FAKULTÄT FÜR !NFORMATIK

Balancing Quality Assurance Methods along the Project Life Cycle: A Concept for Systematic Quality Assurance Strategy Evaluation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Werner Rieschl

Matrikelnummer: 0026991

an der Fakultät für Informatik der Technischen Universität Wien

Betreuung:

Betreuer: Prof. Dr. Stefan Biffl

Mitwirkung: Dipl.-Ing. Dietmar Winkler

Wien, 21.10.2008

__________________________ __________________________

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Technische Universität Wien

A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43/(0)1/58801-0 ▪ http://www.tuwien.ac.at

ERKLÄRUNG ZUR VERFASSUNG DER ARBEIT

Werner Rieschl
Embelgasse 5-7
1050 Wien

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 21.10.2008

ACKNOWLEDGEMENT

I would like to thank Dietmar Winkler and Stefan Biffl for pointing me in the right direction and guiding me through the work on my diplomacy thesis. He also supported me to gain access to some scientific documents.

Furthermore I want to thank my girlfriend Silvia who supported me all the time, Marius Eder who give me useful inputs regarding the development of the prototype and Michael Pedigo who corrected the proofs of this thesis.

My parents for their permanent support during my study.

Thank You!

ABSTRACT

Software products exist in most areas of our life, e.g. in home entertainment equipment, for water supply control, in fully automated production streets and many more. High quality software needs verification and validation to provide compliance with specification documents and customer requirements. Today there are various methods to support construction, verification and validation of software products, e.g. several development methods, reviews, software inspection and testing approaches. The application of one individual method focuses on the specific document type but rarely considers information from previous project phases.[52] Thus, it is reasonable to combine quality assurance methods in order to provide a high quality product along the product life cycle. Quality assurance strategies, e.g. bundles of balanced quality assurance (QA) techniques, aim at improving over product quality. These approaches can be summarized under the headline of software product and process improvement.

Every project has special characteristics which can be used to classify them into specific groups with related attributes (e.g. project size, duration) and affect best practice method selection. Up until now some quality methods are more applicable for particular project types than for others.[17] For instance, it is at hand that testing is more effective on pieces of code than on an architecture document. The selection of an appropriate method depends on the project manager or maybe on company standards as well. The project management's decision is usually based on the experience of past projects with the same team and personal knowledge.

The purpose of this work is to include an overall point of view in that decision processes, e.g. the consideration of the whole product development lifecycle. Therefore, combining different analytical quality assurance methods addressing special types of software projects, special types of quality attributes and company strategies will multiply the positive effect for this venture.

Therefore it is important to realize the need of mapping project related data and candidate methods according to selected criteria which can have an impact on product quality in every project. The key aspects of different quality assurance methods are related to these project elements and therefore they can be compared with the help of predefined values. The main idea is to create a vector of attributes for different quality assurance methods. These attributes can be converted into discrete values to generalize them for comparison with the attributes of different project types.

Because of these reasons this thesis presents an attribute based approach to match different project types with various combinations of quality assurance methods to get an adjusted support for any project. This idea deals with SPPI – Software Product and Process Improvement, as said above. Some metrics will be introduced to compare different projects and methods. These measurements will be necessary for improvement purposes by providing feedback from the project application. It is

necessary to collect data of project types, used methods and the qualitative outcome of any collected project.

This thesis deals with the construction and evaluation of an overall quality strategy that can be adjusted to different project types, to support project and quality managers with their decision making. It should be an approach to fill the research gap of finding the best quality assurance strategy to different software development methods. So we have to look at different software development methods to identify some overall project characteristics.

Therefore a method should be implemented to describe and rate some basic quality assurance methods and different project types, to deploy a framework of attributes to classify different project types and to indicate a way to compare these two approaches to get a feasible solution for attribute based quality decision making. To provide a better view a prototype of this framework will be created. The proposed approach aims at improving products and processes by providing an agreed set of methods.

ABSTRACT - GERMAN

Software Produkte begleiten uns überall in unserem täglichen Leben, beispielsweise im Unterhaltungselektronikbereich, bei der Wasserversorgung oder in voll automatisierten Produktionsstraßen. Qualitativ hochwertige Software muss gemäß den Spezifikationen und Anforderungen mit Hilfe von verschiedenen Entwicklungs- und Qualitätssicherungsmethoden validiert und verifiziert werden. Die Anwendung einer speziellen Methode konzentriert sich meist auf die Evaluierung eines speziellen Dokuments, deren Ergebnisse werden aber selten als Basis für weitere Qualitätsmaßnahmen herangezogen.[52] Daher scheint es sinnvoll verschiedene Methoden in unterschiedlichen Projektphase zu kombinieren, damit deren Effektivität und somit die Qualität des Produktes steigt. Unter dem Schlagwort der Software Produkt und Prozessverbesserung ist daher die Entwicklung von Qualitätsstrategien sinnvoll.

Jedes Projekt weist spezielle Charakteristiken auf, die in verschiede Gruppen von Eigenschaften eingeteilt werden können (z.B. Projektgröße oder Dauer) und die sich auf die Auswahl von bewährten Methoden auswirken. Einige Qualitätsmethoden sind für spezielle Projekttypen und Artefakte besser geeignet als andere.[17] Beispielsweise ist es klar, dass Testprozesse besser auf Code – Teile angewendet werden können als auf ein Architekturdokument. Die Auswahl von geeigneten Methoden bleibt meist dem Projektmanager überlassen, der seine Entscheidung auf Grund von persönlichen Erfahrungen oder Unternehmensstandards fällt.

Das Ziel dieser Arbeit ist, den Entscheidungsprozess von einem umfassenden Standpunkt aus zu betrachten, in dem das gesamte Projekt miteinbezogen wird. Dies kann über eine Kombination von unterschiedlichen Qualitätssicherungsmethoden, die sich mit speziellen Aspekten von Software Projekten beschäftigen, erreicht werden, wodurch sich die positiven Effekte dieser Methoden erhöhen.

Aufgrund der oben genannten Aspekte ist der Vergleich von projektrelevanten Daten mit verfügbaren Qualitätsmethoden über definierte Kriterien notwendig. Die Qualitätsmethoden müssen nach denselben vordefinierten Werten klassifiziert und eingestuft werden, um sie mit den Projekteigenschaften vergleichbar zu machen. Dies kann über einen so genannten Attributsvektor erfolgen, in welchem die Kriterien gesammelt und in diskrete Werte umgewandelt werden.

Diese Arbeit beschäftigt sich also mit einem Attribut basierten Ansatz um verschiedene Projekttypen mit verfügbaren Qualitätsmanagementmethoden gezielt vergleichen zu können, damit die Projektverantwortlichen eine automatisierte Unterstützung bei der Auswahl der geeigneten Methoden haben. Diese Denkweise fällt unter den Begriff der kontinuierlichen Software Produkt und Prozessverbesserung, wie bereits eingangs erwähnt. Zur Umsetzung müssen grundlegende Metriken definiert werden, um eine gemeinsame Vergleichsbasis zu schaffen und um die Entscheidungskriterien auf Grund von neu gewonnenen Erfahrungen gezielt anpassen zu können.

Weiters beschäftigt sich diese Arbeit mit der Entwicklung und Evaluierung eines Prototyps zur Bereitstellung von Qualitätsstrategien, die sich aus einzelnen Qualitätsmethoden zusammensetzen und den Projektverantwortlichen in der Entscheidungsfindung unterstützen sollen. Dieser Ansatz soll eine Brücke zwischen den Entwicklungs- und Qualitätsbereichen von Softwareprojekten bilden, um die Effektivität und die Effizienz von Softwareentwicklungsprozessen zu optimieren.

CONTENTS

1 INTRODUCTION

Nowadays in most electronic devices an internal logic is needed to provide their functionality. At the very beginning “easy” logic was realized with electronic circuits. With the growing complexity and the triumphal procession of the microprocessor, the necessity of effective, complex and almost error free software appeared. But software is a very general term. There are different application areas for software. Subcategories are operating systems which help to connect the hardware with the software, support tools which help to maintain the computer or the network and application software. The purpose of the last type of software is to fulfill the practical requirements of the user, like writing a letter with the help of an office application or developing graphics with a design application. As we can see the variety of software is very high.

The goal of every software engineering company is to produce products of good quality in short time with a minimum amount of money. Because of that the usage of methods to achieve quality assurance is one part of the software engineering process with a very special focus. But what is the main purpose of quality assurance in general? The answer to this question can be found in the definition of quality itself. But what is quality? Quality can be defined as the objective to meet the requirements of all stakeholders involved in the life cycle of the software.

But we don’t have to forget that requirements are not always that clear as they seem to be. Some requirements are not written down or clearly communicated to the vendor of the software because from the view of the customer these requirements are so self evident that he never conceived the idea to record them into a specification or a requirements document. Therefore communication between customer and vendor is very important to ensure that the software development process meets the conceivabilities of the customer during the whole project. At the end of the day we can say that quality assurance is the process to ensure that the developed product meets all necessary requirements and in addition that satisfies the customer and the users.

Various methods with different approaches were developed to increase the product quality. The different approaches aim for several stages during a software development project because the deliverables differs from stage to stage. Quality assurance methods are performed during the whole project life cycle from the very beginning of the project until the deployment and maintenance phase. Due to the fact that these phases are very different, the structure and the output of various quality assurance methods are also very different. It seems reasonable that the achieved information of the conducted quality assurance methods should be beneficial over the complete project life cycle.[50] This means that the knowledge out of each single method should be collected and used as input for other quality assurance methods to gain a maximized profit out of the methods. The current state of the art is that the outcome of a quality assurance method is only used for punctual improvements on the software but is no input for further development stages.[12] If we consider the fact that normally a small number of main risks and problems have to be monitored during the whole project life cycle to guarantee a product of high quality at the end of the day, it is necessary to use all discovered information regarding these main issues to ensure comprehensive knowledge about that. Just the treatment of these issues maximizes the effectiveness of quality assurance methods because the smaller issues can often be solved in one single development stage.

For further example reviews and inspection usually dealing with the first project stages which means that these methods are used for improvement of written documents like the requirements, design, risk analysis and so on. It is at hand that the output of these quality assurance processes is different to the output of some testing methods. Therefore these different methods can not be compared directly which means that there has to be another way to qualify these methods against several types of software development processes. The challenge of every project manager must be to transfer the quality assurance information out of the first project stages into quality inputs for the following stages to ensure that the first findings will not be forgotten at the end of the project.

Another important aspect is that the experiences of past projects should be preserved for upcoming ventures to avoid spending time and money on problems that have been already solved. This leads to a continuous improvement in the business process and the economical efficiency of any company. This argument is also one part of the software product and process improvement concept which should gain continuous progression.

This issue can be observed in daily business life. Every business process has its own weaknesses, bottlenecks and problems. Because of that fact several errors occur on a regular basis. Sometimes these errors look slightly different but with the help of a more precise analysis it can be seen that most of them are the result of very small problematic areas. There are some methods of business process improvement which are dealing with this type of problems to ensure that the company is getting rid of it. But another way of dealing with these problems could be the prevention with the help of the quality assurance methods from the practical point of view. To implement this approach a process of recording and prevention on the basis of the output of quality assurance methods has to be developed. This means that identified problems are not only a chance to enhance the quality of a product; they can also be a practical example and an indicator for a failure in a business process. If a problem in the structures of a company can be identified with the help of such QA issues preventive actions to improve these structures could be conducted and these actions are the requirements for more stable business processes in the future.

Today companies use the developed quality assurance methods punctually to verify the quality of the work done in a specific stage of a project as mentioned before. Since the foundation of software inspection in 1976 by Michael Fagan, much work was done in this scientific field and many various methods for quality assurance in all possible project stages were developed. Recently some scientists tried to give an overview of the whole range of techniques in this area. Oliver Laitenberger from the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern (Germany) compared most of the software inspection technologies in his paper in 2001.[32] He analyzed more than 400 references in his work and brought some aspects in a common vocabulary to get a better overview of this area. His motivation for this work was bringing the huge quantity of methods in a comparable form. If we consider all these aspects we have to agree that every quality manager has the agony of choice in finding the most suitable solution. This choice is never easy and currently there is no automated help to solve this problem.

So we can see that the position of the quality manager is not the easiest. The choice of the best methods depends on several factors like experience with different quality assurance methods, structure of the project, experience of the team members, different risk factors and many others. To make matters worse these factors have all different weightings which make the decision harder. Therefore the need for an overall taxonomy is given to provide a better decision basis for the software industry in this area. The difficulty of this approach is to make all different methods comparable to each other. So the necessity is given to identify core attributes of which are influenced by the quality methods in a different way which makes it possible to rank them.

The next step to support the quality managers is the development of evaluation software which is designed to qualify all available quality assurance methods against the parameters of any different kind of software development processes. The current state of the art is that there are several development processes with different approaches and mindsets but all of them have one goal: to develop high quality software in the most efficient way with a minimum amount of money.

After the burst of the .com bubble at the turn of the millennium every company tries to improve their business processes and to make their whole business more efficient. This means that they try to make their processes leaner and more resistant against external influences and unexpected problems. One result of this process is that the costs of software development shall be minimized while the quality of the developed piece of software shall be maximized. This implies that the software development processes and the quality assurance methods of these companies must be improved to reach this goal. Also the interaction between these two processes and the influence on the outcome is a very interesting issue. Both factors are directly interdependent which means that the increase of one factor automatically implies an increase of the second one. At the end of the day quality assurance methods can contribute a lot to the improvement of the operation processes of a company if the information gained during the usage of the quality methods is used to develop improvement proposals for the current working procedures.

But still a major aspect of this improvement process is the effectiveness of the used quality assurance methods. First they have to fit to the selected software development process. Second they have to be most effective to discover almost all errors with a minimum amount of time and money. Third the QA methods have to share their outputs to ensure permanent software product and process improvement over time.

This work deals with the named aspects of quality assurance methods for software projects, shows the improvable points and develops an approach for a tool based decision finding process regarding to the different types and attributes of software projects. It shall help to describe different methods and approaches with the help of general parameters which can be used for all software development processes and all quality assurance methods. The goal is the introduction of a new point of sight regarding the functionality and the possibilities of quality assurance methods. They must be seen as pieces of a chain which actuate the software development process. So the methods must be combined to ensure the best possible output because we all know that a chain is only as strong as the weakest part if it. Because of that fact the solution of our research gap deals with not only with single quality methods but with quality strategies which consists of a number of methods.

