Software Quality Metrics For Object Oriented Systems

Abdellatif El-Ahmadi

Studie nr. s011837

IMM-B. Eng. – 2006 – 24

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

In this thesis I'm going to analyze the concept of software quality and how it is viewed by different types of software developers in Mærsk Data Defense. I will develop or find a tool that can extract and compute different software metrics and make a translation of these that can be used in a development process. I will develop a system that can be used as a base for all metrics extraction and evaluation of Mærsk Data Defense's software modules and should be a part of the existing development and build process.

My system is called Metric Configuration Program MCP and incorporates an external metric program that computes the software metrics and provides a graphical user interface for different kind of actions.

Resumé

I denne tese vil jeg analysere konceptet software kvalitet og hvordan det opfattes af de forskellige typer af software udviklere i Mærsk Data Defence. Jeg vil udvikle eller finde et værktøj der kan udtage og beregne forskellige softwaremetrikker og lave en fortokling af dette der kan videre benyttes i et udviklingsforløb. Jeg vil udvikle et værktøj der kan bruges som en base for al metrikudtrækning og evaluering af MDD's software moduler og som skulle være en del af den eksisterende udviklingsprocess. Mit system kaldes Metric Configuration Program MCP og den inkorporerer et eksternt metrikværktøj der beregner softwaremetrikkerne og leverer en grafisk brugergrænseflade for diverse former for handlinger.

Preface

This thesis was prepared at Informatics Mathematical Modeling, the Technical University of Denmark together with Mærsk Data Defense, in partial fulfillment of the requirements for acquiring the B.Sc. degree in engineering.

Unified Process has been implicitly utilized, during the analysis and design of the project, where there was need for this without extensive explanation of the subject.

The implementation of the system is done using C# using MS Visual Studio 2005.
The database is implemented using MS Sql Server 2005.

This project is composed of a theoretical section about software quality and a case study where the theory about software quality is directed towards a concrete problem.

Lyngby, June 2006

Abdellatif El-Ahmadi

Acknowledgements

To my family: I would like to give my father and mother my deepest gratitude and deepest respect. Having reached this point in my life has only been possible because of your encouragement and great support.

To Bjarne Poulsen: I thank you for the professional guidance you gave me and the good grasp you had of my project because, since I sometimes had lost focus myself, it was possible for you to steer me back to the right path.

To Torben, Henning and Mærsk Data Defense: I thank you for letting me be a part of the company during my internship and for giving me the opportunity to do my bachelors project and thereby helping me in finishing my education.

To Antonio Fausto: I thank you for the support and brilliant ideas that you contributed to my project and for helping me with a thorough review of my work.

Table of contents

1 Introduction

1.1 Purpose

This chapter will describe the purpose of this project and provide the reader with a basic understanding that is required to form an overview of the foundation and content of this project. It will also highlight the areas where this project will be useful at Mærsk Data Defense.

1.2 Mærsk Data Defense Background

During my internship at Mærsk Data Defense I have experienced how they develop software applications. I have participated in several meetings and reviews about the overall software and design architecture where the people involved reviewed each others documents and agreed that the documents were accepted. This gave me some ideas to my bachelor project that there must be a way of measuring on the end software product. In others fields of engineering it is possible to make a direct measurement on the product that you develop, like e.g. the automotive industry. When a prototype of an automobile is developed it goes through quality measurements to ensure that it meets the accepted standards. The quality of e.g. the brakes is verified by a quality control where various aspects can be tested, like e.g. the hardness of the material. This gives an exact number to work with and can direct you to improve the production.
With software development there aren't exactly the same possibilities to measure these quality factors. The software is tested by different types of tests which verify that it is working properly under different types of conditions and meets the customer's requirements. These tests only ensures that you have developed software that the customers ordered, they do not directly tell you anything about the quality of the software.

This gave me an idea of developing a system that can make these quality measurements.
Before I can do so I must first define what software quality is. A direct measurement is very complicated and the definition of high quality software is thereby just as complex.
Software quality is perceived differently from person to person and therefore there isn't an unambiguous answer to what software quality is.

1.3 Basis for software measurement

Object-oriented design and development has become very popular in today's software development environment. There is indeed a great benefit to this way of designing software and this is now a day a widely recognized method of software improvement. This way of designing software is, opposed to the old procedural way of designing software, object oriented. As object-oriented design uses different approaches to developing software, one also has to use different ways of inspecting this software and taking measurement on it. The field of object-oriented metrics is a relatively new study. The traditional metrics such as counting the lines of code is not sufficient for object-oriented development metrics. Software measurement should be an essential part of a development process, just like testing. If we don't measure the developed software we might not know whether we are violating any object oriented principles at an early stage. By testing the software we get bug free and good working software, but we can't test whether or not we have a good object oriented design. We could make a perfect piece of working software and still end up with a design that for example is not maintainable and therefore are not benefiting of the object oriented principles. Later on in chapter 2, I will go more thoroughly into what software quality is composed of. For now, let me just for say that the need for measuring is important, because a key point of any attempt to improve something is that we must measure where we are in our work and software development is no exception.

1.4 Needs and Expectations

My expectation to this project is that it shall establish a new phase in the existing development process. I also expect that it will get some attention and that software measurement will become a very important and essential part of future software development; whether it is going to be my system or some other system is not relevant. As this is a complete new concept in the development process, I don’t expect that it will be a huge success; only that it will be useful in some degree and I expect that it will inspire the project leaders to establish a software metric program.

1.5 Project Description

In the light of previous considerations about software quality I have decided to define the scope of my project to deal with software measurement as part of software quality.
The basis of this project is to develop a system that can be used to extract different software metrics on the software that are developed at Mærsk Data Defense.

- I will highlight the vagueness about software quality and analyze what software quality is and find the meaning of it from different persons view.
- I will find out how measurements on a software product can be performed.
- I will show examples of how the result of the measurement can be interpreted and further used.
- Develop a system that can assist in the extraction of software measurements.

1.6 The contents of this thesis

Chapter 1 has described some basic things about the foundation and description this project
In chapter 2 I will cover the main subject namely, software quality, and give a short description of some code and design metrics and finally end with a presentation of the metric application that I have used in my project.
In chapter 3 I will take the step from the theory in chapter 2 and into a case study where I will implement a prototype of a system that can be used to extract software metrics.
Chapter 3 is constructed of the software engineering artifacts like analysis, design, implementation and test and will end up with a small prototype.
Chapter 4 is the project conclusion, which is composed of 4 parts, a chapter summary, the final conclusion, a broader perspective and finally a short description of future extensions to the system.
Chapter 5 is the list of literature that I have read and which have inspired and assisted me in writing this thesis.
As this copy of the thesis is open to the public, I have excluded some parts in the appendix like the internships report and the source code for my system.
Finally is an appendix A, which in this copy only contains use cases.

2 Software Quality

2.1 Purpose

This chapter contains the basic theory on software quality and it explains the theory about some of the code and design metrics that are present up to date. The purpose is to find the essence of what software quality is and what meaning the software has for different people. It will also give a short description of some of the software measurement possibilities and discuss how the result could be interpreted.
In the end of this chapter is a short description of some of the popular metric applications and the chosen one.

2.2 Into the quagmire

Object-oriented design and development is becoming very popular in today's software development. Object-oriented development not only requires a different approach to designing software but also a different approach onto software measurement. Since Object-oriented technology primarily is based on the use of objects and not algorithms, the approach to software metrics has to be different from the traditional metrics, such as *Lines Of Code* or *Cyclomatic Complexity*. These metrics has become very popular as a standard in the old traditional procedural programming but their use in Object-oriented development is far from enough alone. Though, they are not quite deprecated; in combination with other Object-oriented metrics they can still give some value. In fact some of the new proposed Object-oriented metrics are based on e.g. the *Cyclomatic Complexity* metric.
Before any attempts are made in improving the quality of software we must first find out what quality is. Software quality is pretty much like beauty; it lies in the eyes of the beholder.

Software quality is not only a very wide and abstract term; it is also kind of polymorphic, in the sense that it can possess many different definitions depending on the viewer. This forces us, first of all, to define the stakeholders that are involved and then view their requirements with respect to software quality.
Instead of just listing the different stakeholders I have made a figure (figure 2.1) that illustrates these in a more superior view.

