The product of each integration phase is tested with test cases and the test data derived from the corresponding design or specification base line on the left hand side of the diagram [Mcd93].

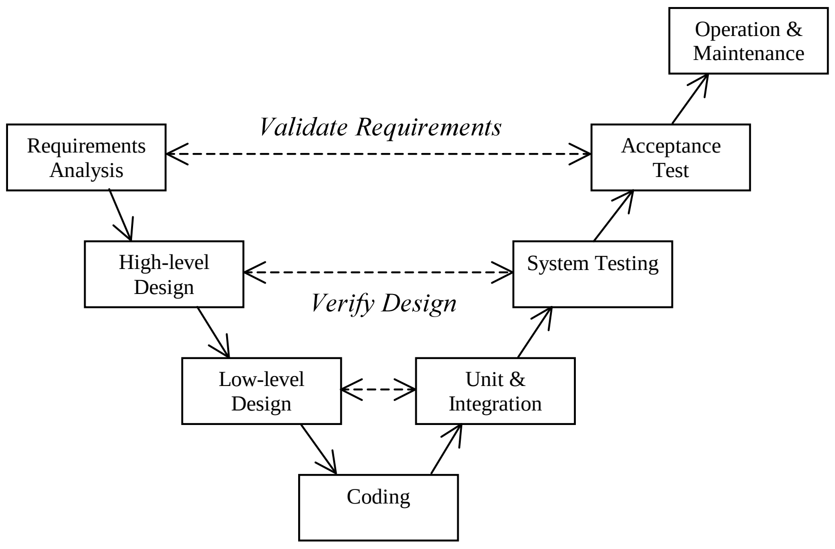


Figure 4.2. Example of a V-model diagram.

The reason for describing the V-model is that it shows where in the testing process Basic Test is applied (see section 4.3.2.1).

4.1.3 Levels of testing

To be able to discover as many defects as possible, testing is often performed at three different levels: unit, module and system.

4.1.3.1 Unit testing

The lowest level of testing is of a unit. A unit is a small part of the system, for example a class or even a method. The unit testing is often performed by the programmer him/herself, and should usually be carried out as soon as possible after the unit has been coded. This is one of the few occasions when a programmer tests his/her own code; at the higher levels the testing is often conducted by a special test-team.

Since some units rely on others, and not all units are ready at the same time, stubs are often used. A stub consists of skeleton-code and is only used to be able to test the unit that is undergoing testing.

4.1.3.2 Integration testing

Integration is the phase when the units are put together to form modules. Now you will see how the units interact, and if the interfaces between the units has been properly designed and implemented. This will most probably help you in finding defects that were not found using stubs.

4.1.3.3 System testing

System testing is aiming for testing the functionality of the complete system. This kind of testing is often done with the help of a test specification, which states how to carry out the tests. Test specifications are often also used when doing integration testing.

4.1.3.4 Regression testing

Regression testing, that is re-testing of a product after changes has been made to the code or to the environment, is an important part of the test process. This kind of testing is a highly repetitive task, and can therefore be seen as too monotone to be run manually. Therefore it is important (almost necessary) that you try to automate as much of it as possible, otherwise the regression testing will simply be skipped in many cases.

There are some difficult tradeoffs to do regarding regression testing:

- The time to spend on testing of new code versus the time of testing old functionality.
- Should a test be automated or should it be kept manual?

There can also be problems with the reproduction of tests, therefore it is important that you document how you managed to find the defect, what inputs that were used etc., otherwise it may simply be impossible to reproduce the defect.

4.2 Automated testing

This section contains an introduction to automated testing. Things such as what automated testing is and the benefits and drawbacks with automated testing are discussed.

4.2.1 What is automated testing

During unit and integration testing, there is often a large amount of routine-work involved. This might include creating programs used to provide a test case with test data, or to monitor the result of a test and compare it with the expected output. Much of the work in this process is highly repetitive and therefore a candidate for automation.

Automated testing often includes some kind of tool for simulating an environment for running tests. They often provide a special language for specifying tests, and additionally to describe the expected outcome of a test, and thereby saving a programmer resources when it comes to examining test outcomes [Mcd93].

According to [Dus99], automated testing can be defined like this: "The management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated test tool".

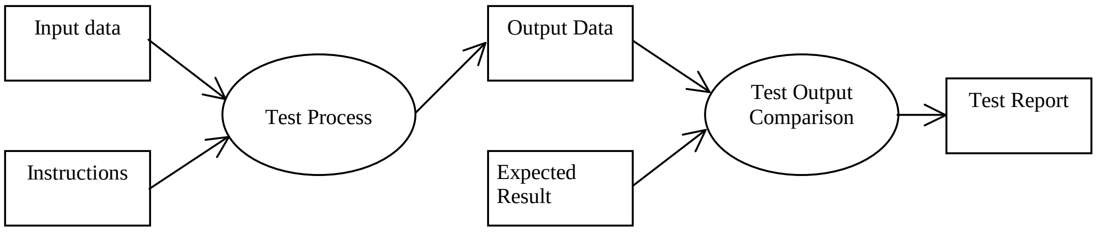


Figure 4.3. A typical flow in an automated test process.

4.2.2 Benefits of test automation

As the software engineering industry wants better quality products within shrinking budgets, companies turn to new methods to minimize the schedule and effort of projects. Software testing is one of these areas, and automated testing is supposed to be the “silver bullet” to solve the testing problems.

Manual testing is considered a labor-intensive and error-prone task, and it does not provide the same amount of quality checks that are possible through the (correct) use of an automated test tool. However, manual testing will not be replaced entirely, because some tests simply cannot be automated, such as verifying the printout of a document [Few99, Dus99].

Below is a summarization of the benefits that a company can draw from introducing automated testing. The three main benefits are [Dus99]:

- Reduction of the test effort and minimization of the schedule.
- The production of a more reliable software system.
- Improvement of the quality of the test effort.

In summary, more thorough testing can be achieved with less effort, giving increases to both quality and productivity [Few99].

4.2.2.1 Reduction of test effort and minimization of schedule

To get a product to the market as early as possible may mean the difference between success or catastrophy, which in turn might lead to company survival or death [Dus99]. Therefore, one of the main arguments for introducing automated testing is the assumed time reduction. This will lead to the saving of resources, or the possibility to execute a larger number of tests during a given time, which in turn leads to greater confidence in the system. According to [Few99], a timesaving of 80% is possible. However, the introduction of automated testing into an organization can lead to high costs in the beginning, see 4.2.3 for more information.

4.2.2.2 Production of a more reliable software system

Automated testing gives you the possibility to run tests faster and therefore more often compared with manual testing. By running tests more often you introduce a new and higher level of quality in the product, because you know all the time that the features work [Few99]. This is especially important when changes have been made to the software, and regression testing needs to be performed. The effort of running regression tests should be kept minimal, partly due to the fact that this is a repetitive task, and can therefore be seen as too monotone to be run manually. The focus on the testing can easily turn to the new functionality instead, although the old functionality can be equally or even more important.

Another problem with manual testing is that it can simply be impossible to conduct certain types of tests manually. Performance- and load/stress testing are areas that can be greatly improved by introducing automated testing. For example, simulating 200 users of a system would require huge manual resources while it can be done much more easily using automated methods [Few99].

4.2.2.3 Improvement of the quality of the test effort

Many testing tasks are considered boring. If it is possible to automate such a task, for example testing 100 combinations of input, this is a great advantage. The use of automated tests will also ensure that the input to the test case is exactly the same time after time, something that cannot be guaranteed when doing manual testing [Few99]. The use of automated testing also simplifies the reproduction of defects. It might be the case when using manual testing that the defect cannot be reproduced simply because the tester can't remember exactly how he/she discovered the defect.

Better use of resources

If the test cases can be run automatically, the test personnel will probably get more time to other more important things. This might lead to greater accuracy, higher staff morale and it also frees time to let the testers develop better tests [Few99]. This will also make the testers more motivated because they can be put into more challenging tasks than to just run the same test case manually over and over again.

It is not only the human resources that can be better used when introducing automated testing. It is also possible to make use of idle computer time. For example, automated testing enables you to run tests at night when the computers would otherwise be in idle mode [Few99].

4.2.3 Things to think about when automating testing

Before introducing automating testing into a project or company, several considerations must be done to whether or not the automation will pay off. It is important to state that not everybody earns time and money by automating the test process. In this section, some of the things that you should consider before introducing automated testing are discussed.