Overview of the Thesis

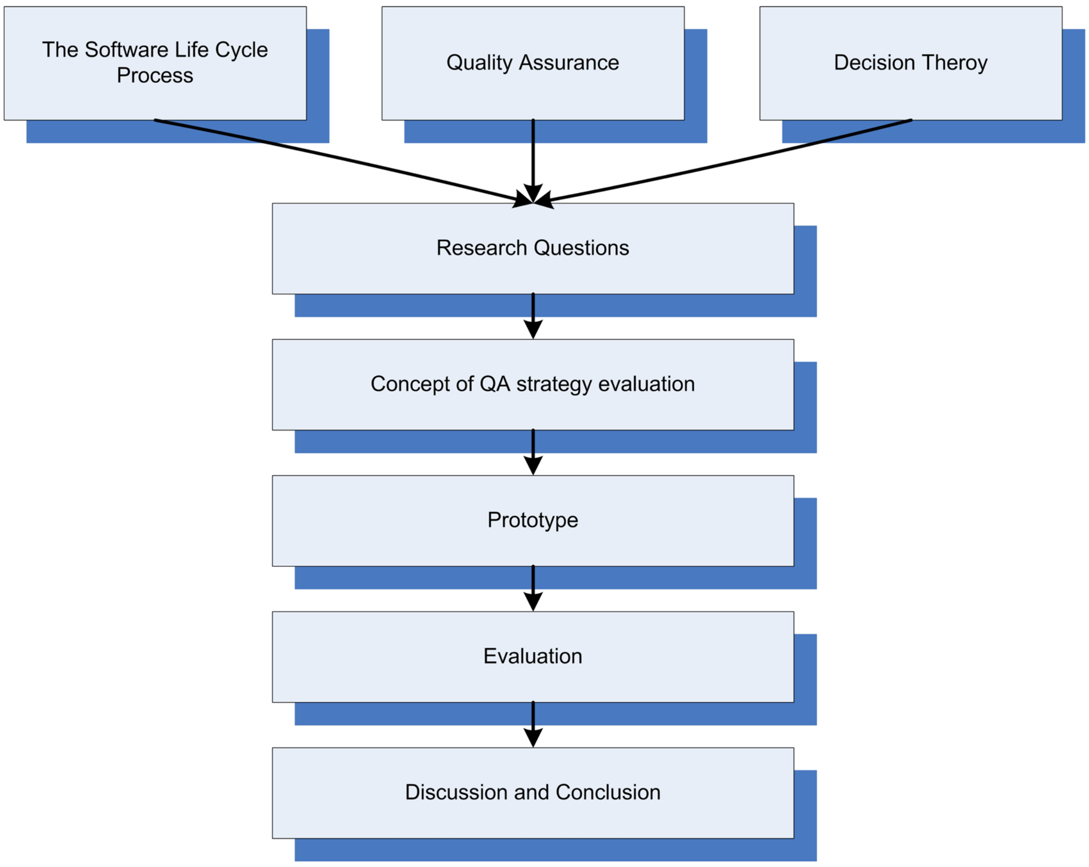

The Software Life Cycle Process
Quality Assurance
Decision Theroy
Research Questions
Concept of QA strategy evaluation
Prototype
Evaluation
Discussion and Conclusion

Fig. 1: Structure of the thesis

The figure above shows the general structure of the thesis and should help to navigate thru it.

The first three chapters illustrate the theoretical background of this thesis. They should help to get an overview of the theory that is used in the practical chapters. There is some information about the project life cycle and some general examples of project models of software processes, a brief introduction in some quality assurance methods and basics about measurement methods and decision theory.

With the basics of these theoretical chapters, the research questions are formulated to lead over to the practical part of this thesis. This chapter introduces the basic solution principles which seem to be necessary and hand over to the most suitable set of solutions in this area of research.

The next two chapters deal with the prototype. The goal is the implementation of several evaluation criteria for software development processes and quality assurance methods. With the help of weighted arguments the prototype provides a sequence of accurate quality assurance support over the whole software product life cycle to ensure continuous software product and process improvement. This is necessary to make business players advantageous in this area.

The last two chapters discuss the results of the thesis in relation to the research questions. They shall also give a short perspective for future research on this topic. Furthermore these chapters deal with insights gained during the implementation of the prototype and the possible problems and boundaries regarding the solution approach of the prototype.

This overview shall help to keep the reader on track, help him to navigate to the thesis and to jump to the most interesting parts from his point of view if he wants to.

2 SOFTWARE DEVELOPMENT PROCESS

The life cycle of any software project shows some of the same basic characteristics. A project typically starts with an idea or a vision to create something new or enhance an existing tool. In fact a need for a new tool is identified. This vision will be discussed in a small project group to get more different views of the existing problem and the proper solution. There will be developed some approaches to solve the problem. The most feasible one is picked up to enter the next project phase.

This phase is the development of the selected solution. The development is done with the help of a specific project model. Dependent on this model, artifacts like an architecture document, a design document or a risk document are created. These artifacts are needed to perform the implementation of the software, which is the following project phase.

The project is divided in modules and functions to allocate them to the programming teams. Parallel to the implementation starts the tests phase in which the functions and modules are tested according to their specification. Afterwards the deployment and maintenance phase starts. At this stage error handling and upgrades are important to guarantee good software for the customer.

This life cycle is a short version of the basic steps in the software development process. In the following sections some basic software development processes are illustrated to give an overview of the state of the art. The different models are the reason, why it is necessary to evaluate the different characteristics of a software development process.

2.1 ISO 12207 – SOFTWARE LIFE CYCLE PROCESSES

The International Organization for Standardization was founded in 1947 by a committee of 25 countries. Today 125 nations are in this organization. The business of ISO is the unification of products in various areas. Examples are radio frequencies, currency codes, date and time notation and also compression methods for audio and video files like MPEG.[59]

Software life cycle processes are regulated in the norm 12207. The purpose of this norm is the creation of a standardized framework for software development processes. In the norm the problem and the resulting need is described as the following:

"There is a proliferation of standards, procedures, methods, tools and environments for developing and managing software. This proliferation has created difficulties in software management and engineering, especially in integrating products and services. The software discipline needs to migrate from this proliferation to a common framework that can be used by software practitioners to "speak the same language" to create and manage software" – [23]

The norm is organized in seven chapters. The first three are scope, references and definitions. The fourth chapter is about the application of this standard and gives an overview about the three main chapters which describe primary processes (chapter five), supporting processes (chapter six) and organizational processes (chapter seven). This norm has to be seen as a framework for the companies and shall help them to create their ISO certified management system. Therefore the norm provides guidelines to produce software products of high quality.[42] The following figure gives an overview of the contents of this norm:

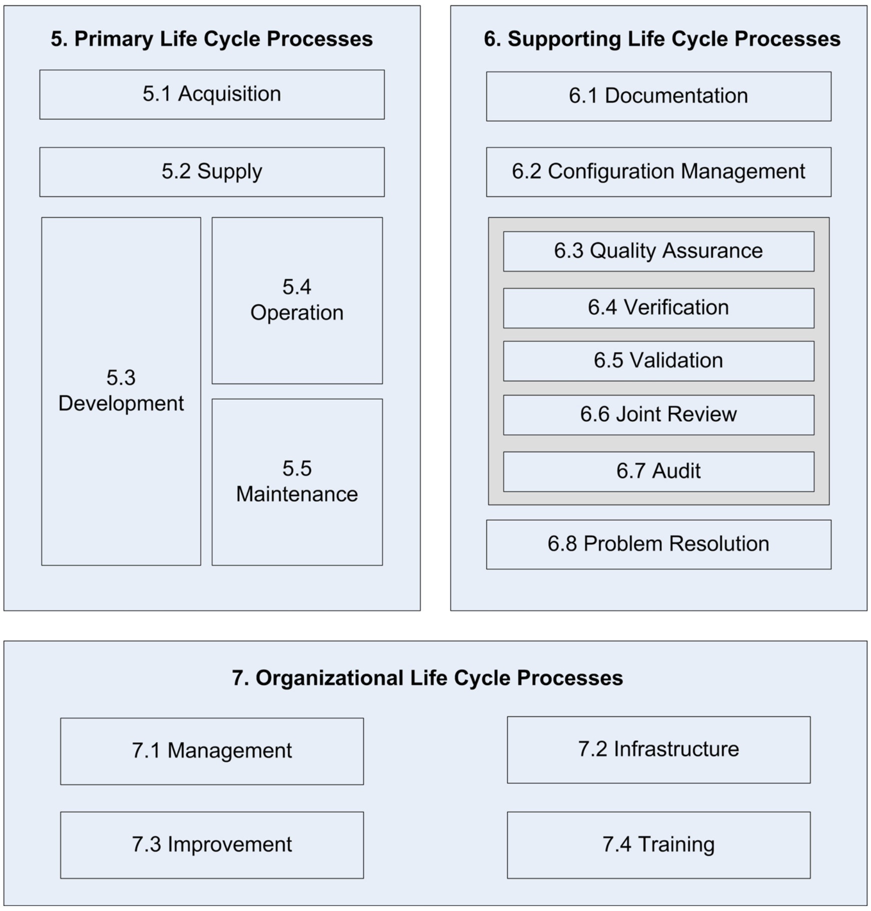


Fig. 2: Structure of the ISO 12207 [23]

2.1.1 Primary life cycle processes

These are the main processes for software development. As depicted above they are split into five sections. During acquisition the needs for the system are defined and evaluated. This is the initial phase of the project. The acquirer defines requirements, chooses suppliers and works on the details of the contract. This includes the package of deliverables, terms and conditions of the contract and the technical constraints.

The main tasks of the supply process are planning, execution, review and delivery of the system. In the planning phase documents like the work breakdown structure, risk analysis and the quality management plan are created. In the execution phase the supplier deals with the development, operation and maintenance of the product in accordance with the next three processes in this section. His responsibility is the monitoring and control of the progress and quality of the product. In the review and evaluation phase the supplier coordinates the review activities to guarantee an acceptable amount of quality of the product. He is also responsible for the communication to the acquirer and provides information of the progress of the development process.

During the delivery phase the supplier has to make sure that the delivered software complies with the requirements described in the contract. Furthermore he is responsible for the support of the acquirer after delivery of the product.

The development process can be seen as the core process. All activities of the developer are described here. Requirements analysis, design, coding, testing and integration happen in this phase. It is important to mention that this norm does not exactly regulate the way to perform these tasks. This means that there is no method defined like the waterfall model, only the required activities for any relevant strategy are listed.

The norm prescribes documentation, consistency, traceability and other similar requirements for the development process of certified companies. They also have to define communication interfaces between their departments to ensure an effective working environment. Several testing phases are also described in this norm to guarantee that all units, and their interaction and the whole system are functionally correct. In the operational phase of the product assistance and management of corrective actions shall provided as it was defined in the contract.

The processes of this section are supported by the activities in the sections six and seven.

2.1.2 Supporting life cycle processes

As the name implies these processes aid the primary processes. They are split into eight parts. The documentation process provides a guideline for standardized documentation during the development process. Configuration management deals with identification, control, status accounting and evaluation of the configuration of the product. In quality assurance the quality of the product and the development process are monitored. During verification the product is checked against the requirements of the customer at the beginning of the project. This phase includes verification work of the contract itself, the requirements, design, code, integration and documentation.

The validation process deals with the test configuration. It helps to ensure that the tests have the right scope which means that all critical parts of the software are tested carefully.

The joint review process monitors the status of the project and is held by two parties where one is the reviewing party and the other is the reviewed party. This process shall help to get an overview of the progress of the project. It is also a proofing tool for the schedule of the development process.

In an audit the work of the team is validated against the requirements, the contract and the project plans. Problems and deviations are recorded to guarantee that an appropriate solution will be found as soon as possible. This is also the target of the problem resolution process. This process provides guidance for a structured and documented problem resolution.

2.1.3 Organizational life cycle processes

These processes help to ensure that the main processes of the company work in an appropriate way. It consists of the management process, the infrastructure process, the improvement process and the training process.

The management processes give an overview of the responsibility of the companies' management. Their responsibility is planning and allocation of projects, control of execution and review and evaluation of the work.

The infrastructure process has to supply and maintain the infrastructure needed for any other processes. In fact a company is constrained to provide the whole working environment that is necessary to produce their products. This includes materials as well as employees.

ISO standards require a permanent improvement of the companies' processes. This is written down in chapter 7.3 of this norm. A certified company has to evaluate and improve their processes.

Last but not least the training process describes the necessity of trained personnel in a company. Therefore a training plan for the education of the employees is required.

This framework helps to ensure a repeatable and controllable development process which is important for decision making. This is done by the selection of quality assurance methods during a development process. Only a consistent working environment is able to guarantee the experience needed for the analysis of the effectiveness of a quality assurance method.

The ISO 12207 – Software Life Cycle Processes Framework can be seen a basis for every other software development process.[30] The description of the processes is very general. Only the important points are ruled with this framework. This guarantees that every ISO 12207 using company has the ability to implement their preferred working process into the ISO framework and receive the certificate for this norm. Because of that ISO 12207 provides a good overview of a general software development process and helps to understand the processes described in the following chapters.

2.2 TRADITIONAL AND AGILE SOFTWARE PROCESSES

Since the early stages of software development many different methods and processes have evolved. Today we can separate them into two basic groups.

On the one hand there are traditional processes like the waterfall model or the V – model. The characteristics of these models are a complex process and the creation of several different documents. The product is in detail planned at the beginning of the project and implemented and tested at the final stages.

On the other hand new development methods appeared in the last few years which called agile processes. The whole development process is less complex than the traditional ways, the methods are unbureaucratic and only few rules are given. The roots of agile software development are in the early 1990's and the popularity grew with the introduction of Extreme Programming in 1999 by Kent Beck. Nowadays more and more companies use agile software development processes.

Agile processes are built on the knowledge and the experience of the employees of a company. The main issue is working software, not their documentation and therefore the continuous interaction with the customer is very important to meet the requirements for the software. It is controversial if planned software development with the help of agile processes is possible. But the percentage of companies which use such methods is growing and therefore the acceptance is approved.

In the following sections are short overviews of some software development models of both categories. Afterwards the V – Model and Scrum are described more detailed since these two methods are used in the practical part of this thesis.

2.2.1 Waterfall model

The waterfall model is a non iterative software development approach which is organized in different sequential project phases. It was first mentioned in the book "Managing the Development of Large Software Systems" by Winston Royce in 1970. The name of the model proceeds from the graphical representation of it. The following figure shows the possible types of the model:

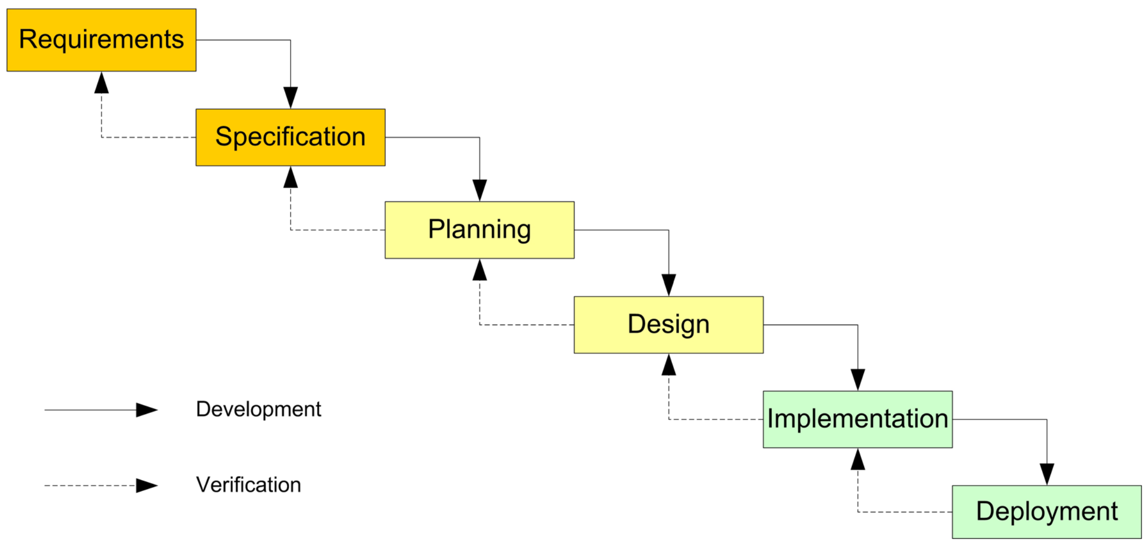


Fig. 3: The Waterfall Model [66]

As depicted it starts with the requirements analysis and ends with the deployment and maintenance of the software product. The output of a phase is the input for the following phase. Every activity ends with a predefined document. Therefore this model can be qualified as a document driven software development process.