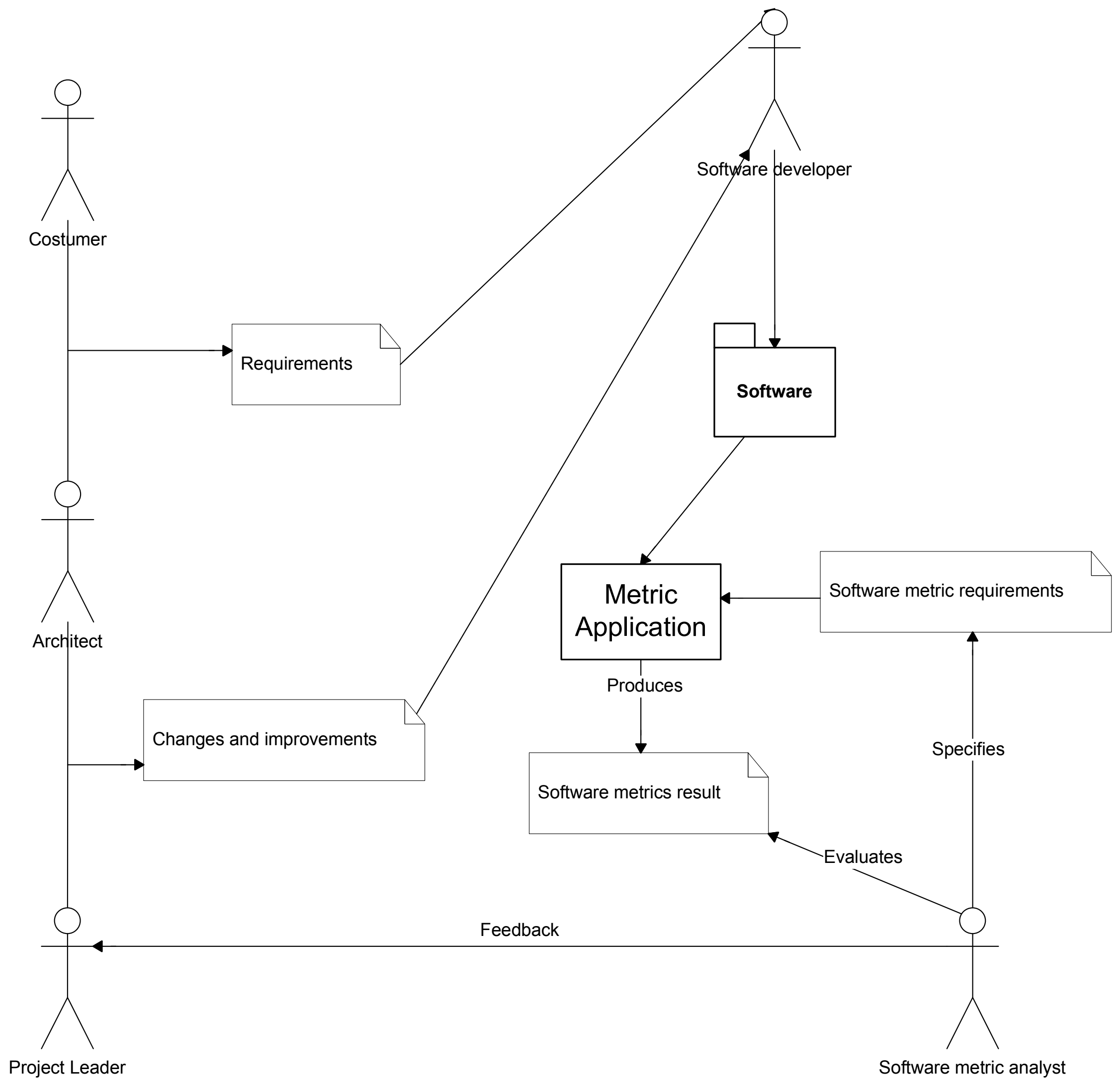


Figure 2.1 a description of the development process with the metric extraction included and the stakeholders involved in the process.

Figure 2.1 shows the different individuals in a creation of a software product and that software quality simply can't be regarded in the same way by everyone.
Take for example the costumer, he has some requirements that he desires in the end product.
The developer then implements a system from a requirements specification. If the costumer's requirements were directly transformed to the end product then it is certain that the costumer would be happy with respect to the functionality of the system assuming that he hasn't changed his mind.
Unfortunately this is not the issue; there are always some requirements that are missing.
Figure 2.2 illustrates the problem. It shows the amount of the costumer's requirements () that are transferred to the end product together with some other requirements from the developer. For this reason the costumer looks mostly at the product from the outside with respect to the functionality.

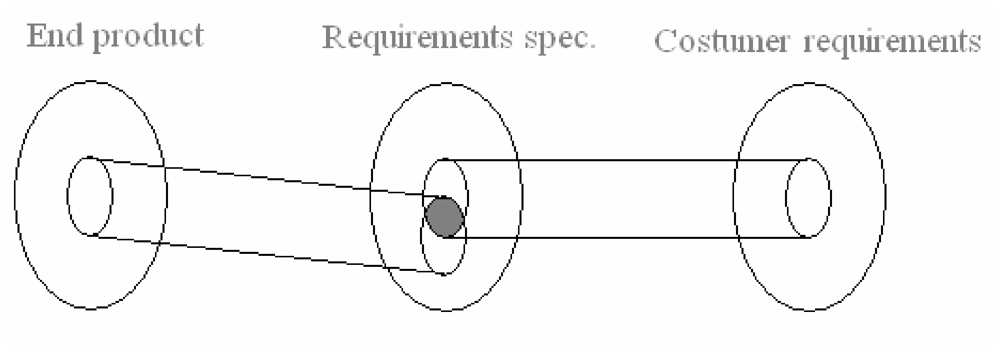


Figure 2.2 this shows the costumers requirements and the functionality in the end product.

If the costumer is satisfied with the end product, i.e. it lives up to his expectations, then for the costumer this is good quality. He might later find it to be rubbish during the maintenance because of the way it has been constructed but this is irrelevant now.
Looking at the quality from the company view then quality might have a different meaning.
They also look at the product from the inside (they created it).
As they are developing software that uses object oriented way of thinking they aim at ending up with something that complies with certain characteristics. So if we strive to end up with something that contains specific characteristics then this must be what quality is from the company view.
If I want my product to be of a high quality I also have to specify which quality factors that I'm interested in.

If we want something to be reusable we try to make it general to fit different kinds of situations.
If we create a large class to perform many specific assignments and bind it to specific classes then we can almost be certain that this class won't be much reused in other contexts.
Some might only want a software product to be used once and only in one form so they might only think of the quality factors in their product as being reliable, but nevertheless the quality has to be specified in factors that can be measured by metrics.
The thing that we should evaluate for improving the quality is not necessarily only the software product as the implementation but might also be the design specification, documentation, user manuals etc. as these also has an effect on the end product **[4]**.
When speaking of measuring on the software product there are two ways to tackle it; by looking internally and externally. The internal attributes are for example the size, the coupling the cohesiveness, the modularity, hierarchy etc.
These internal attributes are what are created when developing the product and they can be directly measured and they affect the behavior of the end product (the external attributes) **[2, 4]**.
An analogy from the automotive industry is that if we wanted to improve the performance of a car in terms of brake horsepower, we would start by tuning the engine by mounting a turbo charger. The performance here is an *external* characteristic and cannot be measured directly (like software quality) and the gain in brake horse power that we get by mounting a turbo charger is the *internal* characteristic. We aim at some external characteristics which in turn are directly affected by the external characteristics. This applies to software engineering as well where good internal characteristic implies good external quality.

In **[4]** Fenton states that if the previous statement was wrong then almost all the software engineering research has been worthless but at the same time claims that there has been few scientific attempts to establish a specific relationship between these points.
Having a clear knowledge of the internal attributes and a clear goal of the wanted external characteristics is crucial in order to obtain something useful.
It is appropriate to end with a quotation by Gilb in **[5]**:
"*Projects without clear goals will not achieve their goals clearly*".

In figure 2.3 are the combined quality factors.
The figure is a combination of the quality factors which has been proposed by some of the leading researchers in the field of software quality.

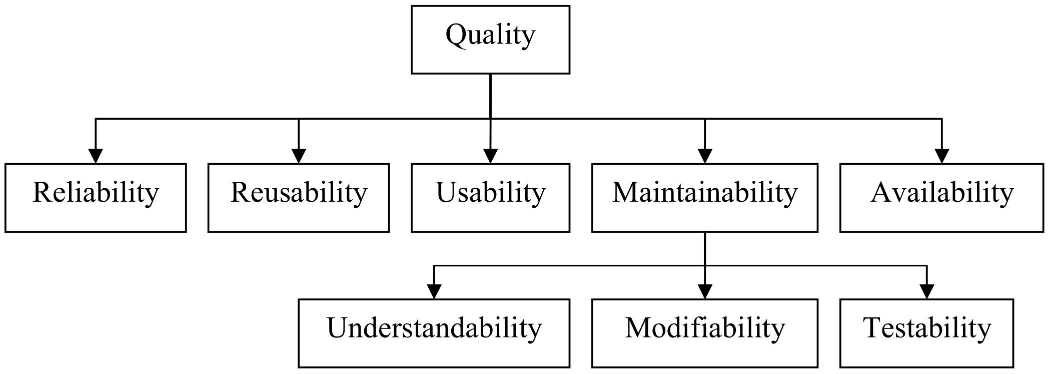


Figure 2.3 Quality can be considered a being composed of reliability, reusability, usability, maintainability and availability, where maintainability is then composed of understandability, modifiability and testability.

Apart from these quality factors there are also other characteristics that are interesting like the complexity of the software. The complexity is not something that you can directly avoid in your design. There are many factors that contribute to the complexity. Or put it another way, the complexity influences some of the wanted characteristics mentioned above. Complexity is a very abstract term and it is the most used term but at the same time it is also the least well-defined. The thing about complexity is that it has to be interpreted individually before it gives any useful meaning apart from the intuitive perception we have of it as being something that is difficult.