4.2.3.1 Doesn't replace manual testing

Not all tests should be automated – only the best ones to automate and the ones that are re-run most often. Some tests will simply be easier to keep manual, or the cost of automating the test will be so high that it is not economic to do so. Tests that shouldn't be automated might include [Few99]:

- Tests that are rarely run.
- Tests that are run on software that changes much and often, and thereby also forcing the test cases to be changed a lot.
- Tests that require human verification, for example verification of a color schema.
- Tests that require human interaction, for example swiping a card through a card reader etc.

4.2.3.2 Effectiveness

James Back reported in his (extensive) experience that automated tests found only 15% of the defects while manual testing found 85%" [Bac97]. This statement is based on the fact that every test that is automated needs to be tested to make sure it is correct. Testing the test case is done by running it manually. If the software that is being tested contains defects, they will be revealed here, when the test is run manually. When the automated tests are run later (regression testing), they have already been run before, and therefore there is less chance to find defects.

4.2.3.3 Cost

One of the biggest reasons for introducing automated testing in a project is the supposed save of time. However, companies must be aware of the fact that the introduction of automated testing doesn't always pay off immediately, due to the high start-up cost. The high start-up cost depends on that the introduction of automated testing requires a careful analysis of the software process to be able to determine which parts that could be automated. It is rather so that the time spent on testing will be increased in the beginning, since there is a learning curve associated with the introduction of a new testing tool [Dus99] and a new level of complexity in the test process, which makes it more time consuming. While the learning process is going on, it is most likely that the manual tests will be run in parallel with the automated testing to ensure quality, and the cost of the test effort will obviously increase.

4.2.3.4 Quality of the test process

Before even thinking about automated testing, you must make sure that the testing is good enough from the beginning. Trying to use an automated test tool without a proper testing process in place, will most certainly result in an ad hoc, non-repeatable, non-measurable test program. An ad hoc implementation of automatic testing can increase the cost compared with manual testing with 125-150% [Dus99].

"If the existing test cases are bad in finding defects, there is no use in automating them. Automating chaos just gives faster chaos" [Few99].

4.2.4 Automated comparison

To be able to verify that the output of a software program is correct (or not correct), comparisons must be made between the actual result and the expected outcome. This is called test verification. Expected outcome is the output when a program behaves in a correct way.

Is it possible to automate this process, and what can you earn by doing so? According to [Few99], automating the comparison process is the most "automatable" task in software testing, and most often the most beneficial to automate. If you don't automate the comparison-part of the automated testing, you have only gotten about halfway towards a totally automated testing environment.

Humans are not very good at comparing things in an exact way, for example lists of numbers, screen outputs or data of any kind. Therefore it is easy to make mistakes, and because it is a repetitive task most humans find it boring, which makes it even more likely that errors will be made. To compare data of different kind is exactly what a computer is good at. Besides, a computer never gets bored!

Automated testing might and often will generate a lot of outcomes. Most of these outcomes must be verified in detail in some way. However, in some cases only the execution of test cases are automated and the comparison is still done manually (or vice versa). This of course depends on the nature of the test case. However, the conclusion is that the potential of automated comparison is very high. Automated comparison is quicker and far more reliable than manual comparison.

4.3 Testing at EPK

This section describes the testing process used at the MPC department. Things such as at which levels testing is performed, who is performing the testing, and what parts in the test process that are automated today are discussed.

4.3.1 Development

The size of a typical project at the MPC department is usually rather large. The number of developers in such a project has in the past been about 15-20 people, and the duration about 8-10 months (when counting low). The size of a software system might be several hundred thousands lines of code. However, experience has shown that those projects tend to be very complex and hard to handle for different reasons, for example it is hard to actually finish such projects. Therefore, a reduction of the scope of the projects will be made, which in turn means that the number of people in a project will decrease (the number of developers in a project might be about 10 people). The concept of more and smaller projects will hopefully make the development more effective.

The development is mostly performed using C++, but the Graphical User Interfaces is coded in Java, HTML and JavaScript. The GUIs is used for the administration of the SMPC and GMPC.

The MPS is divided into nodes that in turn are divided into several different components as shown in the figure below. Each component is then assigned to a developer who has the responsibility to develop and to some extent also test that component.

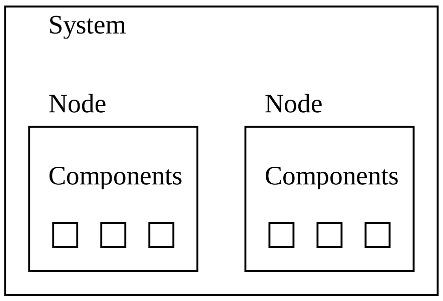


Figure 4.4: High level system-structure.

4.3.1.1 Additional quality-improvement activities

To achieve as high quality as possible other quality-improvement activities are used as a complement to testing. Code review is such a technique and has earlier been used at the MPC department, but the experience of this is not very good since it:

- Takes too much time for the attendants to prepare
- Is hard to get all the right people together at the same time

However, code reviews can be a good way for new or inexperienced developers to gain much knowledge about the system.

Some commercial testing tools are also used to detect different types of defects in the code. Purify by Rational, which is used mostly to find memory leaks, is the most frequently used tool. Quantify and PureCoverage, also by Rational, are also used to some extent.

4.3.2 Levels of testing

The testing is divided into three main levels plus one extra step, which is the first test of a product at an actual customers system. Each level can be assigned several names; therefore it may be some confusion about separating them from each other.

1. Basic Test. See 4.3.2.1 for more information.
2. Function Test. The testing of two or more components together, sometimes with the use of simulators. This testing activity is supposed to be performed by the “design-project”, but for the moment the same persons that do the system test perform this activity.
3. System Test. This is the testing of the entire system using real telecom nodes. Ericsson has their own test network to be able to test the products. For this, a special test-team is assigned.
4. FOA. This is the very first time the system is installed and run at an actual customer’s system.

4.3.2.1 Basic Test

Basic Test can be compared with the concept of unit test (4.1.3.1), but at component level. The person that develops a component in the system usually tests his/her own component.

Basic Test makes use of a component (developed by EPK) called AutoTest or in short, TC. With this component it is possible to test other components in the system, although it is not possible to test different components together. The TC can be used to test the functional requirements but also for load testing.

Very simplified, the Basic Test process works like this:

1. Test cases and test scripts are created for the actual component.
2. The component is implemented.
3. The test scripts are used as input to the AutoTest (TC) tool to test the component.
4. AutoTest produces output that may be used to verify if the test passed or not.

General

The general idea for TC is to ease the testing of components. This is done by providing a tool that makes it possible to collect and run a set of test cases. The collection of test cases makes it possible to reuse and automatically run this set of tests; that is it can be used for regression testing, but also as a help for system testers.

The automatic execution of a set of test cases will hopefully make the developer more motivated to run tests even if he/she is under time pressure. If the developer knows that a set of tests will test a specific new feature, this will also allow for other tests within the same set to be run (thereby also test other functionality).

This approach to testing can be seen as positive from two aspects:

- The developer knows what parts of a component that really works.
- This will give a way to define development progress to project managers.

The idea of using this for progress tracking requires that the test cases are designed and implemented before the actual software.

Applicability

The use of the concept of TC can be started already in the analysis-phase of a project. Not that any tests are executed at this time, but it is important that test-issues are considered already prior to the start of the implementation.

In the implementation-phase of a project, the TC is used extensively. At this time, the TC is used both to verify that new features are implemented correct, as well as to re-test old functionality. When testing moves on to the system testing-phase, TC is used for regression testing, since it then might be changes in the code due to bug-fixes.

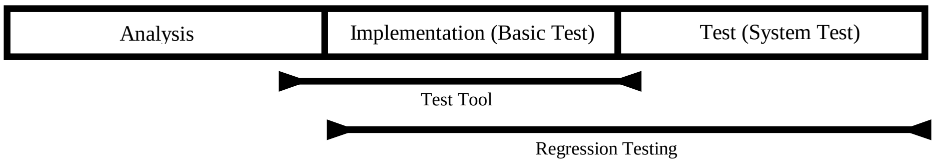


Figure 4.5. Schematic overview of where in the development process the TC should be used

Functionality

The purpose of the TC is to test components towards other components. The TC reads test scripts as input and executes the information in the scripts to test functionality in the tested component.

The TC is sequential, that is it tests one component after another and doesn't support concurrent testing of several components. Therefore, in its current shape, the TC isn't suitable for system testing.