This model is used in many companies and public institutions like the NASA or the Department of Defence of the United States. It is useful for software projects with clear and less changing requirements. The project stages are spitted and the planning of the projects is easy. But critics of the model say that the effectiveness is only given for small or non complex projects. Furthermore later changes of requirements are expensive because of the massive documentation at the beginning of the project.

There are several variations of this model to improve it and to solve some basic problems. Today some other models are more popular than this one but it is still used.

2.2.2 The V – Model

The V – model can be seen as an enhancement of the waterfall model and it is widely spread in the software development business. It has its roots in the late eighties of the last century. In 1997 the release of the last revision of this model took place. The model theoretically separates the specification phase from implementation and testing.

In the figure below we can see an overview of this development model. The process cycle starts at the left top of the V and ends at the right top. With the help of this notation the project can be split into three main phases except for the basic idea and the operation and maintenance stage. The more we are going down to the bottom of the V the degree of detail goes up.

On the first level of detail we can see the user view of the software. At the beginning of the project this view is manifested in the specification of the system and at the end of the project the user has to deal with the finished software. These two parts of the system are linked with the acceptance tests which check the functionality of the software on the application level according to the needs of the users.

The architectural view of software is settled one level deeper. This view deals with the system design one the one hand and on the other are the design tests the system tests and the integration tests. This is the level where the single features of software are put together to a whole. And at the beginning of the process the idea of the customer will be spitted into useful modules to gain the full complexity of the application.

The lowest level of this model is called implementation view and deals with the realization of the software. On the planning side the module are specified and on the practical side the modules are tested. Between these two steps the coding of the software is done. As we can see in the figure the model has a well defined structure for the development of the software with connections between the planning and testing phases of the application. Therefore it is a good way for analytical software development.

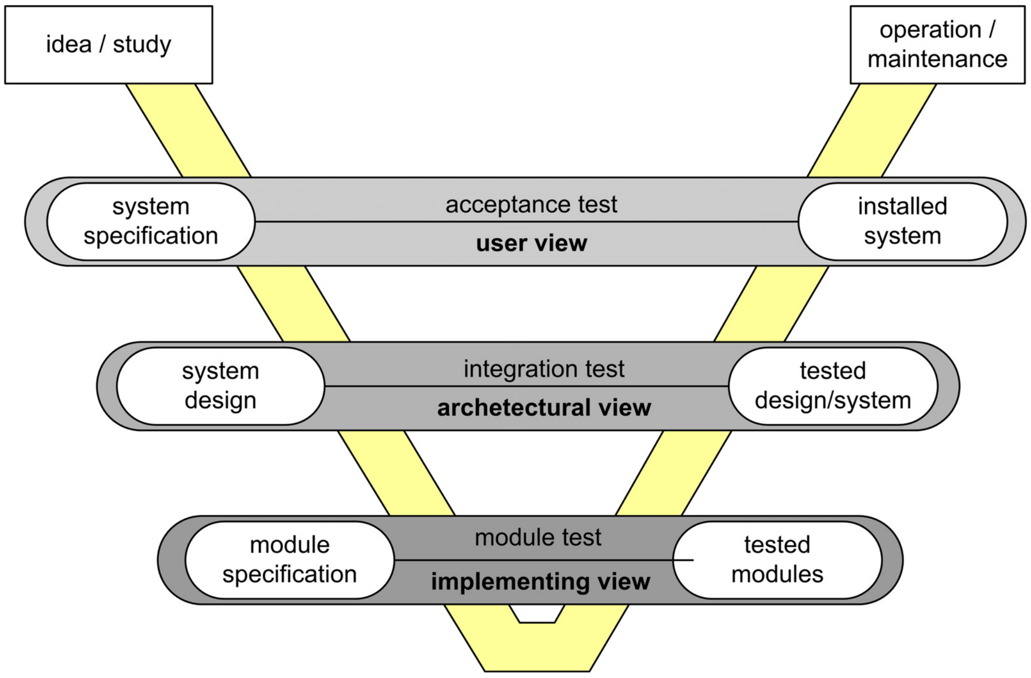


Fig. 4: Overview of the V – Model [66]

Many different figures of the V – model are described in various references and all of them look a bit different. Some are overloaded with information others are very puristic. But the basic concepts of this model are everywhere the same. The figure above is from a lecture of software processes of our institute and it gives a very good overview of the main things of this model. As we can see the quality assurance methods which are figured as the gray bars represents the connection between the planning side and the implementation side of the "V". This underscores the importance of quality assurance in the software development process.

This model provides an analytical and structured way to produce software. It is very useful for big and complex projects to keep an overview of the whole development process which is normally split into single parts of responsibilities.

The V – model is a representative of the group of traditional software development processes. It intends iterations of the single steps for ensuring a better level of quality. It is widely spread in the business area of software development and there are many references in the literature of this model.

However if the process is done carefully replicability of the work for the customer is very good and an experienced development team can plan a project very exactly relating to project time and costs.

Deprecators of this model are of the opinion that it is too inflexible for today's software development. These people prefer agile software processes. Therefore they also think that too much effort is laid on the planning phase in this model.

It is quite correct that a discovered mistake late in the development process can be very expensive. Because of that fact a well done planning phase is extremely essential to avoid such mistakes. But if the customers requirements change very often it is hard to gain an effective output in an appropriate time with a minimum amount of costs.

This model will be playing a major role in the practical session of this thesis because there are some attributes which can be used as criteria for qualifying a software development projects. As mentioned before it is advancement of former processes. All criteria of these can be also found or implemented in this process model. Therefore the V – model will be the representative for the traditional software development processes.

2.2.3 V – Model XT

The V – model XT is a derivative of the basic V – model from 1997. It was introduced in 2005. "XT" stands for "extreme tailoring" which means that the model is adaptable and scalable to any respective needs of a project. It represents the state of the art relating to technological standards.

The goals of this system are minimization of project risks, reduction of overall costs of the system life cycle improvement of process and product quality and intensive communication with the customer. In Germany for instance this model is mandatory for any public project. Requirements of producers and customers are considered. It is also an iterative development process like the basic V – model and it is usable for all kinds of applications because of the flexibility of the system. A further innovation is a company specific improvement process for their software development process.

The basic idea of this model is that all projects can be executed along the same schema. The projects can be classified into some project types. This types have adapted realization strategies with own procedure models. These models define activities, roles and outputs. With the help of these models the project can be brought forward from decision point to decision point. At these points the further strategy is chosen by the project manager after the evaluation of the project state. In other models a decision point is often called milestone.

The following figure shows an overview of this model including the communication between the customer and the supplier. As we can see the communication plays a major role in this model to ensure the effectiveness of the development process and the satisfaction of the customer.

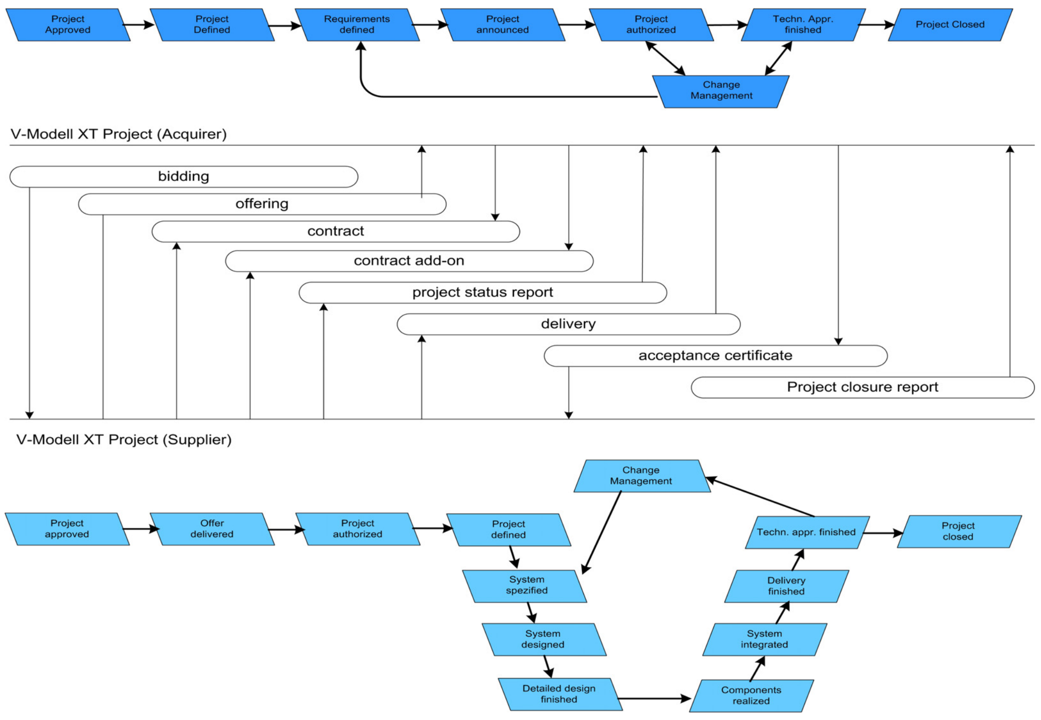


Fig. 5: Communication in the V – Model XT [55]

The tailoring approach of this model consists of four main steps. The first is the selection of a project type. This classification considers the purpose of the project and the different roles. The next step is the selection of adequate process modules. They are split into three types of modules. The core modules are mandatory for every project type, which are quality assurance, project management, configuration management and modification management. Required modules depend on the project type and the application domain. Examples are system construction or requirements specification. Optional modules are subjected to the application domain of the project, e.g. hardware development, software development or system security.

The next step is the selection of an execution strategy. This strategy consists of decision gates which are necessary to link the products of the process modules. The fourth step is the individual project planning which is made by the project manager after the determination of the first three steps.

The V – model XT has a tool support for decision making to find the appropriate process for an existing problem, is compatible with common software engineering assessment models and has a process measure framework to ensure continuous improvement of the development process which is also covered over tailoring methods.[49] Therefore it can be seen as a very flexible development framework which is adapted for the requirements of today's software development.

2.2.4 Rational Unified Process (RUP)

The Rational Unified Process is an object oriented software development model and was developed by Philippe Kruchten form Rational Software in 1996. [31] Rational Software was taken over by IBM in 2002. IBM improved the RUP and today companies working with the sixth version of this development model. The development of the RUP is closely linked with the development of the Unified Markup Language (UML).[67]

As we can see in the figure below the RUP – Meta model describes four main project stages and nine disciplines consisting of six engineering and three supporting workflows. During the project phases, several iterations are made to ensure a proper quality standard for the software product. In the inception stage the feasibility of the project is proven. If the project does not pass this stage, it can be revised or discarded.

In the elaboration stage the theoretical development of the software is made. The development team generates several documents, e.g. the architecture of the software, use case models, a risk analysis and some prototypes. The documents are validated with some quality assurance methods to ensure that the risk of conception failures during the construction phase is minimized. If some findings are made the design of the software will be revised at this project stage because fixing errors during the construction phase causes much higher costs.

After that the construction of the software starts. In this phase the main coding work will be done. The core components and the majority of the features are implemented. At the end of this phase the first release of the software take place. As depicted in the figure below implementation, test, configuration and change management are the main disciplines in this project stage. But all other workflows are also going on. Basically all workflows exist during the whole project. Only the focus of them changes.

In the last phase the software is deployed to the customer. It includes testing, training of end users, configuration and maintenance. The quality of the software and the satisfaction of the customer needs are validated. This phase is closed with the product release.

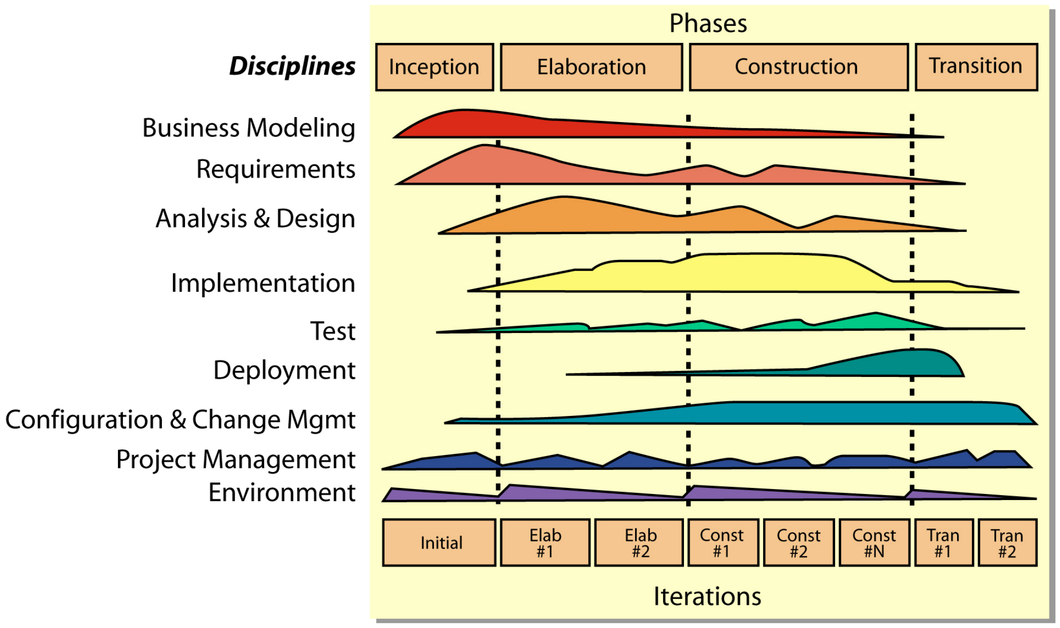


Fig. 6: The Rational Unified Process [29]

Each phase of the project is systematically planned and controlled. Any problems lead to the iteration of a document or artifact.

This software development process fits for large and complex projects which need good preparation and planning. Therefore it is very important that most of the failures are sorted out during the planning stages to avoid an explosion of the project costs.

Critics of this system suggest that the Rational Unified Process is too numb to react on requirement changes from the customer. But it is still a very popular software development process.

2.2.5 Extreme programming

Extreme programming is an agile software development method which was invented from Kent Beck in 1999 during his work in the Chrysler Company. It is less formalized than the traditional models but because of much iteration it has a good structure and a high level of effectiveness. The focus of this model is the implementation of the software and not the documents in the conception phase at the beginning of the project. But it has well defined procedures and is no guidance to chaos.[56]

Extreme programming (XP) considers the fact that in many cases requirements of software products changes during the project. Therefore Kent wanted to create a software development method which is able to react on theses changes without a massive loss of time and money.

XP consists of short development cycles and therefore the direction of the project can be changed very quickly. Only the features which are really necessary for the software will be implemented. If the customer recognizes that an additional feature is needful it will be assembled in the upcoming development cycle.