The complexity affects many aspects of the software quality and therefore it is important to pay attention to it. As figure 2.4 illustrates, the software complexity affects, the understandability, modifiability and testability in that sense that it becomes harder to comprehend for someone who hasn't made it, harder to change and it requires more elaborate testing. The software complexity it is sort of a psychological factor which we can't directly control, because if the software is maintained the software complexity is affected in a good direction, and if the software made is complex it affects the maintenance in a bad direction; it is a circular process and requires that the software constantly getting maintained to keep the complexity as low as possible.

Maintenance is very essential and helps to avoid future problems.

So measuring on software products points out where the problem areas could be and thereby indirectly reduce the development time and cost in the future, because early actions can be taken in preventing the software becoming more complex.

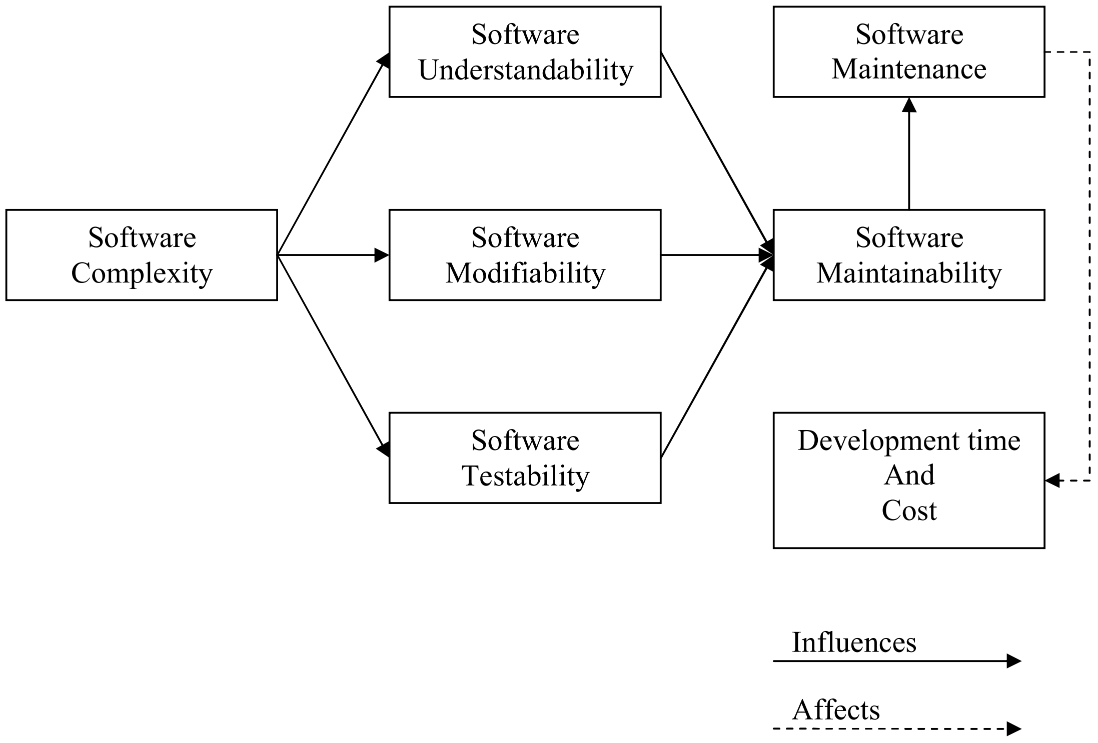


Figure 2.4 The interconnection between the external characteristics, especially the maintainability and its contributing factors, and the complexity. **[2]**

2.3 Code and design metrics

In this part I will explain some theory about the different software metrics that are most often mentioned in literature about software quality.
For each of the metrics I have described the overall theory and then gone deeper into the technical part and interpretation of the specific metric if there is a reason for doing it.
Every metric has a value that should be compared against some other value to obtain any useful interpretation. There isn't any strict number for these values as these values should be gained for every project over a period of measurements and then use these values as thresholds, but in some cases threshold values can be given as an indicator. The study of object oriented software metrics is relatively new and there are many different people presenting their own interpretation of these. There is still a large discussion on what characteristics a software metric should fulfill before it is useful and defined as a valid metric.
Chidamber and Kemerer are two of the leading authors in software metrics and have introduced in an article **[6]** a basic suite for colleting object oriented code and design metrics.
I'm just going to give an explanation of some of these metrics. As the number of metrics is large I have only selected a few of these to describe. I have included two of the old metrics, the *cyclomatic complexity* and *lines of code*. Even though they are not sufficient they are easy understandable and if used and interpreted correctly, they can still provide some value to the evaluation of software quality.

2.3.1 Cyclomatic Complexity

The Cyclomatic Complexity is a procedural software metric but is also useful to object oriented design at the method level. It has been introduced by Thomas McCabe in 1976 and it measures the number of linearly-independent paths through a program module. This measure provides a single ordinal number that can be compared to the complexity of other programs. Cyclomatic complexity is often referred to simply as program complexity, or as McCabe's complexity. It is often used in concert with other software metrics. It is not an object oriented metric as it has nothing to do with the object oriented technology. What still makes it useful in today’s object-oriented design metrics is that it is easy to apply and understand.

The cyclomatic complexity for a module is computed from a connected graph of the module which is also the control flow within the program and is derived from graph theory.

$$\mathbf{v}(G) = e - n + 2$$

where

v(G) = the cyclomatic complexity

e = the number of edges of the graph

n = the number of nodes of the graph

The result is a number of independent test paths in the program.

A programming example also demonstrates how little is required to increase the complexity. Listing 2.1 shows a rather simple structure, which has a cc value of 3, and this value becomes much larger if the method grows with many decisions points.

```
public int getValue(int param1) {
  int value = 0;
  if (param1 == 0)  {
    value = 4;
  } else {
    value = 0;
  }
  return value;
}
```

Listing 2.1. A simple program with an if-else structure

A method with a low cyclomatic complexity is generally better. A low cyclomatic complexity contributes to a program's *understandability* and indicates that it is open to modification at a lower risk than a more complex program. A cyclomatic complexity is also a strong indicator of the *testability* of the program as a complex program requires more elaborate testing to ensure that it works. There are different interpretations of the values but as an indicator, a value for the cyclomatic complexity below 10 indicates a simple program with increasing complexity for increasing value. Because this metric isn't designed for object oriented design it has to be used and interpreted carefully. In general it should be compared with other values of cyclomatic complexity. As with all other metrics one should analyze the whole picture and especially pay attention to the areas with the worst values. **[2,6]**

2.3.2 Lines Of Code

This metric measures the size of a method. It measures the physical lines of code in a method or sometimes the lines of code of IL instructions, depending of how it is implemented. In VIL it measures IL lines. The size of the method could be an indication of an easy to understand program and easy to maintain, so it affects the modifiability, understandability and also testability.
The threshold values are very hard to define because they depend on the programming language. In average for C++ projects a value of 24 lines[1] of code per function. This metric should not be thought as an answer to high quality as a large program isn't necessarily a good constructed one. It should rather be used evaluate the size and to locate the larger areas in a program for further inspection. **[1]**

[1] Page 40 in [1]

2.3.3 Depth of Inheritance Tree

The depth of a class in the inheritance hierarchy is the maximum length from the class to the top, measured by the maximum number of ancestor classes.
The deeper a class is in the hierarchy the more methods it can inherit which make it harder to predict the behavior. Deep class trees affect the *complexity* of the design because more classes and methods are involved and from figure 2.4 we can see that complexity affects the *understandability*, *modifiability* and *testability*. **[2,6]**

2.3.4 Weighted Methods per Class

The Weighted Methods per Class metric is a count of the methods in a class or the cyclomatic complexity of the methods within a class. The cyclomatic complexity is computed with the cyclomatic complexity metric. The cyclomatic complexity of the methods in a class is a predictor of how much time is required to maintain the class and thereby all its methods. The larger number of methods in a class and the more complex they are the more time and effort is required on not only maintaining that specific class but also at paying attention to the impact it will have on any children inheriting from this class as this will affect the *reusability*. **[2,6]**

2.3.5 Response For a Class

The Response For a Class metric is a count of all the set of methods that can be invoked as a result of a message call by an another object or by methods within that class, which is all the methods that accessible in the class hierarchy.
It is a view of the complexity of the class based on the number of methods and the communication with other classes. If a class has a large number of methods that can be invoked from others the complexity is affected. Also the testing becomes more complicated it requires more time from the testers side to get a good understanding. So this metrics evaluates the *testability* and *understandability*. **[2,6]**

2.3.6 Practical usage of metrics

When computing the software metric values one should also consider what to do with these values afterwards. Nearly every book on software metrics (the ones that I have read) propose a set of guidelines of how to interpret the overall meaning of the metrics, but there hasn't been many who has proposed statistical evidence that for example a CC value of 10 is twice as complex as a CC value of 5. I don't think that one can come up with a specific translation of the value as they are very individual; they must be interpreted as the measurements go on. One thing to do is to collect the metric values and present them in a histogram. A histogram over the values will demonstrate prevailing and extreme values very clearly. One could for example take all the CC values for all methods in a class and present them in a figure like the one beneath.