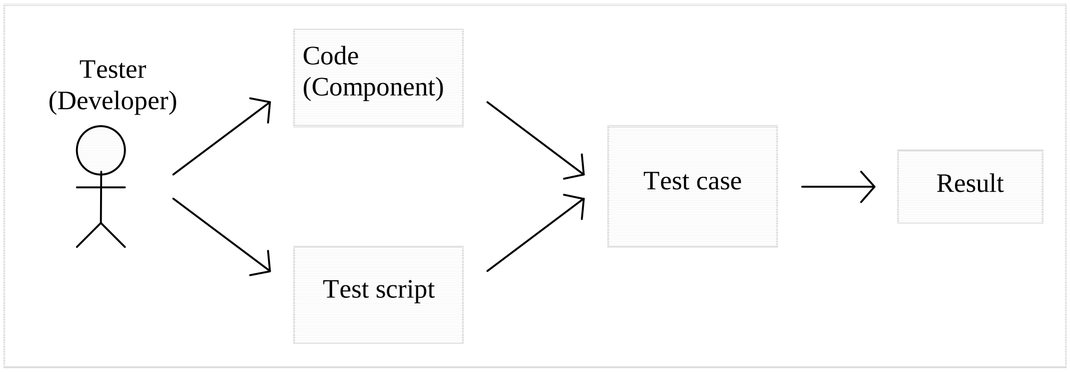


Figure 4.6. Basic Test. The developer writes the components and its corresponding test scripts. The test case is executed and produces a result that indicates the status of the test case.

Log files (Output)

The result of the test case is stored in files with time-stamps. The information in these files could be used for statistic calculations and to monitor test activities.

Today, the content of these files are mailed to the "owner" of the test case after it has been run (for further information about the log files see 6.3.2).

4.3.2.2 Nightly Build

Each night all components are included in a process called Nightly Build. This is a process that automatically compiles and builds all components. If there are any problems in some component this is visualized on a summary web-page so it is easy to see where the problems are and who is responsible for the component containing the defects. The Nightly Build is a very time consuming task that takes several hours to complete. Note that this process doesn’t include any testing, but only compiles the components.

4.3.3 Conclusions

The conclusions we can draw from the testing process at the MPC department after performing our case study and interviews (see Appendix A) are as described in the following sections.

4.3.3.1 Problems

- The organization of the testing is rather inefficient at the levels below the System Test. Test programs are often rewritten to suite new components. It should be possible to increase the effectiveness quite much.
- The incremental development can in some cases lead to problems. The different teams (developers vs. testers) do not always have the same picture of what the different increments shall contain. Therefore the testers may get problems since they sometimes think that they get a *complete* work package, when it is in fact considered *not complete* by the developers. This affects the system testers the most, since they don’t know exactly what to test in a certain increment.

4.3.3.2 Areas for improvement

- Design the test cases before the development start. By doing so, you will achieve a higher understanding of what functionality the component is really supposed to include, and how the component should work.
- A list of the things that shall be tested should be compiled - before the development starts. This will help to avoid the problems when developers insert functions that are not necessary, but only “might” be needed later on. Since it is not on the list of tests it is not likely that they would insert any unnecessary code. This can be compared with one part of eXtreme Programming (XP) which says that you should always keep the design as simple as possible to solve the current problem.
- The Nightly Build procedure should include the Daily Test so that all test cases also would be run over night. However, since the build-process takes so much time it may be a need for performance improvement in that area before any additional tests can be included.
- To assure good regression testing all old test cases should be saved in some sort of test tool so they could be run again at any time. Otherwise it is easy for developers to simply remove a test case that they think is unnecessary. The test environment partially supports this today since the developers create test-scripts for the basic test procedure. Those scripts can be saved to be run again at a later time, but the way this is done could be improved.

4.4 Chapter summary

Testing is usually performed at different levels: unit, integration and system. The V-model shows the different levels of development and their corresponding testing activities. Another testing concept is regression testing, which is the re-testing of “old” functionality after changes has been made in the code.

To improve the way testing is done, organizations often turn to automated testing. Automated testing can be defined like this [Dus99]: “The management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated test tool”.

The main benefits an organization can draw from automated testing are:

- Reduction of the test effort and minimization of the schedule.
- The production of a more reliable software system.
- Improvement of the quality of the test effort.

Some things to think about before automating testing are:

- Manual testing can’t always be completely replaced by manual testing.
- In some cases manual testing can be more effective than automated testing.
- The introduction of automated testing often means high start-up costs.
- The existing testing process must be well developed before introducing automated testing.

Test verification is the process of showing that the software works as expected. One way of performing test verification is by using automated comparison. The potential of automated comparison is very high. Automated comparison is quicker and far more reliable than manual comparison.

To summarize, when using automated testing more thorough testing can be achieved with less effort, giving increases to both quality and productivity.

The MPC department uses a process called Basic Test to test software at component level. Very simplified the Basic Test process works like this:

1. Test cases and test scripts are created for the actual component.
2. The component is implemented.
3. The test scripts are used as input to the AutoTest (TC) tool to test the component.
4. AutoTest produces output that may be used to verify if the test passed or not.

5 INTEGRATING PROGRESS TRACKING AND AUTOMATED TESTING

This section covers the advantages and disadvantages with the integration of progress tracking and automated testing. The next chapter (chapter 6, proposed solution) describes how to practically implement the integration in an organization.

5.1 Introduction

The problem that we have stated earlier in this thesis is that the tracking of progress in software projects is often done in some subjective way. The reason for using this type of measurements is probably because it is much easier to get hold of this kind of information. However, it is really no meaning of gathering information that you know may be incorrect since this can affect the project in a bad way, for example a delay or bad atmosphere within the project. According to our own experience it can be both hard and boring to come up with the figures as a developer for the regular occurring progress reporting occasions. It can be especially hard to come up with usable and precise quantitative measures. Instead it is common to say one of these subjective figures that might be completely wrong.

Our approach to solve this problem has been to investigate the possibility to integrate progress tracking and automated testing. This is to us the most intuitive way to really measure the progress in a correct way since it is truly objective. It is only through a well defined test that you really can say that a component is complete [Vig94]!

5.2 Advantages

This section describes the advantages that can be gained from the integration of automated testing and progress tracking.

5.2.1 Correctness

Are the automatically generated figures actually correct and shows the actual progress? In other words are the number of test cases that have passed testing a good measure of the progress or not? According to [Vig94] this really *is* a concrete and objective way to do this.

For example if you have a component with five requirements and one test case per requirement in a test suite like in the table 5-1, the progress could be calculated in a few different ways.

Test case	Estimated development time (h)	Real time (h)
A (tests req. a)	10	30
B (tests req. b)	10	5
C (tests req. c)	15	20
D (tests req. d)	20	15
E (tests req. e)	20	20

Table 5-1. Estimated development time represents the estimated time to develop the corresponding requirement (one test case tests one requirement). Real time is the actual time that has been spent so far on implementing the corresponding requirement.

If the A and B test cases in table 5-1 were complete and has passed testing the developer might report that he has completed 2 out of 5 tests (or requirements) (40%). This would give the erroneous indication that 40% percent of the component is complete. However, if you consider the estimated time figures you see that the actual progress only is 27% (earned value). Our point is that by integrating the progress- and testingprocess the *correct* progress figures can be gathered automatically! It is important to state that for this to work, estimations have to be made per requirement (or by test case).

The problem when doing this manually, is that every time a test case passes, you would have to edit some kind of document to show that the test case has passed (and you can thereby count the progress for it). However, since software often changes a lot and regression testing is performed, it isn't sure that this test case passes the next time the test case is performed. Then you would have to edit the document again and mark the test case as incomplete. The benefit by doing this automatically is that this procedure can be taken care of without any human interaction.

5.2.2 Time-saving

In software projects of today the managers and developers are forced to produce more in shorter time and at less cost than ever before [Dus99]. Like all the different parts of software projects the *management* also should try to find new methods and ways of working to make it as efficient as possible. According to our experience much of the manager's time is consumed on progress tracking activities, hence this is an area where improvements probably could be made.

There are several aspects of the timesaving issue regarding the integration between testing and progress tracking. First we have from the developers' point of view. They do not have to spend valuable time on activities that really are management related, for example checking which requirements are complete and writing reports etc. According to our experience they should devote their time at what they do best – namely to *develop software* and spend as little time as possible on management activities.

When discussing the managers there are different perspectives that must be taken into consideration. Different kind of managers wants different level of detail in the progress reports. For example, the lower management wants the reports to be more detailed to follow the development in greater detail, while higher management wants a more simplified and distinct view of the progress of the whole project. The task of "rephrasing" the data for the higher management is often performed by persons on different levels in the organization, which causes time to be wasted (see appendix A). These reports and diagrams should instead be automatically generated from the existing information.

5.2.3 More frequent reports

The regular way of performing progress reporting often means to deliver a regular occurring report, for example once per week. In such an environment you normally cannot get an updated report on "the push of a button", instead you must consult with the development staff to get a proper view of the progress at a given time. This leads to that the managers sometimes only have rather old progress figures available, which can be a potential risk.