XP identifies five important values to develop software of good quality:

1. Communication: The developers and the customer have to communicate to coordinate their needs and their work. In traditional systems requirement documents have this function. This is one big argument for the flexibility of Extreme Programming.
2. Simplicity: "Keep it short and simple" – a motto for this development style. Every problem shall be solved in the simplest possible way. Additional functions are necessary if the customer request them. But there is another good reason for simplicity. If a developer has to modify software, he can understand and adapt the structure easier when the code is not complex and confusing.
3. Feedback: XP has three different views of feedback. Feedback form the customer helps to react to his needs and requirement changes. Feedback from the development team about project time and occurred problems is necessary to plan the whole project. Feedback from the software is given through the results of tests. Therefore the programmer gets direct feedback after changes of modules or the implementation of new functions.
4. Courage: This means that programmers should concentrate on their work and not on things which can happen in the future. They must have the courage to do everything that is necessary in the current situation. The code has to be easy to modify, old code should be thrown away and the focus is only on the existing problem.
5. Respect: In a loose working atmosphere respect is a very important thing to avoid anarchy. The team members have to support each other in every phase of the project.

Extreme Programming has some working practices. Examples are pair programming, Test driven development, collective code ownership and coding standards taken are some of them and they are worth to explain them in some words.[57]

Pair programming rests upon the four eye principle. Two programmers share one computer. One of them is the "driver" who codes the problem. The other one is the partner who tries to keep a view of the overall problem. The roles are changing regularly to boost the exchange of knowledge. The permanent review of the code guaranties less failures and a product of higher quality. This practice is also used to give a newcomer the chance to learn form an expert.

In test driven development the test routines are developed before the coding of the software take place. Because of that the developer gets a new sight of the problem and therefore often earns a better view. In collective code ownership the tasks are assigned to the whole programming team. No team member has any monopoly of knowledge and the success of the project depends on the coordination of the team. This practice benefits the communication of the team.

A development team works usually with coding standards, because the roles of the programmers are changing during every project. To ensure that everyone can understand the work of his partners they have to do the coding along predefined guidelines. This is also a very important point to avoid chaos.

Today extreme programming is a broad field with many options to implement software. Critics complain about the lack of structure in this method while proponents love this attribute. Never the less XP is established in the software development community.

2.2.6 Scrum

Scrum is located in the area of agile software development processes. Recently this software development process gained much popularity in the area of agile software methods.

The first ideas of this development method were mentioned in an article in 1986 by Hirotaka Takeuchi and Ikujiro Nonaka. In the 1990's Ken Schwaber and Jeff Sutherland implemented this technology in two companies at the same time. These actions can be seen as the origin of Scrum.[69]

As all other development methods Scrum has some predefined roles. The most important roles are the product owner who represents the customer, the Scrum Master who is the head of planning and development and the team which is responsible for the delivery of the software.

The basic idea of Scrum is that many software projects are too complex to plan their development in the long run. It is much better to implement a self organizing development team which is responsible for building and executing work packages. The sum of these packages is called “backlog”. It can be separated into a product and a sprint backlog.

The product backlog consists of all required features of the customer including technical dependencies. The elements of the product backlog are prioritized before the beginning of a new “sprint”. Elements with high priority are applied to the next sprint. A sprint is one working phase in the development cycle. The sprint backlog consists of all elements that have to be delivered during this particular phase. Normally one sprint lasts about 30 days and it has a special goal that has to be reached within this period. The working capacity of one package of a sprint should not exceed sixteen hours. Therefore bigger packages have to be split into smaller ones to gain more control over the time schedule. It is also important that the maximum working capacity of the Scrum team has to be adhered. During the sprint a daily meeting of fifteen minutes is held to ensure that all members of the team are working in time and on the same goals. The topics of this meeting are the accomplishments since the last meeting, the occurrence of any problems and the planned accomplishments until the next meeting. These topics ensure that every member of the development team has the same amount of information. Every development team has its own predefined rules for the daily meetings. These include the fixed schedule of the meeting, the way of preceding the meeting and possibly some special features which are useful for every team member. On a "burn down chart" all remaining tasks for the actual sprint are listed to make sure that the sprint is still on time.

During a sprint the working packages are well defined and a change of requirements is hardly possible. If a development team needs more flexibility, the duration of a sprint can be reduced. The operating principle is depicted in the following figure.

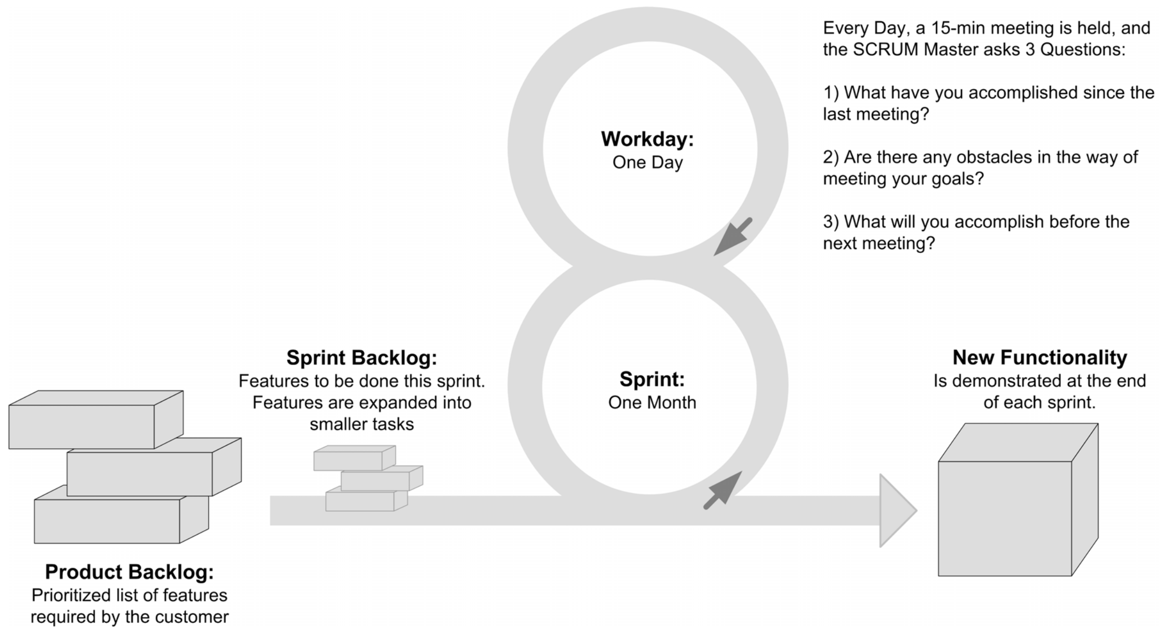

Every Day, a 15-min meeting is held, and the SCRUM Master asks 3 Questions:
1) What have you accomplished since the last meeting?
2) Are there any obstacles in the way of meeting your goals?
3) What will you accomplish before the next meeting?
Workday: One Day
Sprint: One Month
Sprint Backlog: Features to be done this sprint. Features are expanded into smaller tasks
New Functionality Is demonstrated at the end of each sprint.
Product Backlog: Prioritized list of features required by the customer

Fig. 7: The operating principle of Scrum [66]

Every sprint is set up in an iterative circle of four steps. During the development phase all required tasks to reach the goal of the sprint will be realized. In the wrapping phase the single elements are put together to prove the interaction between the elements and the functionality as a whole. These tests are made during the review phase of the sprint. The basis for the reviews is the requirements of the customer. If there appearing any problems, possible adaptations are developed during the adjustment phase.

Scrum was already used in several business sectors like finance, medics and the internet economy to produce software products of good quality. One big advantage is the every person that is involved in this process has to learn to see the projects form all points of view because each is responsible for the development of his own part. Therefore the decision of an appropriate quality assurance method has a special meaning in this framework. In traditional software development methods every team member has his specific role.[3]

With the help of this framework big and complex projects can be separated into functionalities and features to get a better overview of the required working steps for the development of the software. This can be seen as one big advantage of Scrum. It has the power to handle complex requirements without loosing the flexibility for adjustments.

It is also important for Scrum that the development team have a look at the past sprints to evaluate their outcome and to learn for upcoming development cycles. But Scrum is not a fixed development model in common sense; it is considerably more an attitude. The development team has to live the "Scrum way of thinking" to produce a product of good quality. Many approaches to implement Scrum have failed because of wrong expectations.

Altogether Scrum is a very effective development method which turns out in several successful projects but the method can hardly compared with the traditional software development style.

All methods presented in this chapter have several peculiarities because they use different approaches to the area of structured software development. Some of them are approved very long and others are quite new but the success of these methods show that different ways are possible to reach the designated goals in the area of software development. Therefore we use basic characteristics to describe the environment of a software project environment. The basis of these characteristics is the ISO 12207 model of life cycle processes because this model is a framework for all specific development processes and it can be seen as a structured approach to subsume them. They criteria for the comparison with quality assurance methods are general to provide a high level of flexibility. Special adjustments for each development method must be practically evaluated.

Finally the following table gives an overview about all methods described in this chapter:

Sub chapter	Method	Category	Invented
2.2.1	Waterfall model	traditional	1970
2.2.2	V - model	traditional	1986
2.2.3	V - model XT	traditional	2005
2.2.4	RUP	traditional	~1990
2.2.5	XP	agile	1999
2.2.6	Scrum	agile	1986

3 QUALITY ASSURANCE

This chapter shall give an overview of a selection of basic quality assurance methods which are typically used in different phases of the software development processes. It starts with the necessity and the stat of the art in quality assurance and then four methods of this area will be described to get an overview of the methods and their possible linkage to the software development processes.

3.1 STATE OF THE ART IN QUALITY ASSURANCE THEORY

Quality assurance is one major part of software development. With its help software development companies ensure the quality of their products in multiple ways. Quality can not be measured in an absolute way; it can be seen as a value of the stakeholders of a project. This value is given to the development team with the software requirements. The need of quality assurance methods arose in industrial development during the last century. Thru the growing complexity of industrial products the need for quality grew constantly. In the area of software development multiple types of requirements and constraints on the part of hardware, security, handling and many others demand the usage of quality assurance methods.

Today there are several quality assurance methods with different approaches and methods to cover all possible application fields. Quality terms are part of almost every contract. Therefore it is important to identify the most effective quality assurance method for a specific working process to meet the quality conditions required by the customer.[37]

There are some basic project management steps necessary to apply useful quality assurance methods. First it is important to identify the needs of the customer and to be informed about changes of requirements. During the development process these requirements has to be assembled in the software product even though the customer's requirements definition is not always clear and strait forward. In this case quality assessment methods can help to provide good requirements for the implementation phases. As soon as the requirements are clear they are the input for quality assurance methods during the implementation phase to ensure that the finished product meets the requirements of the customer.

Another argument for quality assurance is the necessity of an almost error free code. Major software systems have a very big amount of source code. Some of the modules are standardized items which are often used in many products like database connection objects or modules for user administration. These pieces are most often adapted to the current problem. But some pieces have to be developed completely new. Everyone who has coded anytime in his life will agree that it is very easy to make mistakes during both processes. Today software development environments help to avoid typing errors by highlighting special phrases. Never the less checking the functionality of a finished piece of code is a very important thing. Testing methods which are also a part of quality assurance are developed to check the code for errors.

If we look at the quality assurance practices in companies we can often see that the quality levels of their products play a very important role. A single product is checked with the help of several stages of quality methods. But the findings in these quality checks are only used to solve the problems that occurred during this check. The findings are not stored and reused for upcoming quality checks in later project stages. Even though the benefit of the reuse is at hand it is rarely accomplished in practice. So we can mark this point as a possible research topic for this thesis.[45]

It is obvious that many companies spend a lot of money for the quality assurance of their products. Another fact is that the relation between costs and effectiveness is indirect proportional by an increasing number of quality assurance activities. The following figure depicts the relationship between these two values:

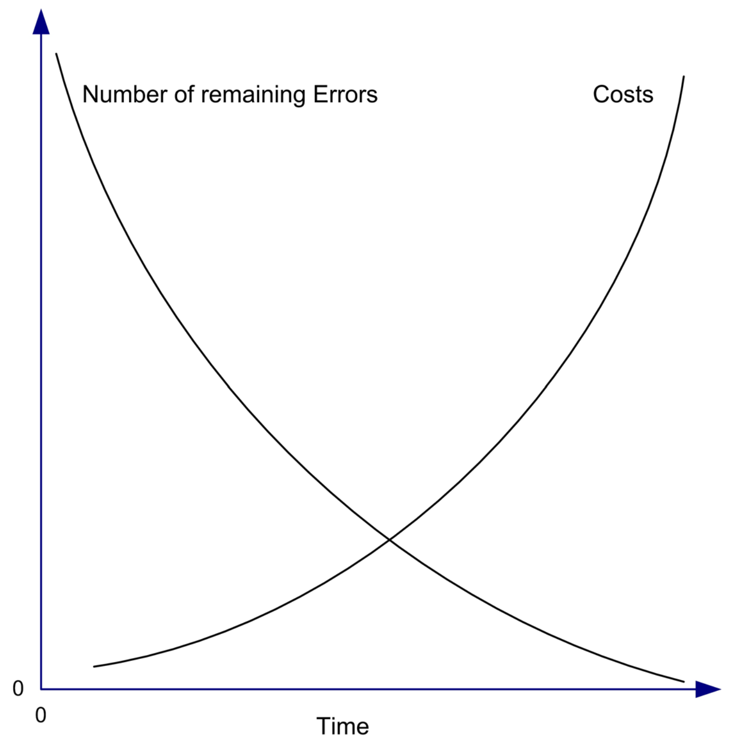


Fig. 8: Relation between the number of errors and costs

As we can see the effectiveness of the first quality assurance methods is very high. With less amount of money we are able to find some errors. But as soon as the majority of errors are fixed the costs to find additional errors increases very fast. Therefore it is really important to get as much information as possible out of every single quality test. Because of that the reuse of the gathered information seems to be quite useful. The goal is to keep the costs low to move the crossing point of the two curves to the right side in order to discover more errors with the same amount of money.

There is a similar relationship between early and late discovered errors. If an error is found in the early planning stages of a project the correction activities are much cheaper than the work on a late discovered error. This is also a reason why well executed quality assurance methods are highly valuable for a company. They help to save money during the whole project if they are conducted in the right amount.

Today, there are many different quality assurance methods are in use as mentioned before. Three of the more popular methods are described in the following sections to get a better view of this topic.

3.2 INSPECTION

Software inspection is one of the most popular quality assurance methods involved in the various software engineering processes. The goal of inspection is the approval of a document created during the engineering process to use it for further work. This QA technique is normally used for requirements documents and every kind of specification and test plans.