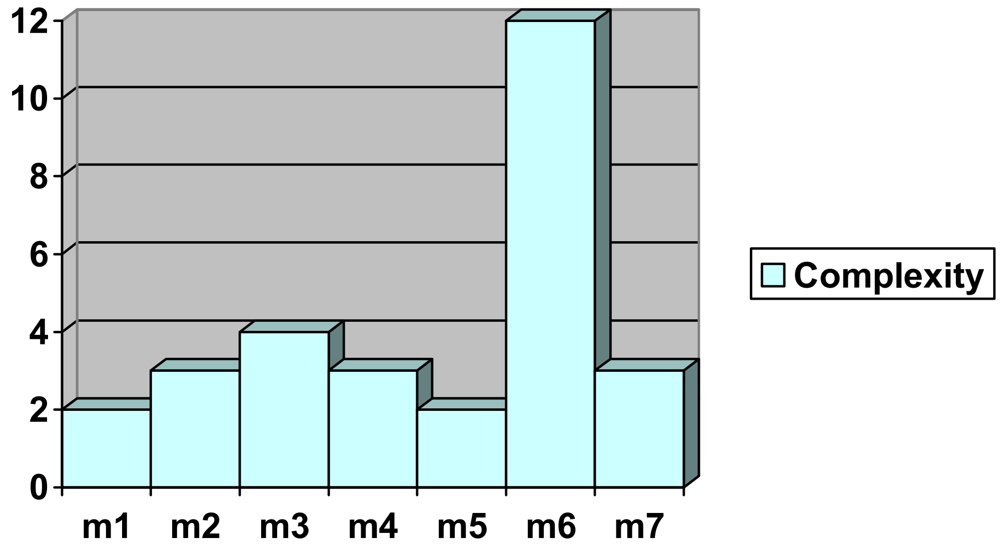


Figure 2.5. A histogram over the cyclomatic complexity of fictive methods in a class

This figure gives an excellent picture of the complexities for the methods in this class and shows that most of the values are approximately around 2 – 4 but for methods 6 it is 12. This is clearly a reason for wanting to inspect this method in particular. One could then ask the question, what if the values were all 4; does this, then, indicate that these methods aren't complex?
The answer could be both yes or no.

It could imply that they are not complex because the value is fairly small, but it could as well be that they all had a value of 20. It could then also simply indicate that they all were equally complex. This just demonstrates that one cannot just directly say that this value is good or bad or somewhere in-between. The values should be compared with other values for other classes to give any useful meaning. If all other methods in other classes have a value of approximately 5 and this class' methods were approximately 10 then again these methods are twice as large as the average and should be inspected.

It is a constant comparison and one cannot by extracting a single value directly imply that this value is low or high as this should be regarded with respect to other values to define what is low and what is high. The guideline values that are mentioned for example in the cyclomatic complexity have also been derived by a comparison between different kinds of programs. To make use of the values one has to avoid a narrow view and instead consider the wide perspective.

2.4 State of the art metric application

In my search for a program to compute software metrics I have discovered a program on the web, which is called VIL. In the beginning of my project period I tried to examine the possibility of creating such program myself but I found out that much of my time was going to be wasted, as making such a program is very time consuming. To compute metrics from a software module requires that you have some sort of knowledge about the programming language that is used and make a parser that recognizes the keywords in a program. As I only have 10 weeks to do the project I decided that making such a program was a waste of time and that it wouldn't contribute that much to the overall project. I found some different programs on the web that compute software metrics on different programming languages
In table 2.1 are my requirements to the metric application and a variety of some of the applications found on the web.

Requirement	VIL	PREfast	FxCop	PREsharp	Essential Metrics
For C#	√	-	√	√	-
Freeware	√	-	√	-	-
Simple and easy-to-use	√	?	-	?	?
Compute static code metrics	√	√	√	√	√

Table 2.1. A list of some of the possible metric applications

I decided to use VIL as it has all the requirements and it there wasn't any complications in the use and it was very easy to begin with in the start of my project. Another alternative if the language is C++ could be FxCop or PREfast which has much more functionalities.
For java projects, Essential Metrics could be used.

VIL is a program that can compute software metrics of .NET assemblies, classes, and methods for all .NET languages, including C# and Visual Basic.NET.
Vil inspects .NET Dll's and Exe's and provides an easy means of rapidly filtering and sorting through thousands of classes and methods in multiple assemblies to find code meeting specific criteria, generate reports, or to help project managers answering questions regarding to their own software progress.
VIL is a command line based program, which easily computes the wanted metrics but it isn’t flexible and hardly adaptable so in order to use it you have to use it in a command prompt and then manually read the reports that it generates. VIL takes some fundamental parameters in order to execute the measurement; these are first of all an assembly (.dll or .exe file), one or more desired metrics, and optionally an output file where the result of the measurement is stored.

3 CASE STUDY: Implementing a measurement program for quality evaluation of Object Oriented Software

3.1 Introduction

In a time of increasing expansion by software in nearly all areas of our life software quality is a very important criteria in order to trust in its reliability and functionality.
It is also important from a business point of view e.g. to get information on the quality of software, since software now a days tend to grow to complex structures over time. Because of the size and complexity of software it is usually impossible to evaluate it manually. For this reason tools to measure software components and extract different metrics are essential to support this task.
This case study introduces a possible solution to what it requires to establish a measurement program. The study does not implement the tool for the actual computation of the metrics but it offers a graphical user interface to such a metric tool and incorporates this tool in a whole system that is user friendly and provides the possibility of storing and viewing the results.

3.2 Requirements Specifications

3.2.1 Functional requirements

The system shall solve some higher-level problems; these are summarized as functional requirements in short descriptions below.

- Encapsulating the execution of external measurement program

The system shall provide a unique way of software measuring. The users shall be able to access the same system and make measurements on the software without knowing how these measurements are computed or even seeing the system that computes them. It provides a common interface and hides out the underlying system.

- Adapt into existing build process

The system shall adapt to the existing build process and be scheduled after a system build has completed.

- Give a translation of the extracted metrics

Besides computing the metric values, the system should also be able to give an interpretation of these numbers and present the measurement in an easy to understand manner.

- Storing measurement data into a database

Every measurement should be stored persistently in a database so the possibility of building up some kind of metric database that can be used to compare the software development over a time period.

- Publishing metrics to the entire company

The result of the completed measurements shall be published on the company webpage so that every employee can see the result.

3.2.2 Requirements Prioritizations

In the previous part a have outlined the requirements in some small groups of functionality. Here I'm going to give a more elaborate description in order so that these can be directly mapped over to a respective use case and then give them a prioritization. I focus on the requirements that give me a working system and only on the core functionality. The core functionality is defined as the first thing that is needed in order for the system to start up. The functionalities that aren't essential to the system are treated as a nice-to-have features and will be implemented if there is the time for it.

Ranking	**Requirement**	**Comment**
High	Perform a measurement	To execute a measurement is the essential of the whole system and it is therefore given a high ranking.
Medium	To save a measurement into the database	To save a measurement is important but not essential for the system to work properly.
Low	To present the measurement from the database	To read the measurement from the database and show them in the system is not essential and is treated as a nice to have functionality.
Low	To view the measurement graphically	To show the measurement data graphically in the system is not essential and is treated as a nice to have functionality.

Table 3.1. A ranking of the requirements and a short description of the task to accomplish.

3.3 Analysis

3.3.1 Purpose

The purpose of this chapter is to define use cases on the basis of the previous requirements and present a prototype of how the systems graphical user interface should look like in order to fulfill the requirements.

3.3.2 Use cases

In the UP the use cases are a very impotent way of describing the system that is being developed. Use cases describe the usage of the system in a structured way and they are in fact the system requirements along with other higher-level requirements.
From the high-level requirements listed in chapter 3 I have identified the following use cases but only included the first one in the report.

- Start new measurement
- Save measurement
- View measurement
- Delete measurement

The rest that are specified can be found in **appendix A.1**.