One thing that could first be seen as a problem is the fact that the progress might vary quite much in the short perspective, since test cases that pass today may fail tomorrow and thereby effect the progress negatively. However, these frequent progress changes aren't visible in the long perspective since the progress is generated very often, and you can choose to visualize only the general trend.

Another advantage with this approach is that the visibility of the progress might be more open to all parties in the project. Since the system is automated it is just as easy to generate an online-report, which anyone, i.e. the developers, managers, testers etc. can see at any time.

5.3 Disadvantages

The integration between automated testing and progress tracking has the disadvantages that automated testing have by itself compared to the more common manual way of testing. We will in this section describe the most important ones of those and also some additional disadvantages derived from the actual integration.

5.3.1 Personal contact

Since one of the main advantages with this way of working is to save time in the management-related activities it is a risk that the actual contact between the developers and managers might decrease. In traditional progress tracking it is the personal contact that is very important and a good way for the managers to get a feel of how things are going. The human contact is not only good for the managers but it is also a good way for the developer to explain problems and also to get help and directions [Joh99]. In other words, personal contact complements the written reports.

So, the organization should not see this integration as a way to be relieved of the human contact, instead it should be looked upon as a way to get *more time* for the informal human contact. This concludes that the integration actually can be an advantage instead of a disadvantage for the personal contact aspects.

5.3.2 Start-up costs

This new way of tracking the progress would mean that the introductory cost would be rather high. To accomplish progress automation it will be necessary to develop or buy a tool that is needed to extract information from the testing process, as well as introducing the new system to the project management. A new tool or a new way of using a tool can be expected to be associated with a learning curve [Dus99]. In the beginning, the two ways of progress tracking will most likely be run in parallel to ensure correctness of the figures, which of course will increase the cost.

With these additional costs in mind it is important to do a thorough study of all the costs and compare them with the existing way of working to see if the transition is economically justified. If so, it is up to the management to decide whether to invest in them or not. If it will be worth the investment in introducing an integrated test-progress environment the projects should be relatively large, hence if the implementation and test phases are really short it could be easier doing the testing the manual way.

5.3.3 Only applicable when automated testing has started

A problem with this type of integration is that it requires the measured components to be very distinct so they can be compared against a given test-suite. In a project many activities are of a less distinct nature, which makes it hard to really get an automatic progress-value for those. In other words it is hard (or impossible) to implement this on various documents and similar work packages, while it is possible on for example source code since it may be distinct enough. This leads to that this kind of integration only is applicable when the actual automated test-process of a project has started.

This gives that it is necessary to keep the existing progress tracking technique as an additional technique within the organization that is used on the other entities and during the phases of the project that does normally not include automated testing. This can be considered to be a drawback since the time earned and higher quality might not be valuable enough for the organization compared with the additional costs, necessary education etc that an integrated progress-automated test environment requires.

5.3.4 Loss of information

In the more traditional way of progress tracking it is common practice that the developers includes their personal comments of the past week’s progress like problems and or any additional information that might be of interest for the managers. In an environment where the progress data is automatically collected it might be easy to loose this important source of information. After all it is the developers that know all the technical problems that can be potential risks for the project, and those must be presented for the managers somehow!

The reasoning above leads to that some additional interaction must exist between the developers and managers. However this should not be a big problem to overcome, but it is still another issue that must be solved to ensure a proper flow of information in an automated environment.

5.3.5 Cultural issues

As almost always when introducing a new tool to a company or a project, there might be some resistance in the beginning from people thinking that the current tools or way of working are satisfactory. These people might not see the potential improvements that could be made by introducing a new kind of tool. Therefore, to get these people to be motivated of learning the new tool, it is important that they get the essential training needed to be able to use the tools properly. It is also important that the tool isn’t too advanced, but rather easy to use. A simple program with an intuitive GUI, will have a larger chance to succeed within an organization compared with a complicated tool [Bri00].

5.4 Current information in this area

Our study of the existing literature in the area of progress connected to testing has shown that it is not much information written on this subject. One might wonder why this is the case? Is it because of that it is not needed or is it simply not yet covered? We think that this can depend on the fact that software engineering, as a science is quite young.

One major technical issue is that it is hard to create a test environment that accepts specifications in natural language as input. Since software engineering is, as we just said, a relatively young science, natural language is still the main way of expressing how things should work [Mcd93]. This is of course a problem when trying to fully automate the test-progress process, since it is hard to get a computer to interpret such specification in a correct way.

5.5 Chapter summary

We have investigated the advantages and disadvantages that exist when integrating progress tracking and automated testing as a way to improve the progress tracking in a project.

Advantages:

- Correctness. Counting passed test cases is a concrete and objective way of measuring progress.
- Time-saving. The automatic gathering of progress information will make the data collection faster compared to performing it manually.
- More frequent reports. Progress figures can be compiled at any time since the process is automated.

Some of the disadvantages:

- Less personal contact. It can be a risk that the interaction between the developers and management decreases.
- High start-up costs. The introduction of a new system will most likely mean the development of new tool and training within the organization.
- Only applicable during automated testing. This way of measuring progress can only be applied when automated testing is performed (usually during the implementation and testing phases).

6 PROPOSED SOLUTION

This section describes one solution that could improve and solve some of the problems that we previously have described regarding progress tracking.

6.1 Prerequisites

Some things are required to exist in the organization if it will be able to apply our proposed solution. If these things do not exist they must be properly implemented before trying to get an automated progress tracking process such as the one we describes.

6.1.1 Automated testing

Naturally the organization already should have a relatively well-structured automated test process to be able to take all the advantages of this proposed solution. What we mean with “well-structured” is that it should/must have pre-specified test cases with expected results, pass-dates (when the test should have passed testing) and of course that the tests must be run automatically etc. The more well-structured and automated the test process is, the more can be gained when applying our proposed solution since the progress figures will be acquired faster and also more accurately than they otherwise would.

It *would* be possible to use our proposed tool together with traditional manual testing, but then the ability to get fast progress at the current point in time will be lost since the tests take considerably more time to perform.

To be able to run the tests at any given time it is necessary that the software also can be compiled and built at any given time. However since it is a common technique to use nightly builds in large software projects it is quite hard to be able to get a new progress measure several times per day because of the required time by the build-process. In such situation the progress tool that we propose can be run at scheduled times when the build have been completed each night.

6.1.2 Automated comparison

It has to be possible to be able to compare the actual and expected outcome from a test case. This could be done manually, but then you don’t take advantage of the strengths of automated comparison (see 4.2.4). This will mean that the idea of getting the progress faster and more frequent is lost.

The process of comparing the output could either be in the existing testing process or in our proposed tool. In this proposed solution we assume that this comparison is performed in the testing process.

It is preferable if the result of the comparison is saved in some kind of file that can be parsed to be able to extract information about the different tests. Of course it can also be possible to retrieve those comparison results “on the fly” if the test process is adaptable enough since the test software must notify our progress-tracking tool.

6.2 Solution

Here we will describe our proposed solution, and general ideas of the necessary input and examples of results that could be generated from the system. The solution is non-complex in theory, but we see this as an advantage rather than a weakness in the solution.

The solution shows an example of how to integrate automated testing and progress tracking. The reason for choosing this integration as a solution is that we started our work on this thesis to try to find ways to improve the automated testing process at the MPC department. However, when performing interviews with project managers, we noticed that there was a problem with progress tracking at the department. It then seemed natural to us to try to combine these two areas as a way to improve progress tracking, since we found it a good idea to measure progress by the number of test cases that had passed.

6.2.1 System Description

The Automatic Progress Tracking System (APTS), as we can call it, that we are proposing is a system that reads the results from the tests, process it and provides progress data in various forms. The system would be inserted as a management tool in the organization and have contact with the test process to get its necessary input.

In figure 6.2.1 you can see that the APTS has contact with the already existing automatic test process, or at least with the result data from the test process (if the organization doesn't choose to integrate the tool so much with the test process).

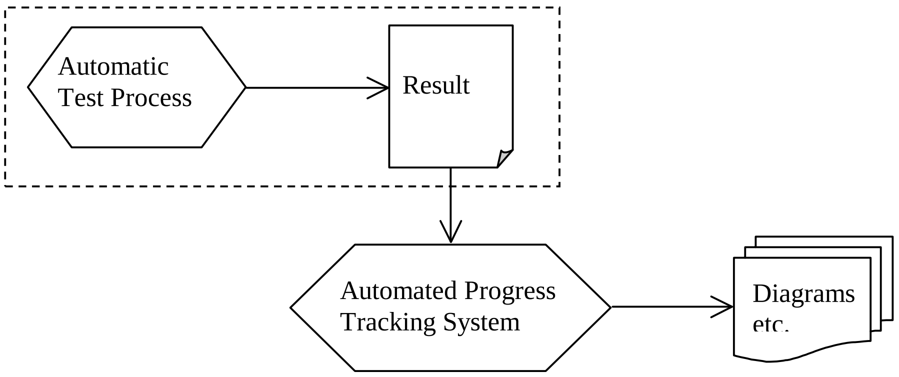


Figure 6.2.1: APTS position in the development process.