Michael Fagan developed this method at IBM Kingston NY Laboratories in the 1970s. In 1976 he reported his findings in his famous paper (Fagan, 1976). [21] Fagan noticed that it is cheaper to discover errors as early as possible in order to save money because later in the project corrective actions are normally very expensive. Therefore it is at hand that quality assurance methods also have to be conducted during the very first project stages to avoid the drawing out of failures. The figure below shows the relative costs of fixing errors during the different project stages:

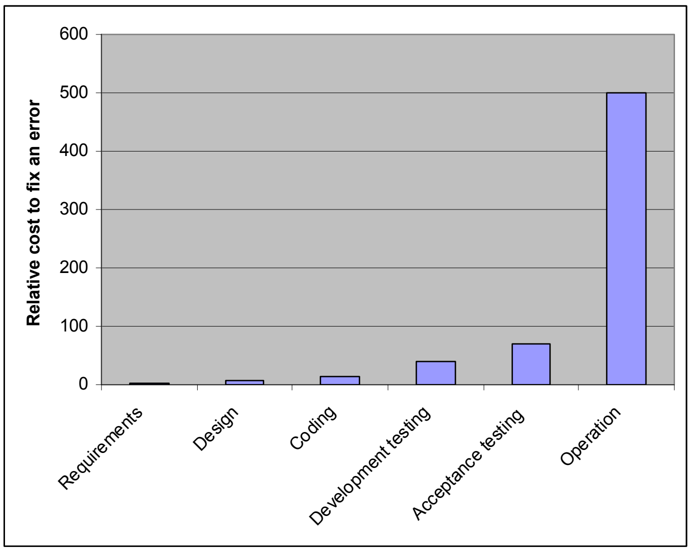


Fig. 9: Relative costs of fixing errors (Boehm, 1981) [21]

As we can see fixing an error in the early project stages is much cheaper than in the late project stages. Today software projects have very strait calculations according to their budget which means that the project manager can not afford many heavy errors in late project stages to ensure that he is able to keep the monetary forecast.

But inspections also cost money. Therefore it is imperative to have well prepared documents before conducting an inspection. Fagan increased the productivity rate of the IBM project of about 300 percent with the help of his inspections. By using walkthroughs the productivity rate increased only by 144 percent. Walkthroughs are peer discussions used for training of knowledge about a document. It is also very helpful for the whole development process to bring the development team on an equal level of knowledge. Furthermore walkthroughs advance the collaboration in the development team because problems are discussed and solved as team and not as a single person. But walkthroughs are not that systematic as inspections and have the focus on problem solving and learning.

Now what are the goals of the inspection? The major goal of inspections is the reduction of errors in any inspected document as we had above. But this brings us to some other advantages which co-occur with this benefit. As the figure shown above, inspections help to reduce the overall project time and costs because of the early detection of errors. But inspections also help to reduce problems in the operation phase and therefore lead to a higher level of customer satisfaction. The productivity of the company also increases by the continuous improvement of the working process on the basis of the findings in the several inspections.[20]

Another big advantage is the so called “four eye principle”. At least two persons have to check one document against completeness and correctness. It is easy making slips of the pen under pressure and to overlook some errors by checking the self written document. Inspections accomplish this principle. Normally three or more persons are checking a document independently before the inspection starts. The chance to discover nearly all errors in the document is therefore extremely high and increases with the number of people who check the document.

Now let us have a look at the inspection process itself. The author of a document gives his quality representative a request for inspection after the finishing of the first release of the paper. The inspection leader checks the first release against pre defined entry criteria to ensure that the document has a certain minimum level of quality. This procedure helps to save resources because inspecting bad documents only wastes time and costs money.

If the document is approved for the inspection process, the inspection leader plans the inspection schedule and the inspection cycles. The kickoff meeting is the start of the inspection. During this meeting, the necessary documents are distributed and roles are assigned to the inspectors. This meeting should be also a motivation for the inspection team and shall help to clarify all upcoming questions. The individual checking phase starts after the kickoff meeting. The inspectors working alone with the help of a checklist and checking procedures to discover some potential defects in the documents which have to be checked. Every finding is then recorded in a list. The lists of the inspectors are the basis for the logging meeting afterwards.

The logging meeting is conducted with the development team, inspectors and sometimes a moderator. During the meeting the inspectors reveal the possible failures which they have discovered before and discuss them with the development team in order to qualify them as real failures, or possible risks for further project stages or as approved. At this meeting further potential errors are discovered and discussed with the development team. The third purpose of this meeting is the preparation of possible ways to improve the documents and the agreement on the further time schedule.

Afterwards all discovered errors are reported to the development team with a log file to resolve the problems. The development team corrects the errors and issues a new revision of the documents which will be the basis for approval in the follow up inspection meeting.

The follow up meeting is not mandatory. It depends on the number of the discovered errors and the complexity of the documents. If there is no follow up meeting the inspection leader checks the new revisions of the documents to ensure that all defects, which are recorded in the log file, are adequate resolved. If a follow up meeting is conducted the participants are the same like in the first meeting. The work is distributed equally to them from the inspection leader described at the beginning of the paragraph.

After the successful check of the revised document the inspection leader checks it against some predefined exit criteria. These shall ensure that some fixed parameters are applicable for this document, e.g. all meetings are conducted, the failures in the log file are resolved properly and so on. If the document passed the exit check, it can be released for further use.

The following figure shows the inspection steps graphically:

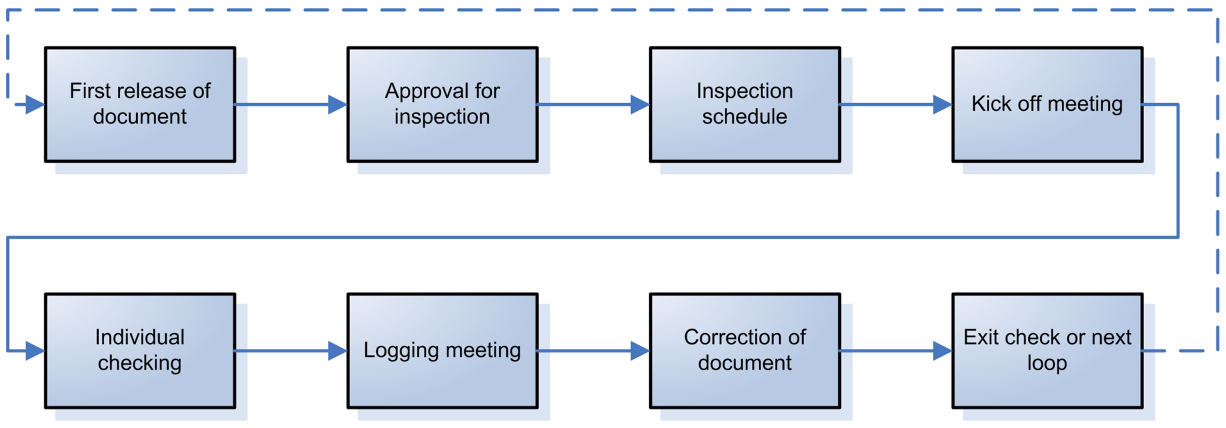


Fig. 10: Workflow of inspections

A big advantage of this process is the possibility to discover problems of the working process. If some failures appear periodically the development process has to be adapted with the help of preventive actions to avoid this failure in the future. It is easy to identify such problems with the help of the log files which are developed during the inspection process. The problems can be classified and collected to create a database for potential process improvements.

Other big advantages are the productivity improvements and the cost savings as mentioned above. Inspections are a structured way to validate documents before the final release is done. But on the other hand the inspection leader has to ensure that the documents meet the minimum criteria in order to guarantee that the inspection will be effective. This is mandatory because the process involves several persons and takes some time. Both are cost factors for the project and therefore only effective inspections make economic sense.

Fagan reported the increases of productivity and the savings of time and costs in his paper in 1976. He put the quality assurance methods of IBM on the next level with the help of his inspection technique.

But advocates of agile software development processes take the view that the inspection process as it described here is too inflexible. They prefer more unbureaucratic ways like it’s mentioned in the chapter of agile software development processes.

3.2.1 Reading techniques

Different reading techniques can be applied during the inspection process. Often used are checklist based reading and usage based reading.

During checklist based reading a list of issues is used to depict all possible kinds of errors. The document is checked systematically against this list. The list contains of several problem possibilities which are grouped according to their topic. Checklists are created and updated with the help of defect analyses and they should be easy understandable to provide a good basis for the inspection team. Another important aspect is that the topics on the list must be consistent and not too general to ensure a proper outcome.[13] Normally each company uses predefined conventions for the content of these checklists to support the inspectors. Today checklist based reading is a standard inspection technique in many software development companies.[47]

Let us assume the inspection team has to check a risk analysis document with the help of a checklist for example. One item of the list could be the question about risks regarding new technologies which have to be applied in the project. Another example for an item could be a question regarding the qualification of interfaces to other systems.

Checklist based reading is a very popular reading technique and also topic of various scientific reports and comparisons to other reading techniques.

Usage based reading tries to evaluate the quality of a document form the view of the user of the software. Therefore the basis of this technique is use cases which describe the requirements of users. The advantage of this approach is that these requirements are necessary for daily use of every piece of software. So these errors have a higher priority than other errors because they are recognized by the user. The purpose of other reading techniques is to find as many faults as possible regardless how important they are. Because of the fact that it is nearly impossible to produce error free software, it is more effective to concentrate on the important faults which could be visible for the user.[47]

Usage based reading has the following workflow: Before the inspection starts the use cases are prioritized according to their importance from the user's point of view. In this step an important input is the requirements document from the customer and the end users. Then the inspection process starts with the selection of the most important use cases and the step by step tracing of the document on the basis of the use case. Identified problems are recorded and reported to the development team. This process will be repeated with all selected use cases.

According to scientific studies usage based reading is more efficient and effective than checklist based reading because it provides a greater possibility to find critical errors.[48] But this fact is also true for every kind of faults. Furthermore inspections with usage based reading detect more unique faults and the duration of the inspection process is even shorter. So we can conclude that there are various differences regarding efficiency even in the same quality assurance method. This fact shows the importance of practical experience to qualify them.

3.3 ARCHITECTURE REVIEW

The goal of this group of methods is an error less or almost error free architecture of a piece of software as the name "Architecture Review" implies. This chapter gives a short overview of two techniques within this domain. These methods shall help to clarify all possible problems at the architectural stage of any software project. In this chapter, the focus is laid on ATAM because of its propagation and awareness in the area of software development. PASA is a lesser known technique and shall provide an alternative to ATAM – which is also the reason why it is mentioned here.

3.3.1 ATAM

ATAM is an acronym for "Architecture Tradeoff Analysis Method". ATAM was developed by Software Engineering Institute at Carnegie Mellon University in Pittsburgh, Pennsylvania and shall help to evolve software architecture.[26] The idea behind it is a well structured approach at the beginning of any software project to define and clarify the most significant points. The following points are the outcome of the ATAM process [6]:

- Detailed definition of quality requirements

- Better architectural documentation
- Structured identification of risks during the first stage of the project
- Better communication between all involved parties
- Improved decision making process at the beginning of the project

The figure below shows an overview of the basic concept of ATAM. We can see the parallelism between business drivers and software architecture at the beginning. Out of the business drivers quality attributes can be elaborated which lead to specific scenarios that can be used for further analysis of the project and help to provide a better overall view for all involved parties. The software architecture is inspected to find different architectural approaches which lead to the intentional outcome. These approaches are evaluated to find the most adequate way of development.

These two items are considered in the following analysis to identify tradeoffs, sensitivity points, risks and non-risks. Afterwards all points that need special attention are recorded and the possible impacts on the predefined project goals are checked. If there are some severe problems, the software architecture and the business drivers have to be reassessed. At this point the ATAM circle starts new.

This procedure will be done until all major problems can be solved or the level of the remaining risks is reduced to a minimum.

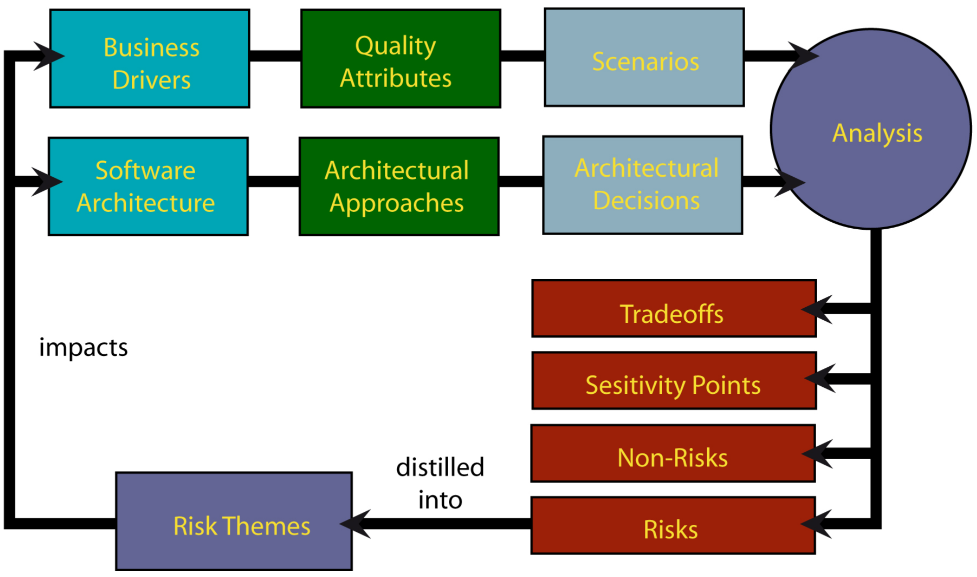


Fig. 11: The concept of ATAM [65]

One big benefit of this system is that the concept of the project is carefully reviewed which gains some advantages. First the stakeholders have the ability to think about their real needs and requirements on the software. Therefore no unexpected surprises can happen at the end of the project. Secondly all involved persons can be sure that the other side has the same information and association of the project – in other words there are no misunderstandings anymore. Thirdly all problems are identified at the beginning of the project which helps to save time and money. This also guarantees that the project team lays special interest on these problems to ensure an acceptable solution to it.

ATAM tries to visualize the consequences of architectural decisions regarding to quality attributes of the system. The architecture of any system determines the achievement of functional and quality goals. Therefore it is important to clarify all possible problems at the beginning of the project. ATAM provides a repeatable analysis method of structural integrity of any systems architecture.[24]

As we can see the focus of ATAM lies on the analysis of all important points for a project to distil risks out of this analysis phase. This is a very structured approach to elaborate information out of the sensitive project points. A similar way is conceivable for the qualification of quality assurance methods relating to predefined project attributes. Once the characteristics of a project are defining the selection of QA methods that can be done with the help of these characteristics if they are also suitable for the characterization of quality assurance methods.

Barbacci, Longstaff, Klein and Weinstock described in their report on quality attributes that there is always a trade-off between different quality goals. They pointed out several possible explanations. One of them is depicted in the following figure:

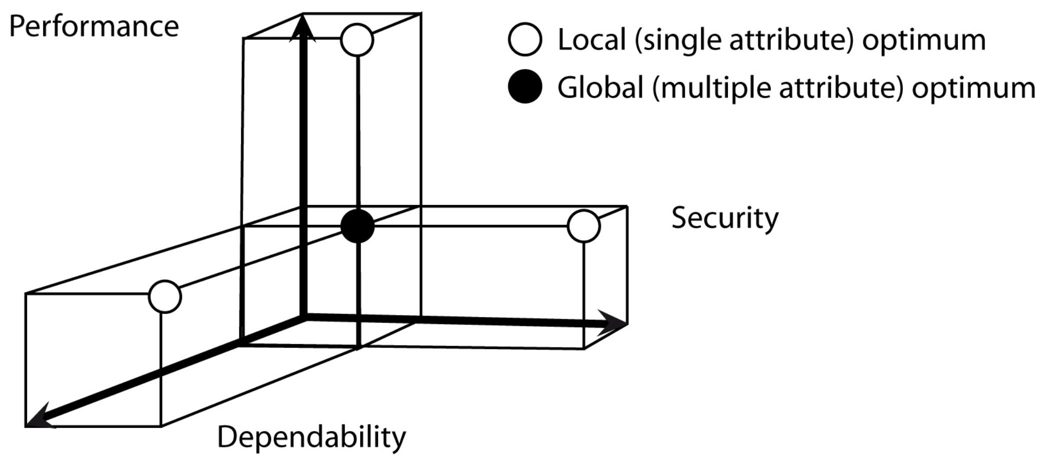

Performance
○ Local (single attribute) optimum
● Global (multiple attribute) optimum
Security
Dependability

Fig. 12: Trade-off between quality attributes [58]

As we can see the local optima are different to the global optimum which considers all different points of sight. Therefore it is very important to qualify a project with the aid of all different aspects to ensure that the overall outcome meets the previous defined requirements. ATAM is based on this concept and shall be an example for every qualification approach which is dealing with multiple aspects.