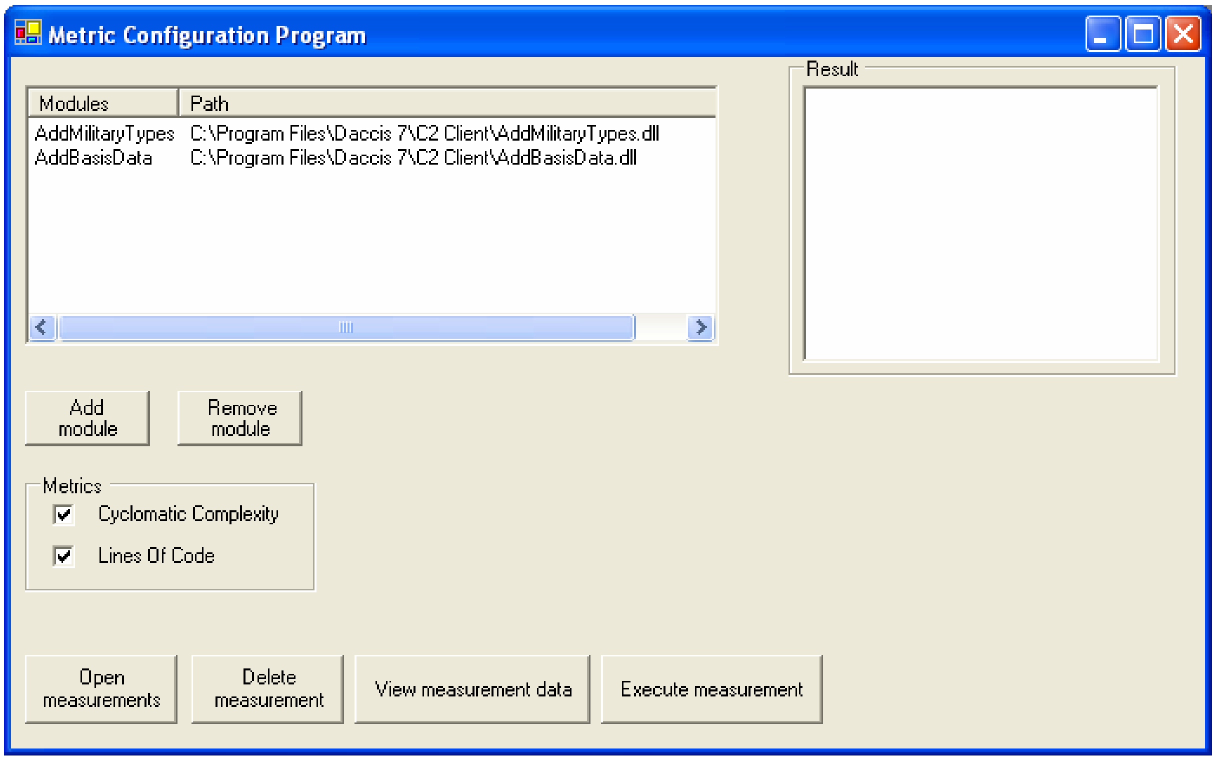
I have decided that the first two use cases should be united into one. This is more appropriate; as the measurement is done it is saved immediately in the database. If it were to be done in two parts I would have to store all the output files for all the completed measurements so that the user can chose which measurement to store later on. Now I only have to store the latest output files until they directly are stored in the database; and afterwards these file are not longer needed. This implies that every measurement completed is stored at the same time and the user cannot choose to store a measurement.

USE CASE 1	Start new measurement	
Goal in Context	An actor accesses the system, select the wanted modules and starts the measurement	
Scope	How to start a new measurement	
Preconditions	The measurement program is installed and running, modules are present for measurement	
Success End Condition	A measurement completes and the result is ready for further processing	
Failed End Condition	The measurement can not begin/complete	
Primary Actor	Software metric analyst, QA manager	
Secondary Actor	Measurement program, metric program	
Trigger	A systembuild is ready or a measurement is requested	
DESCRIPTION	**Step**	**Action**
	1	User selects the desired modules
	2	User select the desired metrics
	3	User starts the measurement
	4	System saves the measurement and responds with an acknowledgement
EXTENSIONS	**Step**	**Branching Action**
	1a	No modules are present: A system build is not ready yet
SUB-VARIATIONS		**Branching Action**

Table 3.2. The use case of executing a measurement.

3.3.3 System prototype

The system prototype is how I would like the final system to look like and what contents it must have so that the implementation can easily take shape. By having an idea of what the GUI should look like I can very easily begin the implementation of the underlying system instead of just starting at an random point. Creating a quick GUI prototype in Visual Studio is very easy and a good starting point and this can be seen in figure 3.1.



Modules	Path
AddMilitaryTypes	C:\Program Files\Daccis 7\C2 Client\AddMilitaryTypes.dll
AddBasisData	C:\Program Files\Daccis 7\C2 Client\AddBasisData.dll

Metrics
- [✓] Cyclomatic Complexity
- [✓] Lines Of Code

Figure 3.1 a prototype of the graphical user interface of the metric configuration program.

The GUI is rather understandable but I will explain what's unclear.

The button labeled *Open measurements* is used to show all the measurements that has been completed and saved. It should extract a list from the database table and present the measurements in the result window.

The button labeled *View measurement data* are used to show the actual data for a measurement. This then extracts the data from the database of the selected measurement(s).

3.3.4 Domain Model

A good way of getting an overview over the entire system and the flow is by looking at a model of the overall system domain. In figure 3.2 is a domain model of the system, which shows my entities and the communication and flow of the program.

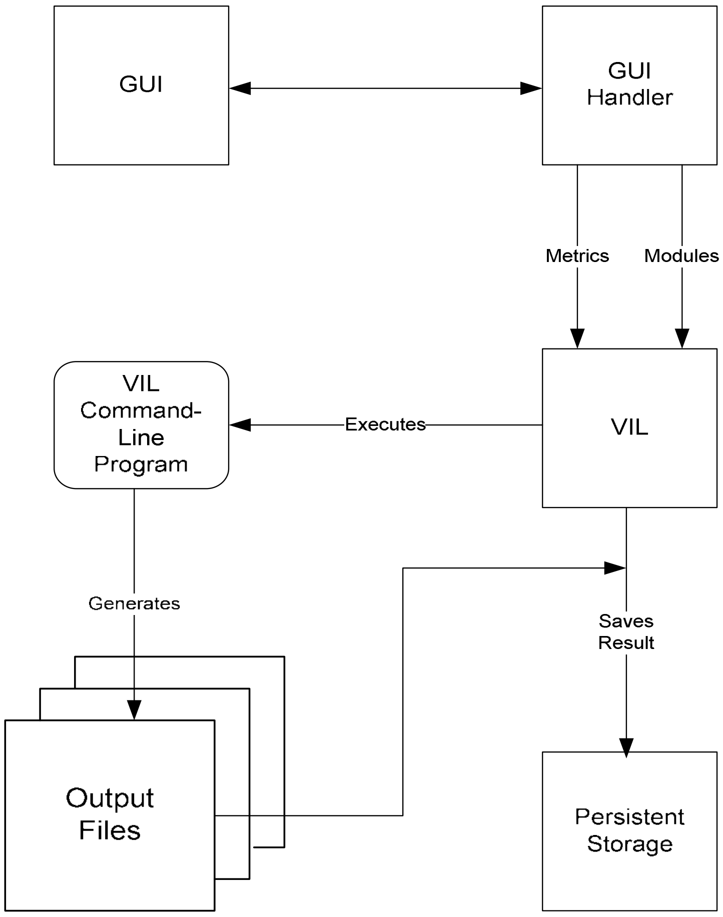


Figure 3.2 a model of the domain in the Metric Configuration Program.

Figure 3.2 shows the different entities and the flow in my system. The GUI simply provides the visuals aspects and the GUIHandler sends the modules and metrics to be evaluated. After execution it saves the generated output files in the database. To view the measurement results; one can either regard them in the result window in the GUI or by any other program that needs to present the result of the measurement for example a webpage more specific an ASP.NET webpage. The result is stored in a database and is easy to access by any means and further present them as whished.

3.3.5 Risk management plan

The risk management plan shows the project risks and estimates their probability and impact on the project. The estimates are simply qualitative and ranked between low and high. The worst risk are of course those both probable and of high impact.

Risk	**Probability**	**Impact**	**Mitigation ideas**
Programming problems caused by insufficient knowledge in C#	Medium to High	High	Start out by reading C# tutorials and windows programming books and do some exercises. As my background is Java I don't think that the probability is very high as C# is similar to Java
Database problems caused by insufficient knowledge in MS SQL	Medium to High	High	Start out by reading database tutorials and exercises. I have some experience with MySQL and therefore the probability is not very high.
Software quality is a new aspect for me so I have to focus on acquiring as much knowledge on this area.	High	High	As software quality is my primary focuses and I don't have any knowledge in it both the probability and impact are very high.

Table 3.3 a listing of the possible problem areas and a prediction of the occurrence.

3.4 Design

3.4.1 Purpose

The purpose of this part is to define the software objects and their collaborations.

3.4.2 Overall architectural design

The architectural design describes how the system is constructed. I have naturally (this is what feels logically correct) chosen to divide my system in layers, more specific in a 3-layered architecture. The benefits for this layered division are many more than I can mention but the overall idea is that you get a system that is easier to test, easier to maintain, easier to extend and not to forget that it's harder for possible errors to spread to the whole system.
The layers are divided after assignment and are illustrated in figure 3.4.