It is the manager or someone else in the organization that is responsible for the progress tracking that should use the APTS. He/she decides when the tool should be activated (manually or at regular occurring intervals), that is when the current progress values should be generated. What the APTS then does as a first step, when it has been activated, is to retrieve, analyze and re-store the test data from the test process.

The retrieval of the information is when the tool collects the actual results from the tests. That information is then analyzed, which mostly consists of the parsing of the data, so it then can be re-stored internally in the tool in an appropriate format. The reason for this is that it can later be used in the actual progress tracking calculations. All the data is stored internally since it should remain accessible at a later time for follow-ups, maintain a good traceability and to be able to calculate (and draw graphs) earned value analysis etc.

When the information is stored inside the tool, it can then be used to calculate the actual progress figures at the current point in time. To be able to compare the current progress figures the development plan with dates must be available for the tool. The plan can for example have been manually entered into the tool or an additional interface towards the planning tool if such tool exists.

Finally the progress figures must be visualized in some way. We have seen that the information that people want and need varies, so they should be able to choose what that should be seen in a tool. The different types of the information can for example be reports, statistics or graphs. The information should be visible in some kind of GUI, if it is in the form of an application or a web page is up to the organization. But as we earlier mentioned it is desirable to have a system where the whole staff can see the information, therefore a web page seems to be the most intuitive, simple and maintainable solution to us.

6.2.2 High level design

To get into more detail of our proposed solution we show an example of a basic high level design, see figure 6.2.2.

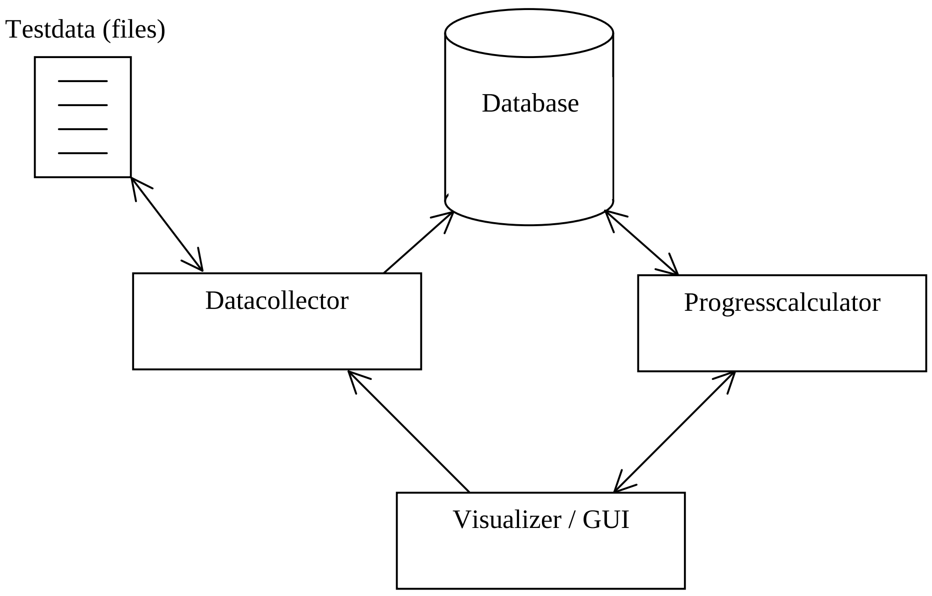


Figure 6.2.2: View of a high level design

Datacollector: The Datacollector parses and analyses the information in the test data generated from the test process. Then it stores the data in a database. The Datacollector can be activated from the GUI, or be automatically started if it is scheduled to do so.

This is the only system-specific component in the APTS. This is due to the fact that it reads the test data from the existing test process. Since the test data that are generated from various test processes can be different, the Datacollector must be customized for the current testing process. All other components in the APTS handle generic data that can be equal in all organizations where the APTS is applied.

Database: In the database, all test information is contained. This includes historical data that is used for all progress calculations.

Progresscalculator: This component is used to calculate the progress from the inputs given to it by the GUI. For example, a user can through the GUI get a progress value for an earlier date or at the current time. The Progresscalculator gets its data from the database.

Visualizer / GUI: This component is where the results from the APTS are visualized. This component can be in the form of a web page or as a regular application. The results given in the GUI is described in higher detail in section 6.2.4.

6.2.3 Input Description

The input required by the APTS to be able to calculate progress metrics is dependant on the need for various figures and diagrams within the specific organization. Some examples of inputs are given below.

Information extracted from the test process (post data)

The information from the test process is required in some form. This can for example be in form of files (textfiles) that is possible to parse to see which test cases that are successfully run and what requirements that are complete. It can also be the case that the testing process notifies the APTS to indicate the success or failure of a test.

Some examples of figures from the test process that can be of interest are [Few99]:

- The total number of test cases.
- The actual number of passed test cases.
- The number of failed test cases.

Additional input

Besides the testing information other information is required to generate the progress figures. Examples of such inputs that might be of use are described below.

- Which test cases that are coupled to a certain requirement (a requirement is tested by one or more test cases).
- The expected number of passed test cases at a certain point in time. This should be easy to extract since you know what requirements that should pass when, and there is a coupling between requirements and test cases.
- Estimated development time for each requirement or the development time to achieve the functionality tested by a certain test case. This is the time that is earned when calculating the earned value.
- The expected number of defects in a certain point in time. This can be extracted by looking at earlier projects and calculating an “average” number of defects that **should** remain in the product at a specific stage in development.
- It could also be possible for the developers to insert their comments for the occurred test failures. These comments can be valuable to have in a progress report since it can help to explain why the test case did not pass.

6.2.4 Output Description

What figures/diagrams that is produced by the APTS depends on what figures that are used as input. The figures/diagrams can be customized in different ways to please all managers, since different managers want different diagrams/reports.

A typical report might contain a summary of the figures that are used as input, for example the number of test cases that has been run, and the number of test cases that has passed. This can be visualized both by figures and with automatically generated diagrams.

Another thing to discuss is *how* this information is to be presented. Should it be a paper report, or visualized on a web page or any other way? One important thing is that the report should be kept short; it should preferably not be more than a single page [Few99]. This will make the report easier to understand, and you don't have to spend that much time on analyzing different figures.

6.2.5 The importance of usability

When discussing the problems regarding project tracking and supervision, one (rather small as we first saw it) detail struck us as the perhaps most important one - *usability*. Usability can be defined as: "*The quality of a system that makes it easy to learn, easy to use, easy to remember, error tolerant, and subjectively pleasing*".

Some ways to improve usability include:

- Shortening the time to accomplish tasks.
- Reducing the number of mistakes made.
- Reducing learning time.
- Improving people's satisfaction with a system.

One of the most significant reasons why people do not like software tools is the lack of intuitive and easy user interaction. A simple software tool with not so much functionality but which is intuitive for the users is in many situations most probably more used than one that requires lots of training and has an extremely large number of features (that *might* be useful sometime in the future). If the tool is advanced, this will most certainly lead to that the use of the tool requires more time [Bri99], and time is for sure one thing that you don't have a lot these days when developing software. If a new tool takes to much time, it might be tempting to solve the problem the usual way, since *"that has always worked in the past"*. Therefore it might be wise to introduce a new tool "in the small" at first and in time add more functionality that is found to be necessary when the introduction has been successful, rather than to offer everything at once.

Of course, you have to, as we earlier have explained, consider that the introduction of a new tool always requires a learning curve, but it cannot be too expensive though.

6.3 Implementation at the MPC department

This section contains a more detailed description of how the general solution in section 6.2 can be applied to the organization at the MPC department. Note that the real benefits of performing automated progress tracking are hard to visualize in a thesis, but it would be more visible in a completely implemented scenario.

6.3.1 General

Our proposed solution makes use of the Basic Test process at the MPC department (described in 4.3.2.1). The main idea works as follows:

1. Information that is required to be able to generate progress figures is inserted into the APTS (for example what requirements and test cases that are supposed to be ready at what time).
2. The output from the Basic Test is parsed and analyzed.
3. The analyzed data is inserted into the database.
4. Figures and diagrams are generated by comparing the information from the test process with the estimated figures.