3.3.2 PASA

PASA is an acronym for “Performance Assessment of Software Architectures. It is a systematic approach to assess the performance of any software architecture. It was developed by Dr. Connie U. Smith and Dr. Lloyd G. Williams.

The following questions regarding software architecture represent the motivation for the development of PASA [40]:

- Will it support the performance you need?
- Will your users be able to complete tasks in the allotted time?
- Will the system scale up to meet your future needs?
- Are your hardware and network capable of supporting the load?

The usual way of conducting this method is the following. At the beginning of the process the benefits and the workflow of PASA are presented to managers and developers to ensure that all involved people have the same information status. This phase is called *process overview*. During *architecture overview* the development team introduces the prospective or current architecture of the software. This step is useful to give the assessment team an overview of the current state.

The next phase deals with the *identification of critical use cases*. This is necessary to identify problems regarding performance and scalability issues. For each of this cases *key performance scenarios* are created to identify the requirements constraints of the software. The next step is the *identification of performance objectives* for each of the key scenarios. These objectives have to be quantitative and measurable to ensure an evaluation at the end of the project. Afterwards an *architecture clarification and discussion* is performed to specify all important scenarios and delicate areas in detail.

The advanced architecture is analyzed in order to check it against the performance objectives which were developed before. If any discrepancies occur architectural alternatives have to be identified in the next step of the PASA process. The outcome of the analysis and the possible alternatives of the problematic areas are presented to developers and managers afterwards. At the end of the PASA workflow the economic relevance is analyzed regarding to the benefits and improvements during the application of PASA.

PASA uses the practical approach to improve software development processes by analyzing the critical areas of any project at the application level. Therefore we can say that PASA is a practice oriented method out of the family of architecture review processes. This conveys if we look at the main tasks of the application process of PASA. The architecture of software is planned on the basis of the stakeholders use cases and performance criteria.

This method ensures a good communication between developers and customers and helps to clarify the points of view of every involved party. PASA is certainly more sophisticated than other quality assurance methods because of the intensive analysis phase and the number of people involved.

3.4 TESTING

Software Testing is also a widely spread quality assurance method in the area of software development. It is an analytical method to find bugs in software and is a tool to prove their functionality. In the beginning testing and debugging were not separated. In 1979 Glenford Myers he intended to split debugging methods from verification approaches to achieve a certain level of quality. The mail goal at this time was also finding errors.[68] In the 1980s these tests evolved to measuring instrument for the quality level of software. At the beginning of the 1990s the next step in test evolution was the crosscheck with the requirements at the beginning of the development. The developers realized that tests are more effective when the requirements are the basis for the test selection. This led to a higher level of customer satisfaction.

There are several different dimensions of quality purposes, like performance, usability, scalability, compatibility or reliability. The focus of these attributes depends on the operational environment of the software. With the help of testing software can be validated and verified. The validation of software proves their functionality according to the requirements of the costumer and the verification ensures that the software meets the specifications. The following paragraphs discuss some testing methods and the testing process itself.

3.4.1 Black box testing

During black box testing the tested piece of software is treated like a “black box”. This means that the internal structure of the software is not observed. The test plan consists of several different input values and the expected output values for the software. For example if we are dealing with software for mathematical multiplications the input values can be 5 and 7 and the expected output value is thirty five. The test engineer executes the test by feeding the program with the input parameters and records the output of the software. If the output matches with the estimated outcome the software passes the test. If the real outcome is different to the estimated one the test fails. After the end of the test series the complete checklist is going to the software engineer to adapt the software according to the outcome of the tests.

Such tests are applicable for the verification of the software. It is useful to generate test cases which consist of inputs that are nearby or over the boundaries of the software to prove their right behavior in these areas.

3.4.2 White box testing

In case of white box testing the test engineer has access to the internal code of the software. The tester has the ability to check all possible states of a loop or an “if” – statement for instance. Let us assume we have an “if” – statement with zero and one as entry criteria. In this case the tester checks if both values are able to appear and the outcome of the statement in both cases.

White box testing is also used to check parts of software which are rarely used and therefore not often checked during black box testing. In this case white box testing helps to ensure the completeness of a testing series.

Furthermore, it is important to test software with the knowledge of code to avoid unexpected program states during the operation phase of the software.

3.4.3 Grey box testing

Recently, grey box testing has become more and more popular. The test engineer who creates the test cases has access to the internal code of the tested software but the conductor of the test series operates at the so called black box level. This means that he only gets the test cases to perform a simple black box test. The advantage of this testing procedure is that the conductor operates like a normal software user and therefore simulates a standard environment for the tests. The idea behind is that it is not necessary to check software states which do not appear during the standard usage procedures.

This type of testing is very useful for integration tests where some modules of different development teams are assembled. It is also helpful to check or determine system boundaries at critical software areas.

Other test procedures aim for special attributes of software. These are for instance performance testing, usability testing and security testing. A performance test checks if the software is working correctly under a high amount of input data. Usability tests shall ensure that the users are able to handle the software easily and security tests aim for the resistance of the software against intruders. Another topic of system security is the correct distribution of user rights.

3.4.4 Testing process

There are some different testing levels for any type of software. Unit testing deals with the correctness of single units of software, e.g. modules or components. Integration testing aims for the interfaces between the modules to ensure their correct interaction. System testing deals with the complete software and shall prove the accordance with the customer requirements.[4] Last but not least acceptance testing is a tool to ensure the satisfaction of the customer and the end users and is often a mandatory part for the final release of the software product.

The procedure of testing shows many different variations which often depends on company standards or different development processes. Therefore the following description of the test process is an example how a test process can look like. Before the start of the test phase a test plan is developed with all scheduled tests in it. This plan contains type, timeframe and responsibilities for each test. According to this plan several test cases are created. These cases contain the necessary system environment, input parameters and the expected outcome of the test. If the tests are done manually, the list of the test cases can be used as a checklist to ensure the completeness of the test series. If there is test automation the cases are programmed according to this list.

The tests can be conducted during the development of the code or in a separate test phase after the coding. The results of the test are recorded and compared with the estimated outcomes. If there are deviations the problems will be reported to the development team to conduct corrective actions on the code in order to repair this error. Testing has normally some loops – also with different test cases – to guarantee that the software is working correctly and to prove the revision of further detected errors.

3.4.5 Test driven development

Test driven development is a hybrid of a software development method and a quality assurance method. It was developed at the end of the 20th century and is one part of the Extreme Programming initiative which is described in chapter 2.2.5. Test driven development normally consists of short iterations with the following workflow. First the test cases for the functionality of the software are written. Afterwards the code to satisfy the functionality is implemented and tested with the help of the test cases developed before.

This means that the requirements of software are defined with the help of test cases and not with requirements documents. The following steps show the design process of test driven development in detail [70]:

1. Add test
   The first step of this software development method is the creation of a new test case which has to fail because the new component is not written yet. It is necessary to understand the requirements of the new module in order to develop a test case. Therefore the knowledge of the software and its requirements has to be very good.

2. Run tests
   This phase deals with the running of all tests including the new ones to ensure that all previous tests work correctly and the new tests fail. If the new test would pass, it would be useless to take it as base for the new implementation.

3. Add code

Now the necessary code to pass the new test will be written. The main goal of this step is that the test can be passed under some circumstances. It is possible that some wrong conditions occur because there are improvement steps for the code afterwards.

4. Run tests
   Now all automated tests are run again to ensure that no other problems have been occurred during the code implementation process. It is always possible that new pieces of code have an effect on the existing code. Because of that this step is really important.
5. Redesign code
   Now the code can be redesigned to remove last problems and to do a cosmetically clean up. The test cases help the programmer to ensure that the code meets all necessary requirements.
6. Iterate
   The whole loop starts again to implement a new requirement into the software. The size of the cycles depends on the complexity of the software and the size of the development team. They can also vary in each iteration.

The advantage of test driven development is that the complete development cycle is based on a large number of tests which ensure a high level of quality. Furthermore the development process is split into small cycles which help to concentrate on the current problem. The conduction of all tests in each cycle guarantees that also the finished code remains error free. Altogether test driven development is a very effective alternative of agile software development processes in combination with a high level of quality assurance.

3.5 AUDITS

With the help of audit processes, products and organizational structures can be evaluated. In many quality management frameworks audits are mandatory to prove the effectiveness of the system and to guarantee a continual improvement. Audits are conducted by a trained auditor. The word “audit” is derived from the Latin word “audire” which means hearing. Therefore the auditor interviews his audit partners about the work they have done and their used methods.[23]

The audit procedure is normally the following. The auditor works out a questionnaire for the audit which is specially made for the project that will be audited. The questionnaire consists of general questions about the working process and the reporting and communication of the project team to ensure that the working environment runs correctly. The second type of questions is about the project. Their content is a general overview of the work on the project and problems and deviations which occurred during the last project phase. Finally the questionnaire shall implicate all relevant answers for the auditor so that he is able to get an idea what is going on and that he can see which tasks need more external control.

It is important that the questions do not have a yes or no answer to ensure a normal conversation between the auditor and the audit partners. This is important because the auditor normally does not know the project in detail. So the audit partners have to tell about the project to explain the risks and problems in detail. Therefore the conversation technique is very important during an audit. The auditor has to provide the feeling to the audit partners that the audit is no test situation. The audit shall help the audit partners to see their project from another point of view to detect deviations.

An overview of the tasks at the audit is sent to the audit partners to guarantee that they have the possibility to prepare themselves for the audit. This helps to get a better result at the audit and to create suggestions for improvement of higher quality.

During an audit the audit partners are asked for their procedures in use, their work on the actual project and general methods of the project team and the company. These areas shall provide a good overview for the auditor. With this information an auditor can normally qualify the status of the project and detect some potential risks.

The gathered information is written down in an audit report. This report consists of detected deviations, necessary improvements, a note about the status of the project, an estimation of the motivation of the audit partners and a rating of the project (including project team). It can be used to improve the quality of the working process and the project itself.

On the basis of the audit report the improvements are scheduled and conduced. After the finalization the improvements have to be controlled for completeness and effect. This is also a task at the following audit to ensure that the improvements are sustainable.

The most critical point at audits is the belief of many audit partners that they are in a personal test situation. The rating of the audit is not dedicated to any member of the project team; therefore it should be a key process indicator for the level of the team and can be used to show the improvement over time. Therefore no one gets a bad mark for disclosing any problems that occurred during the project.

Audits have proved themselves as a quality assurance method for working processes, planning documents and text documents relating to any project. They are not so good for proving code documents or the like. Audits are also mandatory in quality management systems like the ISO standards. They provide constant improvements on the quality management system of a company in the standards. Therefore audits are mostly used for covering an overall view of projects and products. They can be seen as control mechanism for structural views of a company and its work.

All quality assurance methods described in this chapter have different approaches and targets. They were all developed to improve the quality of software products and improved during the years of usage. As data form companies show they are applied in the most suitable parts of software projects which is basically the right approach. But the outcome of these methods is rarely reused as input for the further development stages. This could also lead to an improvement in productivity of the development process. In the right circumstances a combination of various quality assurance methods can lead to a complete quality assurance strategy which can be customized for all kinds of companies and development processes over predefined criteria. This idea will be discussed in the practical sections of this thesis.

4 MEASUREMENT METHODS AND DECISION THEORY

Everyone makes decisions in daily life. Some of them are important, others are not. They are often made subconsciously. The decision theory deals with every kind of decision making in our life because they affect our future. No clear decision can be made without a special objective. A decision problem is present, if there are two or more alternatives that lead to different future states. Even the execution of an action implicates a decision, because the outcome is different whether the action is performed or not. If there are two alternatives leading to the same state in the future, we have no decision problem at all because both alternatives have the same outcome. Therefore, a decision problem is the result of possible different future outcomes.

Bigger decisions are often composed of some partial actions which are combined by the decision maker. One good example for this is the purchasing of a car. This action is normally the sum of different smaller decisions. First of all the necessity of a new car is balanced by the decision maker. If the necessity is given he must decide which type of a car it should be and if his budget is big enough to buy or if he has to lease it. Furthermore he has the choice between a new and a used one.

The next step is the comparison between different make of cars. After all these decisions are done he has to choose personalization attributes like color, features and extras. So we can see that this procedure is quite difficult if we look at it in detail. The same problems can be observed by decisions from stakeholders of a project. They must also recognize various different influences regarding the project to ensure that it is conducted successful.

Decisions are made on the basis of information which is carried together before. So one goal of this thesis is to help the responsible persons gathering information in order to have a good preparation for their decisions.

But it is also important to mention that the decision maker only can perform feasible actions. Dreaming about a world trip is quite nice but if the budget of the traveler is too small, the world trip will not be more than a nice idea. Therefore the quality of the information which is the foundation of a “good” decision is very important. This chapter deals with the background of decisions – the so called decision theory which provides different methods for handling with decision problems.

4.1 BASICS OF DECISION THEORY

Decision theory is a scientific area in mathematics that tries to bring human decision finding into a mathematical model. The basic goal is to find a way of getting an optimal decision for a given problem with the help of a logical solution approach. [34]

In normative decision theory the concern is to find the best solution under the assumption that the decision maker has full information and acts completely rational. The scientific approach is to create rational methods that help to people in their decision finding.

The field of normative decision theory can be split into decision models, general structures for the design of models and rules to describe targets. The target of our use case is the finding of the best quality assurance strategy to given environmental conditions. Therefore our need is the development of an appropriate decision model to support the project manager in his choice of the right quality assurance strategy.

The descriptive decision theory has a different point of sight. In real life people do not act fully rational and the advantage of full information is also not given to them. So this scientific field tries to describe how people will act under "normal" circumstances. An empiric access is used to describe the actions of human beings. Normative and descriptive decision theory is closely related because the aim is getting the perfect decision with partial information and forms of irrationality.