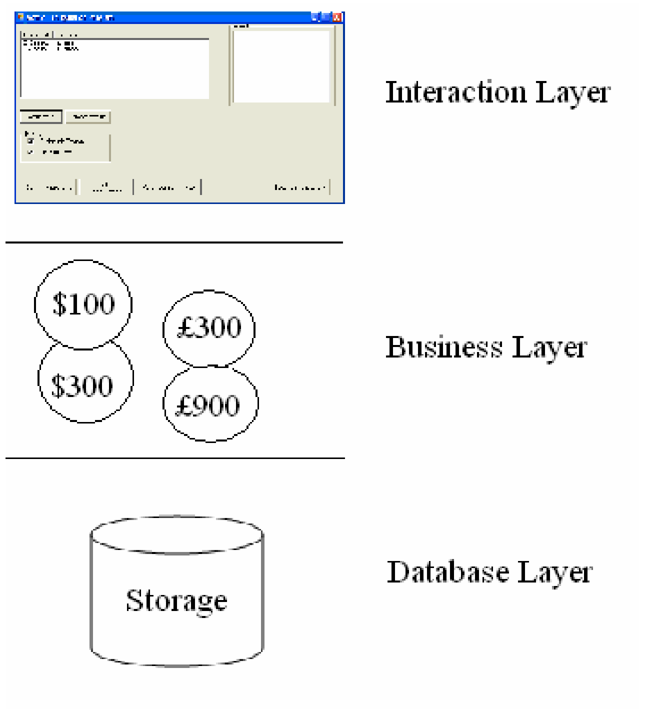


Figure 3.4. The 3 layered architectural design.

The top is the presentation layer (GUI), in the middle the business logic and in the bottom the persistent storage layer (DBMS).
In this architecture each layer is expert in its own area (divided by assignment) and does not worry about how the other layers function. When dividing the architecture in layers the entities are no longer in the same pool and the communication between them should be structured. If they all communicated uncontrollably with each other across the layers there wouldn't be any advantages in making this division in the first place. Instead they should communicate in structured manner and through some nice interfaces. Here is where the design patterns come in action; they present a solution to make the design more robust through methods that were used and tested earlier by others. Take for example the access to the database layer from the business layer, here the Façade Pattern s used to provide a single entry to the database layer; more about this under the implementation in section 3.5.4.1 Database Façade. One of the mentioned benefits of making this division is that it gets easier to test the software. During the development I have tested the functionality of each layer separately by giving the methods input and observing the output.
If it all were in one layer I would have had some major problems and delay in solving the errors because they would be part of a large chain. Also the testing would have to be postponed until I have implemented much of the system. One could argue that it is the same whether to test the whole system in the end or to test a small part ongoing; but one complete system presents many and larger errors whereas a small part presents few and smaller which are quickly dealt with.

3.4.3 Sequence diagram

This part shows the sequence diagram for the use case *Start new measurement* that I have implemented.

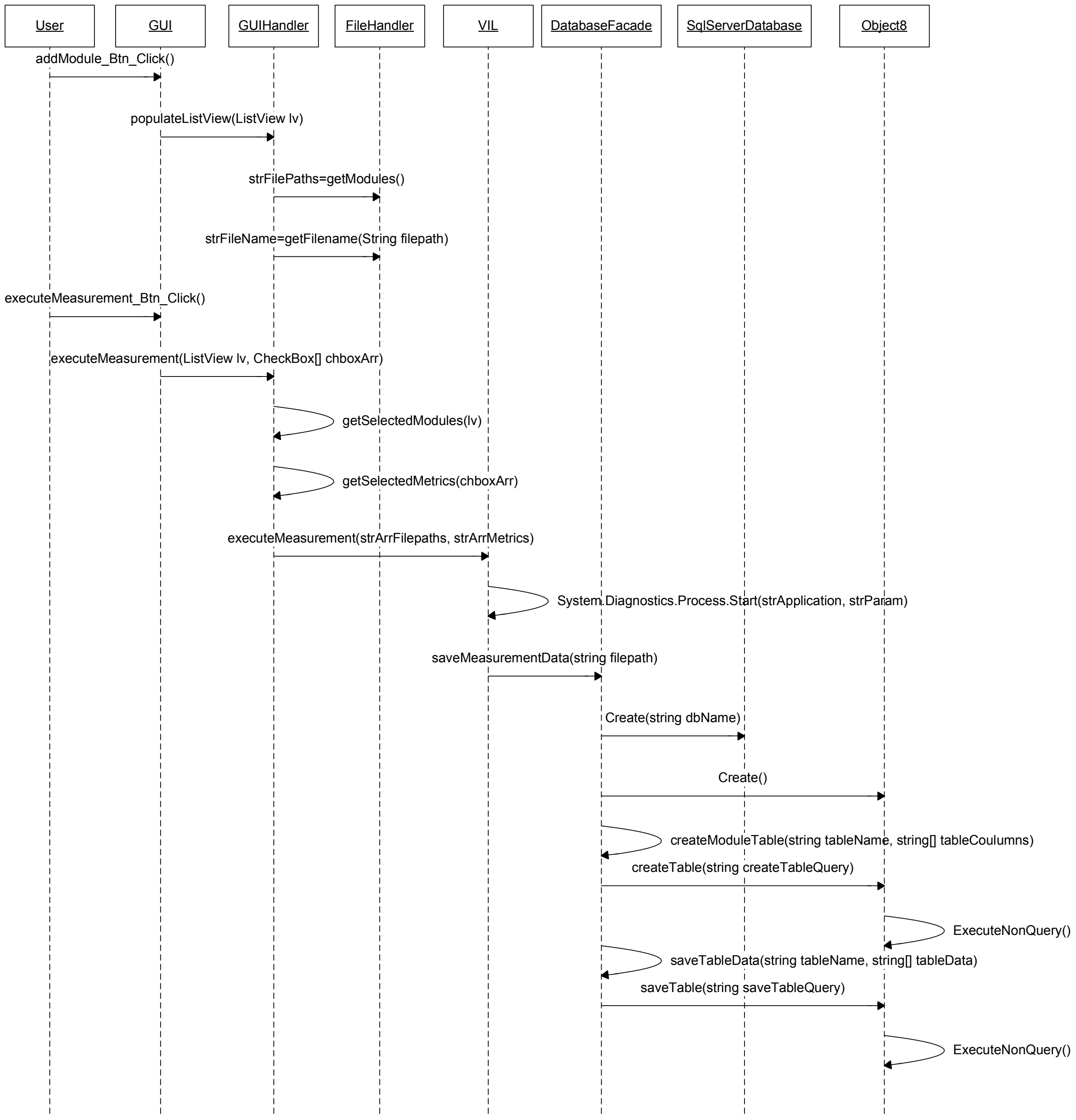

User
GUI
GUIHandler
FileHandler
VIL
DatabaseFacade
SqlServerDatabase
Object8
addModule_Btn_Click()
populateListView(ListView lv)
strFilePaths=getModules()
strFileName=getFilename(String filepath)
executeMeasurement_Btn_Click()
executeMeasurement(ListView lv, CheckBox[] chboxArr)
getSelectedModules(lv)
getSelectedMetrics(chboxArr)
executeMeasurement(strArrFilepaths, strArrMetrics)
System.Diagnostics.Process.Start(strApplication, strParam)
saveMeasurementData(string filepath)
Create(string dbName)
Create()
createModuleTable(string tableName, string[] tableCoulumns)
createTable(string createTableQuery)
ExecuteNonQuery()
saveTableData(string tableName, string[] tableData)
saveTable(string saveTableQuery)
ExecuteNonQuery()

Diagram 3.1. The sequence diagram *start new measurement.*

3.4.4 Class Diagram for Metric configuration program

Here I have inserted the class diagram for the system. It is rather self-explanatory and it shows the architecture and relation between classes.