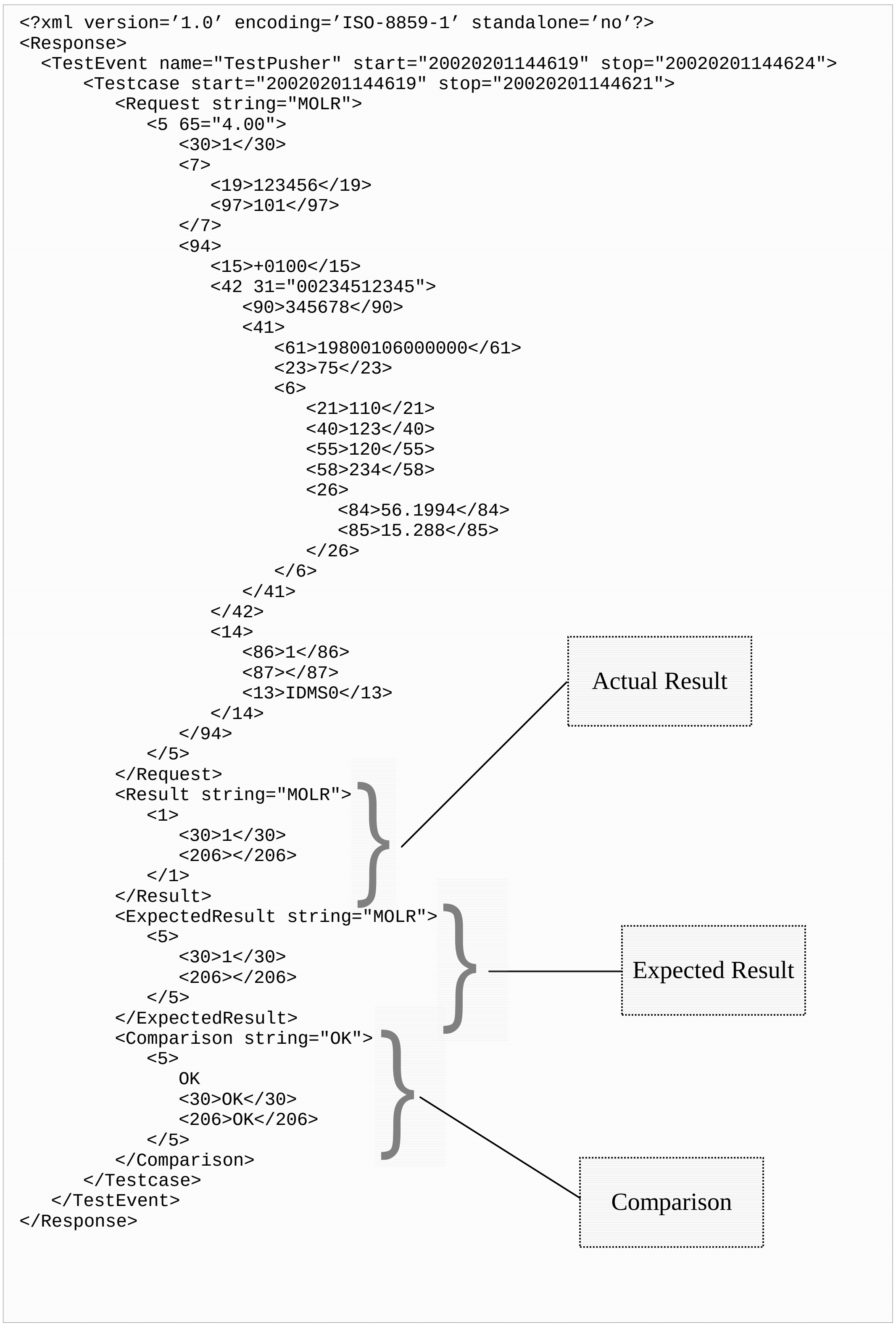
This approach might seem basic in theory, but we think that it could be a good way to improve the progress tracking at the MPC department. In the following sections we describe the output from Basic Test (that is the input to APTS), and give some examples of output that can be produced by APTS.

6.3.2 Input Description

This section describes the input that is necessary to be able to implement the proposed solution, and what parts of this that exists in the present working routines at the MPC department.

Information extracted from the test process (post data)

As mentioned in the Prerequisites-section (6.1), our solution needs the results from the test-process in some form. In the MPC department-case, these results are stored in files (see Basic Test, 4.3.2.1 for further information). The response from a test case can look like the one in figure 6.3.1.



```
<?xml version='1.0' encoding='ISO-8859-1' standalone='no'?>
<Response>
  <TestEvent name="TestPusher" start="20020201144619" stop="20020201144624">
      <Testcase start="20020201144619" stop="20020201144621">
         <Request string="MOLR">
            <5 65="4.00">
               <30>1</30>
               <7>
                  <19>123456</19>
                  <97>101</97>
               </7>
               <94>
                  <15>+0100</15>
                  <42 31="00234512345">
                     <90>345678</90>
                     <41>
                        <61>19800106000000</61>
                        <23>75</23>
                        <6>
                           <21>110</21>
                           <40>123</40>
                           <55>120</55>
                           <58>234</58>
                           <26>
                              <84>56.1994</84>
                              <85>15.288</85>
                           </26>
                        </6>
                     </41>
                  </42>
                  <14>
                     <86>1</86>
                     <87></87>
                     <13>IDMS0</13>
                  </14>
               </94>
            </5>
         </Request>
         <Result string="MOLR">
            <1>
               <30>1</30>
               <206></206>
            </1>
         </Result>
         <ExpectedResult string="MOLR">
            <5>
               <30>1</30>
               <206></206>
            </5>
         </ExpectedResult>
         <Comparison string="OK">
            <5>
               OK
               <30>OK</30>
               <206>OK</206>
            </5>
         </Comparison>
      </Testcase>
   </TestEvent>
</Response>
```

Figure 6.3.1. The result after a test case has been executed.

- Between the `<Request>` and `</Request>` tags the input to the test case is shown.
- Between the `<Result>` and `</Result>` tags the result of the test case is shown.

- Between the `<ExpectedResult>` and `</ExpectedResult>` tags the expected result of the test case is shown.
- Between the `<Comparison>` and `</Comparison>` tags the result of the comparison is shown. If the actual and expected result is equivalent, the result is OK otherwise it is Not OK.

From this information you can see that the current test case has passed (the comparisons are OK). To get a complete view of the status of the project (for example the actual number of passed test cases at a certain point of time), the corresponding results from all test cases are needed.

To be able to calculate the some progress figures other data might be needed. For example if a complete earned value analysis shall be made the planned progress at specific times must be provided, see 6.2.3 for more details.

6.3.3 Output Description

This section describes some of the figures and graphs that can be automatically generated with the input described in 6.3.2.

6.3.3.1 How the output shall be presented

Since the staff at the MPC department uses different operating systems we think that the best way to show the APTS output is on a web page. This way everyone can see the current progress independent on where they are located. Another advantage with a web solution is that it is easy to print the reports on paper if that is more desirable.

6.3.3.2 Different figures and diagrams

From the collected input it is possible to generate several different outputs as we described in 6.2.4. After our interviews (see appendix) and literature study we got some ideas of what data to present. There are a number of different diagram that can be useful for the management:

- Earned value diagram, see example in section 3.1.1.3.
- Number of test cases run versus number of test cases passed, see example in the use case 6.3.4.
- Estimated number of defects at a certain time versus the actual number of defects. In an organization it is important to collect all possible data over the detected defects and to what part of the development process they are derived from. If this is done over several software projects those collected metrics can be used as a reference to compare the current projects with. By that comparison it is possible to get a good indicator of how the current project is doing. See 6.3.4 for an example of such diagram.

Besides the diagrams additional information can be generated to complete the progress report.

6.3.4 Use case description

For simplicity, we assume that the input data from the testing process and the additional input already have been analyzed. This data has been inserted in the database. The content of the database looks like in figure 6.3.2 (the real solution would contain some additional information for example the test execution date).

Test case	Belongs to requirement	Estimated dev. time (h)	Real dev. time (h)	Test case passed?	Estimated finish date
1	A	10	30	Yes	020511
2	A	10	5	No	020511
3	B	15	20	Yes	020514
4	C	20	15	No	020529
5	C	20	20	Yes	020529
6	C	10	10	Yes	020529
7	C	10	10	Yes	020529

Figure 6.3.2. Example of content in the APTS database.

Some ways of calculating progress for the situation in figure 6.3.2: Test case 1, 3, 5, 6 and 7 has passed. This means that 5/7 (71%) test cases have passed. However, if you look at what requirements are complete, only 1/3 (33%) requirement has passed all its corresponding test cases – requirement B.