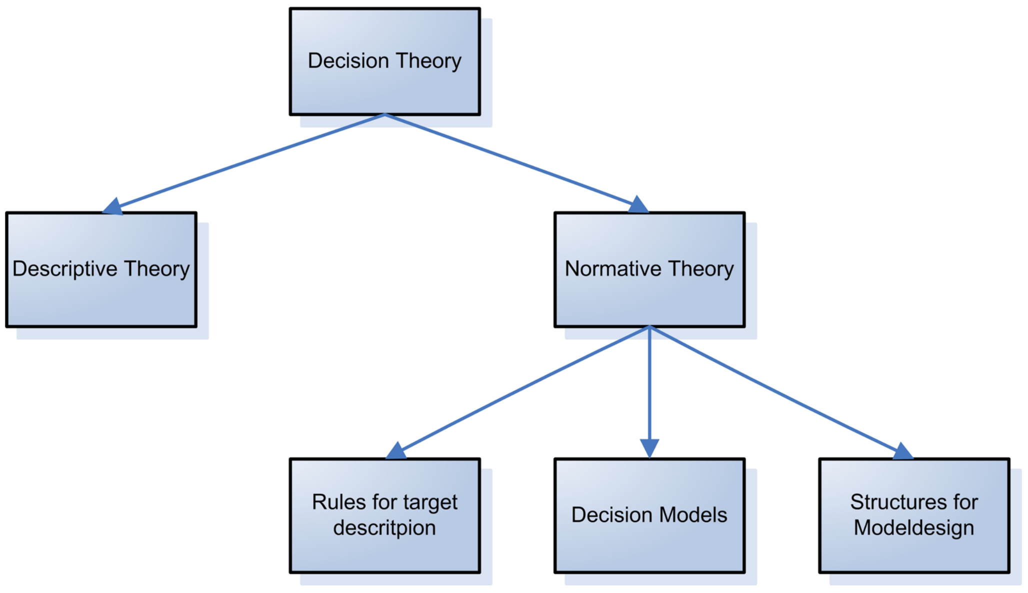


Fig. 13: Overview of decision theory [34]

People are making decisions with the aid of a general decision process that is split into five phases:

1. Formulation of the problem
2. Specification of the target system

3. Defining possible alternatives
4. Selection of an alternative
5. Decisions in the realization phase

4.1.1 Formulation of the problem

The start of the decision process is the occurrence of any nuisance that points out a need for action. Depending on the type of the problem the formulation can be easy or a complex process. If an employee quits a manager can take on a new employee for compensation or he allocates the work to the colleagues. In this case the alternatives are at hand and the formulation is quite simple. If the problem is the invention of a new product line, the alternatives are not as clear as in the example given before. The manager has to collect information about the market, the technological feasibility and other things that are important for the success of the project. In this case the problem will be split into several pieces to ensure that the solution is possible.

In general the formulation of a problem is dependent on the type and the size of the problem and should be convenient.

4.1.2 Specification of the target system

Choosing the right approach to solving a problem depends on the expected outcome. If the manager has no vision about the outcome, he is not able to find the right alternative to solve the problem. Therefore the definition of the targets is very important. Only with a predefined aim the alternatives to reach it can be evaluated properly. But if there are no feasible solutions, the target system has to be adapted to get a solution next to the optimal one. This can be a very tricky task because the right choice of the target system is also a decision problem. Therefore we are dealing with encapsulated decision problems. But in the case of software projects the customer defines the right way with the declaration of his requirements. If there is any problem with the solution he will be the appropriate contact person.

4.1.3 Defining possible alternatives

First of all the conditions and constraints of the solution process have to be evaluated. Alternatives which can not be realized have to be discarded because they are no proper alternatives. The task of the decision maker is to find proper solutions which needs experience creativity and subject specific knowledge. If these skills are more advanced the search for a solution is much easier. But there are no real guidelines for the search after alternatives. In general it is always helpful to have more alternatives because some of them could be disqualified during the selection process. But it is not advisable to create unnecessary alternatives just to have more options to discuss.

In the next step we have to estimate the consequences of the developed alternatives. Since there is no complete information about the things that will happen during the solution process, the forecast is always uncertain. So probabilities over future happenings are joining the decision process. To avoid probability as much as possible we have to gain more information about the environmental conditions around the solution process. It is also advisable to qualify all possible risks which are threatening a successful outcome. Only the risk definition and the thoughts about them reduce the probability of occurrence.

4.1.4 Selection of an alternative

This is the most important phase in the decision process. The methods in it depend on the knowledge of future events or the probability of occurrence of them. The strategy differs if the decision maker has to deal with uncertainties about future occasions or not. If we have the case of perfect foresight about the future we only have to choose the most valuable outcome of the problem and conduct all necessary steps to reach it. In case of uncertainties it is a bit more difficult. In the context of this thesis we have to consider uncertainties because in every software project the manager has to deal with risk. These methods will be discussed in a paragraph below.

4.1.5 Decisions in the Solution phase

If the decision maker starts solving the problem the main decision was made before. But there are several minor decisions which occur during the solution phase. That is because of the uncertainties mentioned in the paragraphs before. So the solution phase itself consists of several smaller decision problems. The smaller these problems are the easier are they to handle. The decision maker has to adapt his chosen alternative to the actual happening conditions. As we can see it in daily business life a project manager is always a problem solver who has to deal with every kind of unexpected things no matter on which type of project he is working.

4.2 THE BASIC MODEL OF THE DECISION THEORY

From outside decisions look partly very different but the basic structure is always the same. Therefore we have a basic model to solve decision problems. The decision maker needs an objective function and a decision field which consists of at least two alternatives, expected results and external environmental conditions. Alternatives are assembled of action variables. Results are defined over goal variables and environmental conditions can be displayed by data which have influence on the model.

The objective function appears in four different ways. First of all the final state can be a fixed value. Another possibility is that the goal variables achieve or beat this value in the final state. The last to ways are to maximize or minimize the goal variables which is in fact the same operational process with only a different algebraic sign.

In case of certainty the objective function is derived directly from the utility function. The maximization of it shows the preferred alternative because in that case the decider is able to gain the highest possible utility for himself. So this is a classical optimization problem in the area of mathematics. Under uncertainty the utility function has to be extended by the intensity of the alternatives. The intensity represents a value which can be compared with the risk factors. For example if there are two alternatives with nearly the same goal values and the higher one is far more risky a rational thinking person would choose the safer option although the utility is not completely maximized. But the second best alternative is better than the absence of any solution.

To bring these considerations into a mathematical model we need the target function and the goal matrix. The target function can be created in different ways and depends on the environmental circumstances of the problem. Some functions can be very small and easy, others are far more sophisticated. But the quality of a target function does not depend on their complexity but on the ideas and concepts which are handled with the function. Because of these considerations it is quite hard to give a good example for a target function which is representative.

The goal matrix consists of all possible alternatives of outcomes, environmental conditions which influence the problem solving process and the probability of their occurrence.

	$w(S_1)$	$w(S_2)$	$w(S_3)$
	S_1	S_2	S_3
A_1	e_{11}	e_{12}	e_{13}
A_2	e_{21}	e_{22}	e_{23}
A_3	e_{31}	e_{32}	e_{33}

Fig. 14: Example of a goal matrix with risk [34]

The figure shows a goal matrix. A_n are the possible alternatives, S_n denote the different environmental conditions, $w(S_n)$ are the related possibilities of occurrence and e_{nm} are the outcomes. The number of possible outcomes can be very high if there are many different alternatives and environmental conditions. Therefore it is often helpful to make a pre selection to reduce the effort of building such a matrix. For instance the environmental states can be reduced by concentrating on the critical outcome data only. It is also possible to classify environmental conditions with identical influences because the affects the problem solving process in the same way and we have the advantage of a smaller number of external influences.

The creation of the goal matrix has another advantage for the decision maker. He has to think about all details of the problem and so he gets a better view on it. Therefore the decision maker gets a different point of view by gathering all relevant information because the level of knowledge about the overall situation changes. In real life people does not have full information about their environment and we are acting in the best way according to our level of knowledge. Therefore gathering information about any problem helps to realize its whole nature.

But the amount of work by the creation of this matrix has to be in proportion to the size and difficulty of the problem. The aim of such a mathematical procedure is to map a real problem on a simplified model by dropping the data which is unimportant for the decision finding process. But this procedure is dynamic because the results of the model often show that the previous found alternatives are no feasible solutions for the problem. With this information the search for alternatives can be continued under new aspects.

It is also important to mention, that these decision models are not objective because the personal point of view of the model builder is put into the model. Therefore new inputs and concepts of none involved persons often help to find better alternatives for a given problem.

4.3 DECISION RULES UNDER UNCERTAINTY

Making decisions under uncertainty is the core problem of decision theory. It has its roots in the 17th century. Blaise Pascal, a French mathematician and scientist established this area of research. His basic idea was that a problem can be solved by a number of actions that all have a different outcome with various probabilities. The outcomes can be ranked by positive and negative criteria. The way to choose the best outcome is to identify all possible outcomes, allocate outcome values and multiply them with the probability of occurrence. Then the way with the highest value should be taken. But it is not that easy to take the right decision.

In 1738 Daniel Bernoulli showed with the aid of the St. Petersburg paradox that the expected value theory form Pascal must be normatively wrong. [60] This paradox deals with a coin toss. This lottery has an infinite expected value and therefore an infinite expected payoff but for a rational person it is not worth accessing the game with a great amount of money.

To access the lottery a person has to pay a fixed amount of money. The pot start at one Euro and is doubled every toss. The game ends if the tail of the coin first appears. The win is one Euro when the tail appears on the first toss, two Euros on the second, four Euros on the third and so on. With these preconditions following question raises: What is it worth entering the game? At this point the subjective worth of money enters this decision. 1000 Euros are very much for a beggar – a great amount for an average earning person and a little amount for a millionaire. To solve this problem, we have to take a mathematical look at the expected value of this game.

The expected value is:

$$E = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 4 + \frac{1}{16}\cdot 8 + \frac{1}{32}\cdot 16 + ... = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + ... = \sum_{k=1}^{\infty} \frac{1}{2} = \infty$$

As we can see the sum of this expected value diverges to infinity and therefore the win for the player too. The large payoff appears very rarely but when it appears, the repay is far higher than the amount paid at the start. According to Pascal, it is rational to join the game at any price but no rational thinking person would enter the game because of the probability of occurrence which is very low, because it is not realistic that the tail of the coin never appears.

This demonstrates that decision making depends on many environmental factors and is not always that easy as it seems to be. Decision making depends on probability, experience and uncertainty. So we have to look closer on the motivations and causes of a decision.

Uncertainties appear in nearly every decision because of the incomplete information booth of the decision maker as we discussed at the end of chapter 4.2. Therefore the goal is to gather as much information as possible to keep the number of remaining uncertainties low. But we do not have to forget that some uncertainties always remaining because of the unpredictability of future happenings.

These circumstances are called risks. The goal of dealing with risks is the identification, description and avoidance of them. The same principles are used in software project management to minimize the uncertainties during any project. First of all risks have to be identified by a look on the predicted progress of a problem solution. During this step the experience of the decision maker plays an essential role because risks and uncertainties of former solution processes can be adapted to the current situation. Risks can affect the schedule, costs and quality of any project or decision problem. These things have to be kept in mind during the identification process because uncertainties which do not affect these objectives are not worth to deal with it further.

After all potential risks are identified a description of them must be made including probability of occurrence, severity of impact on the goals and possibilities for detection and handling of them. Some risks can be influenced and managed but there are also completely external risks which can not be managed. One example for this category can be weather conditions. If anyone is planning an outdoor event he can not influence the weather conditions directly. His only possibility is to choose a date where the probability of good weather conditions is very high.

Risk description is a very important step because dealing with risks is a major step to avoid them. It is nothing worse than forgetting or underestimating risks because they will not be handled during a project or a decision process which normally lead to an unwanted or negative outcome. If projects last a longer period the risk treatment is a continuous process because risks can appear, change or disappear during this period.

Once the risks are described preventive actions should be initiated to avoid them as far as possible. These actions need a responsible person who keeps the risks in mind to ensure a working risk treatment process. The evaluation of risks must be done on a regular basis to observe the changes of the risk situation during the process.

The following figure shows a graphical demonstration of risk treatment. It is related to the Plan – Do – Check – Act cycle which is used in various project management philosophies.

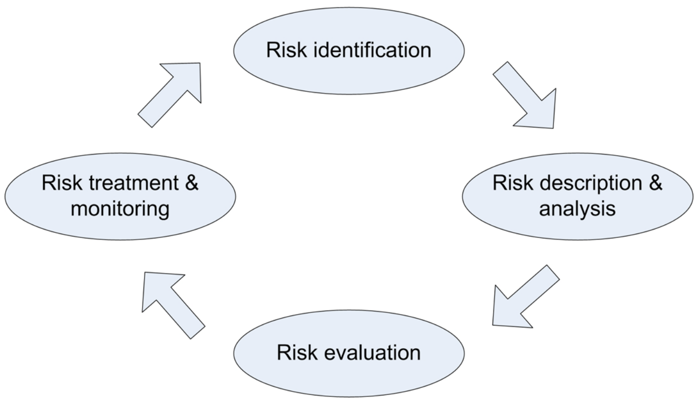


Fig. 15: Risk treatment cycle

These considerations show that handling with uncertainties is a very serious topic which should be done with care to eliminate or mitigate them. Risks are very dynamic especially in complex situations. Therefore risks have to be considered during a decision process or in a decision tool for complex circumstances.

4.4 MAKING DECISION WITH THE HELP OF WEIGHTED ARGUMENTS

As we can see in daily business arguments of decision problems do not have the same value regarding the outcome of the decision process. We can observe that there are some attributes with a greater importance than others. Because of that arguments are weighted during a decision process. If anyone has a problem to solve we can hear the following sentence often: “This is more important to me than that.” So every decision maker weights his arguments which happen implicitly during small decision processes. [28]

In case of the selection of the most adequate quality assurance strategy for a given software development project the weighting of the attributes can not happen implicitly. In fact the decision tool must provide the possibility to weight these attributes to support the decision maker in his duty.

There are several different possibilities of weighting arguments which depends on the structure of the decision tool. One alternative would be a multiplication factor for each value to express the different grades of importance. This implies that we have a predefined weight of attributes on the one hand or that the weighting can be adjusted according to the needs of the user on the other hand. Both possibilities have advantages and disadvantages.[19]

Let us assume we are choosing predefined weightings. This method is very helpful for less experienced project managers because the weighting of attributes strongly depends on the previous experiences with such problems. It is also true that a weighting depends on several input parameters, preconditions and perceptions about the outcome. But in general projects with similar structure and goals have normally similar concerns.

Another advantage of this method is that the usage of the tool is easier because the user do not have to perform the weighting for every project. This fact could increase the acceptance of the tool on the part of the project managers because the usability is higher. But predefined weightings can also be a disadvantage for experienced users because the possibility of fine tuning is not given.

Because of all these arguments weighted values are necessary to fulfill the needs regarding to the theoretical approach of decision finding. An efficient way to implement the weighting of attributes must be developed to guarantee the consistency of the prototype.

4.5 DECISION FINDING WITH MULTIPLE PARAMETERS

This thesis deals with the comparison of parameters of a software development project and various quality assurance methods. Therefore we need several parameters to make a detailed description of both types. All parameters have to be compared during the matching process of the software project and the QA methods.

To get a detailed view of the situation we must split the problem into two aspects. First there are the parameters for the description of the different methods. All parameters are part of the decision finding process. Therefore they must fit for both software development projects and quality assurance methods to create the possibility for comparison. These parameters have to be scaled to mathematical values to ensure the creation of a comparison model. Like in nearly every situation of decision finding we have to deal with two types of parameters. The first group can be scaled to discrete values but the second group can not. In case one the parameters can be described as quantifiable and in the other case we can speak about qualitative variables. Both groups characterize the methods but in case of an automated matching process they must be treated differently.

The second aspect deals with the fact that the parameters of the software development project must be compared with many vectors – each for a different quality assurance method. Therefore we have multiple parameters and multiple alternatives. This brings a higher level of complexity into this situation.