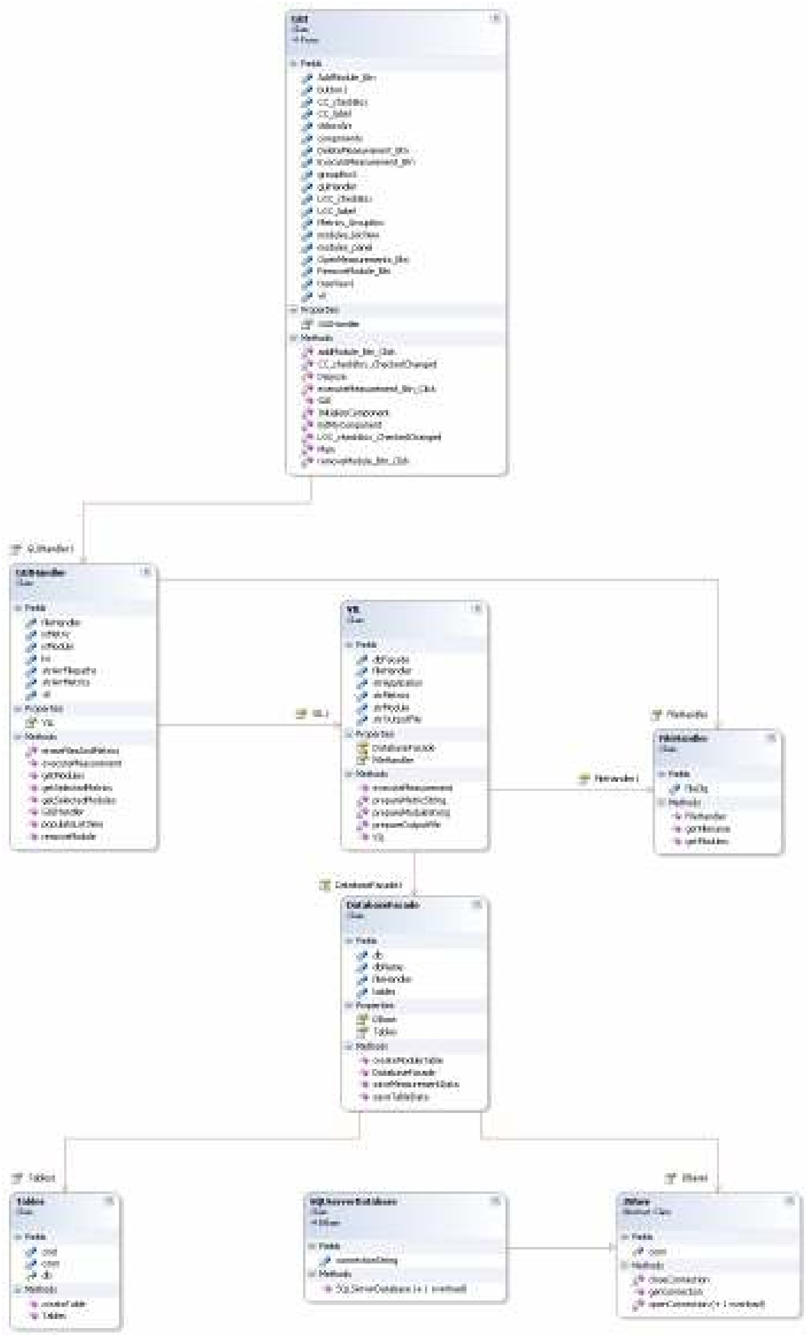


Diagram 3.1 a class diagram for metric configuration program. It also shows the layered architecture of the system horizontally.

3.4.5 Database design

The database contains tables to store the information of the measurements that have been computed. It needs to store the basic information for a measurement for example the time and date but more specific the modules that are contained in a measurement and most important, the actual contents of the measurement on a module. I could just create one large table and fill all the information in it, but then I will have some major problems when the database grows and it will be a very difficult to maintain, that is why I have created them in smaller tables that are easy to maintain.
For every module that has been measured a table is needed to contain the result of that module. As the same module can be measured on several times in different measurements I need to store the result for every module in each table. These tables are named the same as the module itself with the time and date as extension. This is of course done because I would like to save all the tables and then they should differ from each other by the name. As a measurement can contain one or more modules a table is needed for capturing which modules a measurement contains.
This is only one large table and it can have a constant name like: *modules_measurement*.
For being able to know some basic information on when a measurement has been performed and on what build version it has been executed on a table for storing the time, date, build version and measurement id I also needed.
This is 3 tables in total that are required for being able to store the wanted and relevant information on the measurements that have been performed.

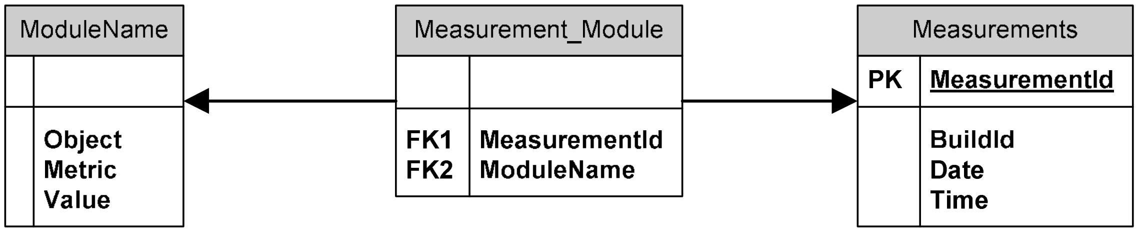


Diagram 3.2 a diagram of the tables in the database. This is only meant as a graphical representation of the tables and do not show the individual fields and their types.

3.5 Implementation and Test

3.5.1 Purpose

The purpose of this part is to give a solution to the how the design should be implemented.
I'm not going to explain every method in each class but only the ones with some special content, which requires elaborate explanation.

3.5.2 Interaction layer

The interaction layer is the composed of the GUI class and the GUIHandler class. Together they complete the task of interaction with the user and underlying system.

3.5.2.1 GUI

This class contains all the graphical user controls like buttons and list views. Its sole purpose is to present the user with the possibility to interact with the system and to redirect commands to the GUIHandler class. The GUI class is sort of a visual thing that the user can see and doesn't contain any business logic. It initializes its own components in the *initializeComponent* method, like setting the size and location and the *initMyComponent* method in which I have some settings for a list view and some checkboxes.
It also contains event handler methods for the relevant controls which then call methods in GUIHandler with some parameters for further processing of these.

3.5.2.2 GUIHandler

This class is the controller for the system. It receives commands from the GUI class and executes them. It is responsible for the further interaction between the user and the rest of the system. The GUIHandler class knows how to fill the GUI class' controls; it receives the control as a reference and does the appropriate action.
It contains methods for getting the modules and metrics from the GUI class and for passing them on to the VIL class.
The methods are rather self-explanatory and supported with appropriate comments but I will just explain the ones with some kind of special content e.g. *executeMeasurement*.
This method takes 2 parameters, a list view and an array of checkboxes. The list view contains the modules to be executed and the checkbox array contains the selected (checked) metric from the GUI class.
It starts out by extracting the content from these controls, as the GUI class only calls this method with the controls as parameters and not the actual content (the GUI class doesn't know anything about the content) when the user click the *execute measurement* button. This is done by calling each method to get the modules and the metrics.
When the modules and metrics are extracted it checks to see whether they contain anything or are empty. If both of the modules and metrics contain something it then calls the *executeMeasurement* in the VIL class with these as parameters.
In the end it erases the list view and checkboxes in the GUI class so no conflicts can occur in the following measurements.

3.5.3 Business layer

This layer is composed of the VIL class and FileHandler class. Here is where all the business is taking place.

3.5.3.1 VIL

This class is the actual class that has to do with the execution of the measurements against the command-line program VIL. It receives all the selected modules and desired metrics from the GUI class via GUIHandler and is in charge of constructing the parameter string in the right format. It contains an object of the DatabaseFacade class in order to communicate with the database.
The interesting method in this class is the *executeMeasurement* which is in charge of the actual metric computation. It calls the VIL command-line program with the modules and metrics to execute. To execute an external program from a C# program, a parameter string must first be constructed. Furthermore the VIL command-line program requires some arguments in the right syntax; these are first prepared and concatenated in a whole string before the execution.
To execute an external program I have used the *Start* which is placed in the *System.Diagnostics.Process* namespace. This method has 5 overloads and I have used the one with two parameters; a program to execute (the VIL program) and the arguments (the modules and metrics in one string).

3.5.3.2 FileHandler

This class is a helper class for general file handling. When the user wants to ad modules to the gui list view the method *getModules* is called which opens a File Dialog which also only allows files with the extension .dll to be selected, finally it returns an array of type string containing the selected modules, which then are filled in by the GUIHandler class.
It also contains other *get* methods to return the filename from a whole filepath.

3.5.4 Database layer

This layer contains classes that concern the database communication. It contains a databaseFacade class, which facilitate and simplifies the database access from the business layer, classes that construct the database server and a class that executes command to create tables and to insert data.

3.5.4.1 DatabaseFacade

This class is the (entry) to the database. It is responsible for letting the business layer accessing the database layer in an easy and well-defined manner. This is what the Façade Pattern is all about. The basic idea of the Façade pattern is to provide a unified entry point to access a set of underlying components. This makes the subsystem easier to use because we now have only one place to access and in that place the subsystem is "wrapped" and known. This reduces the complexity to accessing the subsystem. Figure 3.5 shows an example of how the Façade Pattern is used.

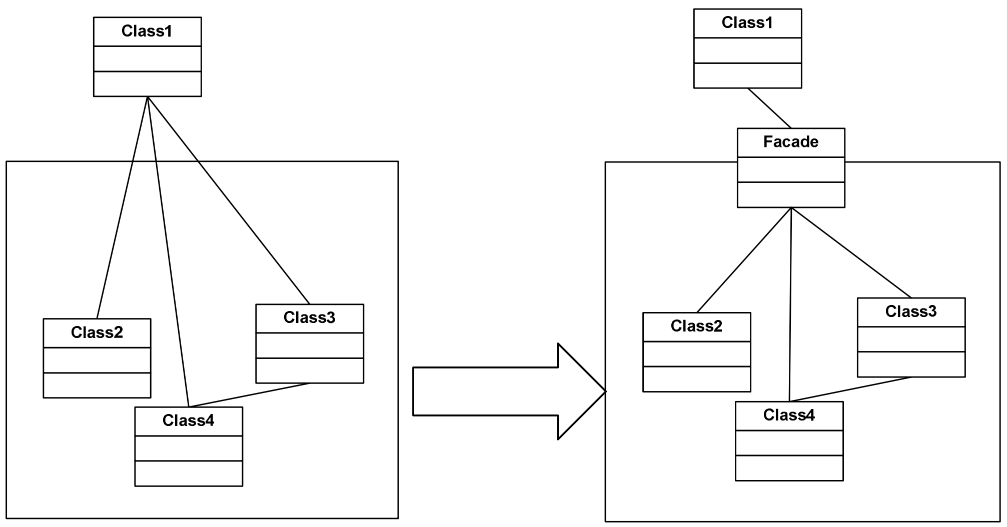


Figure 3.5 a representation of the façade pattern.