When counting earned value, the results are as follows: if counting test cases, the progress is 68% (65 h / 95 h), and if counting requirements, the progress is 16% (15 h / 95 h).

The reason for the big differences in this example is the low amount of test cases and requirements. When applied to a “real” system with hundreds of test cases and requirements, the diagrams could look like in figure 6.3.3. However, figure 6.3.2 shows the basic principle of how to use the database content to calculate different progress values.

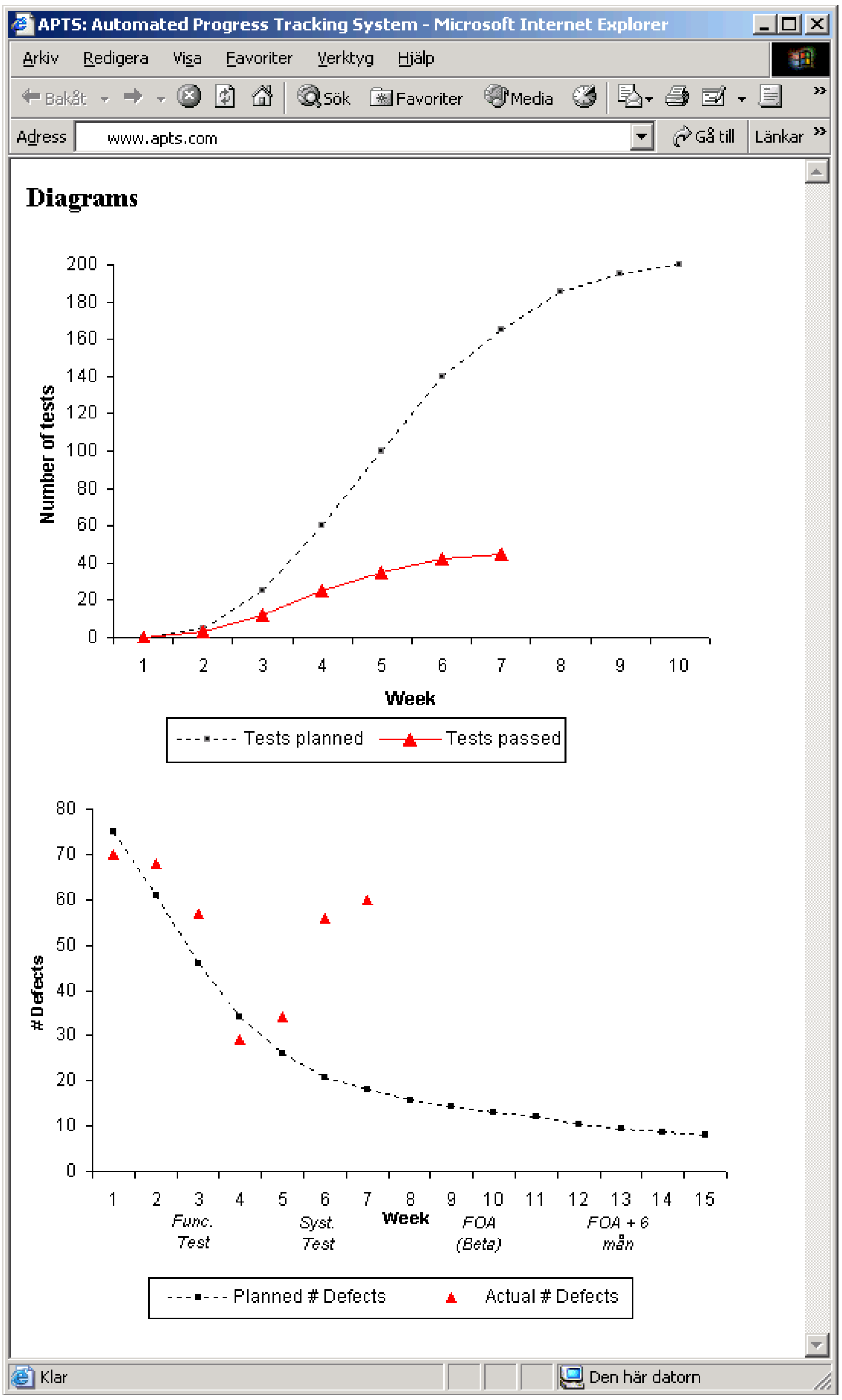


Figure 6.3.3. Examples of diagrams that can be generated from the APTS. The project described has just finished its 7th week. The diagram at the top shows the number of test cases executed versus the number of test cases that passed. The diagram at the bottom shows the expected number of defects at a certain point in time versus the actual number of defects.

6.4 Alternative solutions

In some sections in this thesis (for example 3.1.1.2, Reporting techniques and 5.5.1, Personal contact), we have mentioned that it in some cases can be an advantage to use “regular” talks between the project manager and the developers to get a view of the current status. However, according to us, as this reporting works today it is a problem with this way of measuring, namely that the figures you get can be too subjective. But wouldn't it in some way be possible to improve this reporting process to make it more objective? It is possible to make subjective measures more accurate, for example by letting people with much experience from similar situations do the estimations (expert judgement). However, we haven't been looking further at this question in this thesis, but have focused on if progress tracking could be improved by integrating it with automated testing.

6.5 Chapter summary

The idea of our proposed solution is to integrate automated testing and progress tracking. The prerequisite to achieve this is to have an automated test process that uses some kind of test verification, in our example automated comparison.

Our proposed solution, APTS, works like this:

1. Information that is required to be able to generate progress figures is inserted into the APTS (for example what requirements and test cases that are supposed to be ready at what time).
2. The output from the tests is parsed and analyzed.
3. The analyzed data is inserted into a database.
4. Figures and diagrams are automatically generated by comparing the information from the test process with the estimated figures.

7 CONCLUSION

We think that the area of progress tracking involves a lot of metrics that is not really accurate. This can depend on the fact that many progress tracking figures depends upon subjective comments from developers. It is not only the correctness of the progress figures that is a problem; it can also be the time it takes to retrieve the figures. The progress figures should be up to date, since old progress metrics are not very useful. This is a problem, which in the end can lead to that a project fails due to bad project control. The problem with progress tracking was confirmed by our case study performed at the MPC department at Ericsson Software Technology (EPK) in Ronneby. For example, managers reported that they don't get the information they want, and it takes time to re-format the progress graphs- and figures to suite managers at different levels in the organization.

But, you might say - "The usual way of tracking progress has always worked in the past!". But has it really done that? Studies have shown that it is common that software projects fail or overspend their budgets. Of course this does not solely depend on bad progress tracking, but it might be one contributing factor. Therefore we think that progress tracking in software projects should be improved in some way.

From our literature study we have tried to identify techniques that could be used to improve progress tracking. We have especially investigated the possibility to include data from the testing process into progress tracking and thereby achieve integration between the two areas. One reason for choosing this solution was that we at the start of the work of this thesis studied the automated testing process at the MPC department. However, when interviewing managers we noticed the problem with progress tracking, and started to discuss the advantages of combining the two areas.

Automatic testing alone can be greatly beneficial for an organization. Our literature study has shown that if correctly applied, automation can be a good way to reduce the effort of testing. Automated testing can also help in producing a more reliable system and also improve the testing process itself. However, it is important that it is applied correctly, otherwise it could mean an increase in cost and the automation is done in vain.

From the literature we have also identified advantages and disadvantages with the integration of automated testing and progress tracking:

Some advantages:

- More correct progress data.
- The progress data can be automatically collected.
- The progress data can be collected more often.

Some disadvantages:

- Only applicable to the phases in the development when automated testing can be performed.
- High start-up costs.

Our proposed solution to the problem with progress tracking is to use automated testing as a way to complement the progress tracking in organizations. By using automated testing, progress figures can be collected fast and the figures are objective (for example the number of test cases that have passed at a certain time). It is only a well-defined test case that can verify if a requirement is fulfilled or not. It is important to state that this solution only is applicable when automated testing actually can be performed.

In short, our proposed solution works like this:

1. Information that is required to be able to generate progress figures is inserted into our system (for example what requirements and test cases that are supposed to be ready at what time).
2. The output from the tests is parsed and analyzed.
3. The analyzed data is inserted into a database.
4. Figures and diagrams are automatically generated by comparing the information from the test process with the estimated figures.

The solution described above is non-complex in theory, but we see this as an advantage rather than a weakness in the solution. We think that it could be a good way to improve the progress tracking at the MPC department. When speaking generally, we think that the proposed solution is mostly applicable in large projects that have many requirements and test cases. If the number of requirements is small, progress tracking could probably be done easier manually.

The work in this thesis has been performed in order to support or contradict the following hypothesis:

"The tracking of progress can be improved by integrating progress tracking aspects into the automated testing process."

So, does our proposed solution improve progress tracking? We think it can improve the way progress tracking is done, but it cannot replace regular progress tracking completely. Therefore, our conclusion is that the proposed solution shall only be used as a complement to "regular" progress tracking, since not all parts of a project can be tracked by using automated testing, for example documents.

8 FURTHER WORK

The next step of our work would naturally be to implement our proposed hypothetical solution at the MPC department. It is our belief that this solution would work as a good complement to the current progress tracking, but you don’t know about all hidden difficulties until you start the real implementation. Thereafter we suggest that it would be good to test the implementation of this proposed solution on a single project to see if it works as intended and gives the desired results. In such project it would also be interesting to look at the economical issues to see if any time could be saved when using this kind of progress tracking solution.

It would also be interesting to investigate in a wider perspective (other departments, companies etc.), if this kind of solution is used in other organizations. If so, it would be interesting to study the features of such a tool, and how it works. For example, while finishing this thesis, in late 2001, a tool from Rational was released that contains some of the functionality that we propose. Unfortunately, we have not been able to test this tool.

It could also be interesting to investigate other techniques than the use of automated testing to improve progress tracking. Could the “regular” progress tracking techniques used today (for example informal talks between managers and development staff) be improved to make the figures more objective?

9 REFERENCES

The following books and papers were used when writing this thesis.

[Ben94] Bennatan, E. M., *Software Project Management: A Practitioner's Approach*, McGraw-Hill Book Company, 1994, ISBN 0-07-707648-6.

[How01] Howes, N. R., *Modern Project Management: Successfully Integrating Project Management Knowledge Areas and Processes*, American Management Association, 2001, ISBN 0-8144-0632-7.

[Boe81] Boehm, Barry W., *Software engineering economics*, Prentice Hall, 1981, ISBN 0-13-822122-7.

[Mar99] Martella R. C., Nelson R., Marchand-Martella N. E., *Research Methods – Learning to become a critical research consumer*, Pearson Higher Education, 1999, ISBN 0-205-27125-1.

[Daw00] Dawson C. W., *The Essence Of Computing Projects A Student's Guide*, 2000, Prentice Hall, ISBN 0-13-021972-X.

[Sei98] Seidman I., *Interviewing as Qualitative Research – A Guide for Researchers in Education and the Social Science,* Teachers' Coll. P, 1998, ISBN 0-8077-3697-X.

[Bec57] Becker, H., Blanche G., *Participant observation and interviewing: A comparison*, 1957.

[Dus99] Dustin, E., Rashka, J., Paul, J., *Automated Software Testing: Introduction, Management and Performance*, Addison-Wesley, 1999, ISBN 0-201-43287-0.

[Few99] Fewster, M., Graham, D., *Software Test Automation: Effective use of test execution tools*, Addison-Wesley, 1999, ISBN 0-201-33140-3.

[Mcd93] John Mcdermid, *Software Engineer's Reference Book*, CRC Press, 1993, ISBN 0-8493-7766-8.

[Pfl01] Pfleeger, S. L., *Software Engineering – Theory and Practice*, 2nd Edition, Prentice Hall, 2001, ISBN 0-13-029049-1.

[Bac97] Bach, J., *Test automation snake* oil, 14th International Conference on Testing Computer Software, US Professional Development Institute, 1997.

[Mar91] Marcotty, M., *Software Implementation*, Prentice Hall, 1991, ISBN: 0-13-823493-0.

[Bei84] Beizer, B., *Software System Testing and Quality Assurance*, International Thomson Computer Press, 1984, ISBN 1-850-32821-8.

[Pre01] Pressman, R., *Software Engineering: A practitioner's approach*, McGraw-Hill, 2001, ISBN 0-07-365578-3.

[Vig94] Vigder, M.R., Kark, A.W., *Software Cost Estimation and Control*, National Research Council Canada, 1994.

[Bri00] Briner, W., Geddes, M., Hastings, C., *Projektledaren*, Svenska Förlaget Liv & Ledarskap AB, 2000, ISBN: 91-7738-498-9.

[Sta94] The Standish Group, *The Chaos Report*, 1994, http://www.pm2go.com/sample_research/chaos_1994_1.asp

[Joh99] Johansson, C., Hall, P., Coquard, M., *Talk to Paula and Peter - They are Experienced*, Published in "Proceedings of the Workshop of Learning Software Organizations, June 16, 1999, Kaiserslautern, Germany", pp 69-76.

[Pet01] Pettersson, C., *MPS Frequently Asked Questions*, an Ericsson internal document, 2001.

[Ols02] Olsson, D., *Traceability – a key to software success*, Dept. of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.

10 APPENDIX A

10.1 Problem confirmation

We have read the results from an earlier study [Ols02] of the MPC department's organization. That study says that there are problems with the progress tracking process, since 2 out of 3 project managers thought that the progress tracking was insufficient. This fact together with our interviews shows that the progress tracking could be improved.

Quotes from the interviews in [Ols02]:

"I feel that I have to request information (nag) in order to receive what I need. This can partly depend on the lack of routines to make it easy for people to report".

"The other of the two biggest problems is progress reporting, especially progress reporting up in the organization. It is hard to get cold hard facts. Usually the progress reporting includes quite a lot of gut feeling".

"On the other hand, I feel that there have not been so much information about the progress and problems for other components".

10.2 Questions

The interviews were performed more like discussions, where we tried to get the interviewee to suppress her/his opinions, and also to get as much information as possible about the present routines at the MPC department. We didn't ask each interviewee the exact same set of questions in the exact same order, much dependent on that the interviewees had different positions (work tasks) and knowledge.

Some of the questions that we asked were:

How does progress measuring work in typical project?

How do you estimate tasks in a project?

What do you think could be improved in the work routines (especially concerning project management and tracking)?

Do you think that the progress measuring could be improved, and if so, how?

How does your test process work?

What parts of your test process is automated?

10.3 Interviews

Only the parts of the interviews that are relevant for this thesis are presented below.

10.3.1 Interview 1, project manager

X thinks that one of the most important things when measuring project status is that the tools aren’t **too advanced to use**; if this is the case they simply **won’t be used**. This was the case the last time X was working as project manager and the reporting tool CMISC was supposed to be used. Instead, the different teams delivered status report each week (written on paper), that then had to be “digitalized” by X. Oral communication were also used as a complement to the paper reports, to get a better view of the status.

X thinks that the reporting process could be **improved by simplifying the tools used**, for example in some kind of **web-solution**.

10.3.2 Interview 2, solution manger

X works as a solution manager, which means that he is the link between the project manager and higher instances within Ericsson. His task is to get an overview of the different projects and deliver status reports to these higher instances. X doesn’t care about “lower” development activities, such as the amount of time spent on a specific task, but wants higher level diagrams of the status of the entire project, what risks that exist, if the time plan is followed and what is supposed to be have been done.

When asking X about the **quality of the progress reporting** to X from the project managers X says: “I do actually find it **worthless**”. With this X means that X does not get the information he wishes in form of the right sort of diagrams etc.

One improvement that X sees is the use of automation where it is applicable. For example, automatically generated diagrams and graphs as a measure of progress is a good idea. X thinks that **as much as possible of todays work that is done manually should be automated** (if you earn time or money by doing so).

X finds it important that **old project data is saved** in some kind of database to be able to recreate and compare old project with new ones to improve for example the estimation process. X also thinks that this is important to be able to **compare projects of different size**; this should be possible as the different parts of projects are of equal relative size.

10.3.3 Interview 3, system architect

X thinks that one important thing regarding project tracking is that the figures really “comes back” to the project, that is that they are visualized to the entire project, for example on a web-site that everybody can monitor.

X thinks that it is important to have binary measures of the progress instead of subjective percent figures. X also mentions that the goals set up by the project managers is often rather unclear, and that the managers often are rather non-technical in their work.

X says that the estimation process is not scientific. Nowadays, you look at earlier projects and components when doing estimations.

X thinks that the testing performed at EPK is rather inefficient. The system test is working rather well, but other testing could be more efficient. X thinks that the most earnings can be made before the system test.

Incremental development leads to problems concerning which parts should be part of each increment. This might lead to problems between developers and testers, that is the testers test functionality that isn't yet implemented by the developers.

There should be a list of all tests that should be made. This will lead to that no extra features are added ("just because it is so easy to add them etc."). This can otherwise be the case, and this can lead to problems.

Old test cases should be saved in some kind of tool, otherwise they will simply be removed. This is of course important to do proper regression testing.

What parts of a system that is supposed to be finished when is often unclear.