As figure 3.5 shows, the coupling between the class that accesses the subsystem and the subsystem is also affected in a positive way. We now have a low coupling that allows us to makes changes in the subsystem without the need to change the upper class.

It has a method called *saveMeasurement* which takes an output filename as a argument and reads the file and calls two methods, one for creating the respective table and one inserting values in this table which. These methods then create the proper sql queries and calls methods in the Tables class to execute the query.
When a measurement is finished VIL creates an output text file, which has the following format:

CC	LOC	NAME
10	200	Method1
5	100	Method2

The first line indicates which metrics that have been selected and the next line is the values.
The selected metrics comes in the first columns and last comes the method on which these values are computed. I have chosen to save the last column first in the table and this is also the primary key and therefore I start out by inserting this column first and then the rest with the metrics. As I read one line from this text file at a time and split into an array I simply just get the last array element and this first and then the rest. This all goes into a query string and then on to the method in the Tables class that executes this query.

3.5.4.2 Dbase

This class is an abstract class and is used to create a database connection. It contains methods to open a SQL connection with close a connection and to return the current connection. It can be expanded to also work with OleDb connections.

3.5.4.3 SQLServerDatabase

This class is the implementation of a SqlServer database. It inherits from the Dbase class and creates a connection towards a SqlServer database. It has to constructer overloads, one which only takes the name of the database to create a connection against, and one which takes some more parameters for also specifying the servername userID and password. Both then calls the base class' *openconnection* with a connectionstring object.

3.5.4.4 Tables

This class is the one that creates tables and inserts the measurement result into these tables.
This class contains only one method at this time, which is *createTable*.
The name of this method is a bit confusing as it is used for creating a table but also for inserting data into tables as it only executes a non-query and therefore can be used for both.

3.5.5 Test

During the implementation phase I have tested my program by basically giving it input values and then verifying the output. When a functionality was completed I tested it with inputs according to the respective use case. I can't say for sure that program is bug-free but with the right inputs it gives the right output and behaves like it should. I have some error handling in my program but as the time ran up I haven't verified that it can cope with every value. Of course a better and more elaborate test is needed but I have focused at getting a small piece of the program to work and demonstrate albeit small some working functionality.

4 Project conclusion

4.1 Chapter summary

In chapter 1 I have given a preview to the problem that I would work with and some basic explanation of the foundation and background of this project. In chapter 1 I have also given quick preview of quality measurements in other fields of engineering and described why measuring on the quality of software is also an important field of software engineering together with the other parts in a software development process.

In chapter 2 I have expanded the theory about software quality and divided the abstract term of software quality into smaller tangible parts which, can be measured by different software metrics.

I have expounded some of the different software metrics that are relevant to object oriented development and given some examples of their practical usage.

In chapter 3 I have laid the surroundings for a case study that should exploit the possibility of setting up a system where software metrics could be easily extracted and processed.

As this project was made in-house at MDD and as a bachelor’s project there hasn’t been any need for analyzing whether or it could be feasible to do or making any cost estimates as I would have done in a commercial project.

4.2 Conclusion

In this thesis I have:

- Analyzed what software quality is and how different stakeholders interpret it.
- Described how to measure on a software product by using software metrics and given a preview of some practical usage and possible ways of how the result could be interpreted.
- Contributed to shedding some light on this fuzzy area of software quality
- Shown that it can be possible to piece together a system to support the extraction of software metrics.

Software metrics provides a quick feedback for software designers and managers. Analyzing and collecting the data can predict design quality. It can lead to significant reduction in costs of the overall implementation and improvement in quality of the final product and in turn reduces the future maintenance cost.

4.3 A broader perspective

As time easily tends to be tight in any project I haven't completed everything that I hoped to and I have, during the project, been forced to limit the tasks, because if I didn't limit myself then I wouldn't have been able to write a fitting description of the most important topics.
In the beginning I was aware of some of the risk that could arise but in the end I can see that I haven't carefully dealt with these in the best way possible, as they seemed to push the time schedule.
Even though I knew that the subject software quality was the essence of my thesis and that it was very intricate I still needed to make an extra effort to clarify and end the subject and that was the main reason why the time became so squeezed.
This had an impact on other things like the implementation and documentation.
I could see that I wasn't going to be able to finish the implementation so this was put on hold.
Near the end of the project period I decided in consultation with my DTU supervisor to focus on the documentation and the report and stop with the implementation as it would require a great effort and time to complete it, which I didn't have.
I prioritized a better report instead of a complete working system as the system could always be extended with more functionality so the last weeks of my project time was dedicated to getting the report completed.
If I would have done anything differently I would have started out by focusing on the main subject, software quality, instead of the implementation.

4.4 Extensions and future additions

The system can receive modules and metrics (CC, LOC) and execute a measurement on these and save the result of one measurement in one table in the database. The computation of other metrics is easy extensible as it only requires that these should be added to the array of supported metrics. As for now the program only creates tables to save the result for the measurement of a module and do not store information on the measurementId, buildId, date and time. These are just formalities and can easily be created when the data for a measurement is about to be saved.
A part from the extension of other functionalities and implementing more use cases in later iterations are other essential extensions that could be implemented in the future for example, that the system could be connected with the development process so that when ever a system build is finished the system automatically get notified and start a measurement on these modules.
Other extensions are that a webpage is developed solely for the purpose of metric presentation and evaluation for the whole company.

5 Bibliography

5.1 Publications

[1] Object Oriented Software Metrics
Mark Lorenz, Jeff Kidd
ISBN: 0-13-179292-X

[2] Object-Oriented Metrics, Measures of complexity
Brian Henderson-Sellers
ISBN: 0-13-239872-9

[3] Best Practices in Software Measurement,
How to use metrics to improve project and process performance
Christof Ebert, Reiner Dumke, Manfred Bundschuh, Andreas Schmietendorf
ISBN: 3-540-208679

[4] Software metrics,
A rigorous approach
Norman E. Fenton
ISBN: 0-412-40440-0

[5] Principles Of Software Engineering Management
Tom Gilb
ISBN: 0201192462

[6] Chidamber Shyam R., Kemerer Chris F.: A Metrics Suite For Object Oriented Designs.
http://www.pitt.edu/~ckemerer/CK%20research%20papers/MetricForOOD_ChidamberKemerer94.pdf

[7] APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design and the Unified Process
Craig Larman
ISBN: 0-13-092569-1

[8] Programming Microsoft Windows With C#
Charles Petzold
ISBN: 0-7356-1370-2

[9] Software Engineering
Principles and Practices
Hans Van Vliet
ISBN: 0-471-97508-7

5.2 Websites

[10] www.1bot.com

This is the homepages of the developers of VIL program. It contains some information on the usage of the program.

A. Appendix

A.1 Use cases

USE CASE 1	Start new measurement	
Goal in Context	An actor accesses the system, select the wanted modules and starts the measurement	
Scope	How to start a new measurement	
Preconditions	The measurement program is installed and running, modules are present for measurement	
Success End Condition	A measurement completes and the result is ready for further processing	
Failed End Condition	The measurement can not begin/complete	
Primary Actor	Software metric analyst, QA manager	
Secondary Actor	Measurement program, metric program	
Trigger	A systembuild is ready or a measurement is requested	
DESCRIPTION	**Step**	**Action**
	1	User selects the desired modules
	2	User select the desired metrics
	3	User starts the measurement
	4	System saves the measurement and responds with an acknowledgement
EXTENSIONS	**Step**	**Branching Action**
	1a	No modules are present: A systembuild is not ready yet
SUB-VARIATIONS		**Branching Action**

USE CASE 2	Save measurement	
Goal in Context	The measurement data should be saved	
Scope	How to save a measurement	
Preconditions	The measurement program is running and a measurement has finished	
Success End Condition	The measurement data has been stored successfully	
Failed End Condition	The measurement data has not been saved	
Primary Actors	Software metric analyst, QA manager	
Secondary Actors	Measurement program, metric program	
Trigger	The measurement has finished	
DESCRIPTION	**Step**	**Action**
	1	Measurement to save is selected
	2	The system saves the measurement data in database
	3	The system responds with an acknowledgement.
EXTENSIONS	**Step**	**Branching Action**
	3a	The data could not be saved Database error: Contact system administrator

USE CASE 3	Delete measurement	
Goal in Context	The measurement is deleted	
Scope	How to delete a measurement	
Preconditions	The measurement program is running and a measurement has been saved	
Success End Condition	The measurement has been deleted successfully	
Failed End Condition	The measurement could not be deleted	
Primary Actors	Software metric analyst, QA manager	
Secondary Actors	Measurement program, metric program	
Trigger	A measurement should be deleted	
DESCRIPTION	**Step**	**Action**
	1	User clicks view measurements
	2	User selects the actual measurement to delete
	3	User click delete measurement
	4	System deletes the measurement and responds with an acknowledgement
EXTENSIONS	**Step**	**Branching Action**
	2a	No measurements are present: Measurements should be executed before deletion
	4a	Measurement could not be deleted: Database error: Contact system administrator