Because of the fact that we are dealing with a greater number of parameters we can subsume them into a vector of variables. This vector represents the specification of the methods. With the help of the vectors the different methods are mathematically comparable and therefore the possibility of automation is given.

4.6 DECISION MAKING AND QUALITY ASSURANCE

The prototype which is discussed in the practical section of this thesis should select the best quality assurance methods for a software project with given preconditions. These conditions have to be ranked under various aspects. Therefore the prototype needs measurement methods and a decision basis to select the most valuable quality assurance methods for any project.

It is at hand that some of the needed arguments are quantitative values which can be scaled to make them comparable. But there are also some attributes for projects which can not be brought into any scaling or ranking. We can call these attributes qualitative values because they describe projects in detail but these details can not be quantified. Therefore these values must be compared in a different way as the quantitative values.

Qualitative values can only meet two conditions – match or no match. This implies that it is possible to conduct a direct matching between the project attributes and one quality assurance method. But a ranking of the different QA methods relating the project attributes is not possible. Therefore qualitative attributes have to be seen as additional information to gain a reasonable description of the project but putting them into the ranking of the QA methods is not applicable.

The information of this chapter is essential for the design of the prototype because all relevant issues and theoretical approaches regarding decision finding are necessary for the functionality of this tool.

Now we can concentrate on the actual problem by formulation the problem and its solution with the help of the theoretical concepts of the last three chapters.

5 RESEARCH QUESTIONS

This chapter gives an overview of the research questions of this thesis and shall be a transition to the practical part of this thesis. The questions shall cover the basic approach of this thesis to find reasonable answers. They shall show that the idea of a matching between software project attributes and quality assurance methods is possible and useful in practical. As we have seen in section three requirements analysis are already done with the help of other approaches like the ATAM process but this process does not cover the selection of the best fitting quality assurance strategy. Therefore the following questions gear towards the possibility of a decision tool for the selection of the most appropriate quality assurance method for any project.

5.1 BASIC RELATIONS BETWEEN THE DIFFERENT METHODS

After a detailed look into the field of software development and quality assurance in this area the first issue of this thesis is the relation of different software development quality assurance. Are there some these methods fitting better together than others? Are some quality assurance methods more effective for a given software development method?

During the theoretical part of this work it seems to be obvious that these expectations are relevant. The prototype discussed in the practical section shall link software development processes with quality assurance methods, bring more detailed answers and gain a fundamental understanding of these relationships.

Research question number one:

What are the Basic relations between software development processes and quality management methods? Is it possible to compare them with the help of common parameters?

This question is the basic questioning for this thesis. If the linkage can be verified the following questions are at hand because they representing the needs to implement this theory in a practical model to solve this problem.

The practical model is a part of the concept of quality strategy evaluation processes. It shall provide a decision finding tool for project managers to find a suitable quality assurance strategy according to different business processes and software development strategies which is one of the main topics of this thesis.

5.2 REQUIREMENTS OF THE PROTOTYPE

The next issue is the linking of the sum of criteria to a decision model for this topic. Which kind of data must be collected out of the software development processes to get a decision base for the evaluation of upcoming projects? Which data can be stored from the different quality assurance methods? Is it possible to find out matching criteria for this purpose? How far can an automated tool cover all necessary decisions of a software project manager?

It is obvious that some attributes are able to be measured quantitatively. But there are certainly some qualitative attributes which can not be covered in a fully automated model. Therefore it is important to define both categories of attributes and provide them to the project manager in order to ensure that all criteria are considered during the selection process of the quality assurance strategy and that the developed model is consistent and complete.

Research question number two:

What are the requirements for an experience based decision model for quality assurance methods? Is there a possibility to cover all relations for the decision process?

5.3 NATURE OF DECISION CRITERIA

The collected data from software engineering processes and from quality assurance methods have to be stored in vectors to make the data comparable for the decision process. If there is a ranking for quality assurance methods according to a chosen software development process key criteria for the evaluation process must be defined. Which characteristics of a development process have influence on the outcome of a quality assurance method? Are these characteristics of the same importance or is there a need to weight them? How can outputs of a quality assurance method used as an input for the following steps of quality assurance?

These questions are essential for the functionality of the whole concept. Like in other evaluation processes, attributes have different values according to the scope of the current problem. Without carefully selected decision attributes an automated evaluation is not possible. But the automation of this process is necessary to avoid poor educated guesses from less experienced project managers.

Another important point is to call attention to qualitative decision attributes which can not be covered with the help of any tool. These attributes complete the inputs for the decision process and must not be forgotten.

<u>Research question number three:</u>

What are the decision criteria for choosing the right quality assurance strategy? How can such an evaluation take place? Are there different types of decision criteria?

If the research questions can be answered with “yes”, it should be possible to implement all necessary steps to build a working prototype of such a decision tool.

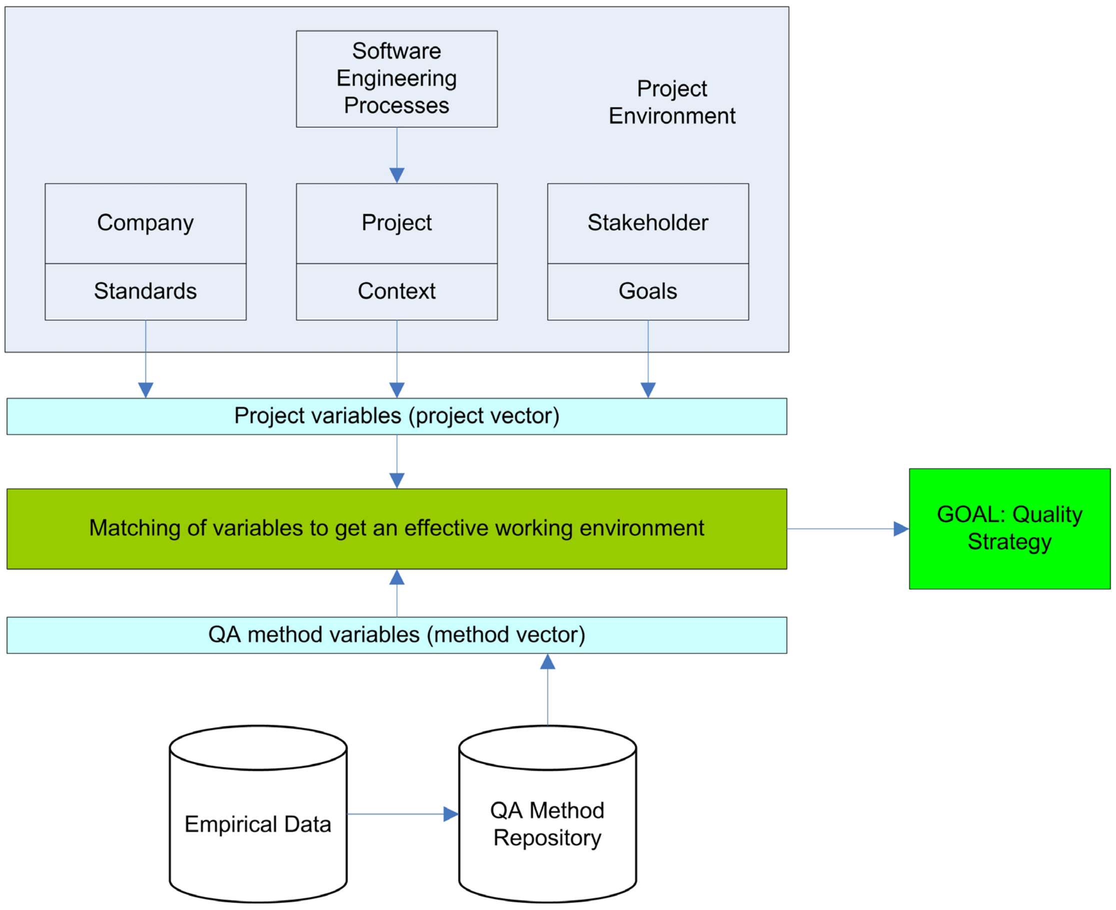
To answer all these questions a prototype for a decision tool will be created and the outcome of the decision process will be discussed. The prototype will be the main part of the whole concept of choosing a suitable quality assurance strategy. The following chapters are dealing with these topics.

6 CONCEPT OF A QA STRATEGY EVALUATION APPROACH

As mentioned before the main part of the solution of the problem is the development of a tool based decision model for quantitative project related attributes. This chapter shows the need of this model and the way to the implementation of a prototype which can be used as decision support for software project managers.

6.1 BASIC APPROACH OF THE MODEL

The following picture shows the need of a connection between software engineering processes and quality management strategies.



Software Engineering Processes
Project Environment
Company
Standards
Project
Context
Stakeholder
Goals
Project variables (project vector)
Matching of variables to get an effective working environment
GOAL: Quality Strategy
QA method variables (method vector)
Empirical Data
QA Method Repository

Fig. 16: Research area of this thesis [51]

This figure depicts the research area of this thesis. Today much experience is gained in the software engineering sector which is shown at the top of the figure. Also the field of quality assurance is widely discovered.

The research gap is shown in the middle of the figure. The dark green area is the linkage between both areas and the need is to discover the matching methodology in order to find a simple way to create a suitable overall quality strategy for every type of software projects. Therefore the main task is the choice of the most suitable quality assurance methods and the combination of them to an effective quality assurance strategy. The effectiveness of the strategy must be ensured on the part of costs, finding of errors and reuse of data. In order to this matching criteria seem to be the best way to a successful solution of this problem.

The idea to fill this research gap is to develop a mapping process of attribute vectors of both, the software process and the QA methods. The following figure describes this process graphically to get a better imagination of this idea:

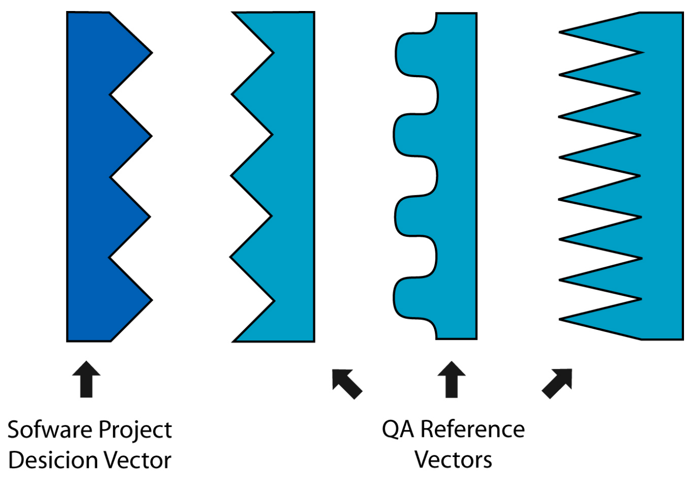


Fig. 17: Graphical overview of the mapping process

The figure should be understood as a schematic explanation of the mapping process. Let us assume we want to build a rectangle out of these building blocks. The main goal is to find the best fitting quality assurance solution for a given software project environment. The attribute values of the decision vector of the project build a specific surface of the vector. This surface is compared with the surfaces of the QA reference vectors to find the best fitting piece to build this rectangular. As we can see in this figure the leftmost brick fits to this specific project vector.

Now let us have a look at the used attributes for developing the prototype. There are two categories of attributes which have to be considered for the solution of the matching problem. The first are quantifiable attributes like team size or project costs. All these criteria can be scaled into a predefined range of values to ensure the comparability with reference values.

The second type is qualitative attributes which does not fit into any reasonable ranking. But these values are also important for the choice of the best fitting quality assurance strategy. In order to remind the project manager of the necessity of such values the solution must provide a possibility of dealing with them.

So how can such a tool support look like? Obviously we need quantitative and qualitative attributes which can describe software development processes and quality assurance methods to get a common basis for the comparison of both processes. These attributes must be scaled into discrete values to make the comparison easier. Chapter 6.2 is dealing with this problem and shows a possible solution. Chapter 6.3 shows that these values are useful as well for quality assurance methods as for software project development processes.

The next step is the development of the logical background and the user interface of this tool. Afterwards a presentation sheet must be generated that can be used as decision support for the choice of the most suitable quality assurance strategy.

In chapter 3.3 we can find an overview of the ATAM process which is shows several relations to this process. As mentioned ATAM is dealing with the development of the most appropriate software architecture considering all relevant influences and risks. The background of this decision supporting tool is similar to ATAM but the referring operational area deals with quality strategies.

Therefore the concept of ATAM can be used as guideline for the creation of a trade off analysis for quality assurance strategies. An additional fact is that hardly any evaluation of whole quality assurance strategies took place up to now. The idea of the development of strategies instead of the selection of single methods enhances the whole concept because an effective combination of various methods to gain an outcome of higher quality is far more valuable for the software engineering industry.

The idea of quality assurance strategies has to grow in the mind of the project managers to use them effectively. Therefore the tool is one of two steps. The second step will be a evaluation meeting with the project team to consider the qualitative attributes and other business values which can not be calculated with any tool because normally they are far too specific to bring them into any reasonable scaling.

In order to consider all discussed arguments the most feasible solution is a decision based tool support for project managers. Therefore following chapters show the way to a prototype of this tool to help the reader to imagine the considered backgrounds and to give him an idea how such a tool can look like.

6.2 REFERENCE VALUES FOR THE ASSESSMENT OF QUALITY ASSURANCE METHODS

This chapter deals with the explanation of the software project and quality assurance attributes which are used in the prototype. As mentioned before the attributes can be split into two groups. Subchapter 6.2.1 deals with the quantitative attributes and subchapter 6.2.2 handles the qualitative attributes. The reader of this work shall get an overview of the possible values and their meaning.

It is at hand that some other attributes can be used to qualify software development processes and quality assurance methods. They values can also depend on the business scope of different companies and their business practices. Therefore the attributes selected for the prototype are general project measures and they seem to be useful for every project evaluation no matter how the special necessities look like.

The following table gives an overview of all quantitative attributes used in the prototype. These attributes are described further in the following subchapter.

Nr.:	**Attribute:**
1	Project costs
2	Project duration
3	Team size
4	Team stability
5	QA experience of the team
6	Project experience of the team
7	Risk factor
8	New technologies/tools
9	Interfaces to other systems
10	Number of components/modules
11	Reuse of pre-existing tools/code
12	Complexity of modules

6.2.1 Quantitative Values

- Project costs:

  This is one of the major variables in the characterization of every type of project in all business areas. The costs are very important for every stakeholder. If a high amount of money is spent to a project everyone involved can be assured of the importance of the project for the investors. It is also at hand, that large and expensive projects have to be planned carefully to ensure that the outcome covers the requirements at the beginning of the project. In order to that the importance of quality assurance methods grows with the increasing amount of investment.

Because of the wide range of possible values of this attribute the user input will be scaled to five discrete values which are chosen on account of experience to structure them from small projects to very large projects. So the user of the tool is able to fill in the exact costs of the project and the logic of the tool converts it into one of the five predefined ranges for the project costs.

- Project duration:

  The duration of a project is also an important characteristic. Normally long term projects are more valuable for a company than short term projects. Therefore we can assume that the importance of the project increases with the duration. We can also see a direct dependency between project costs and duration.

  Another important point regarding to the duration is the complexity. Long term projects are usually more sophisticated. Because of that there are more functions and modules to implement. Regarding to that fact the quality assurance methods have to be selected carefully to be able to guarantee the whole functionality of the software.

  Further on the longer the duration of the project the higher is the possibility of changing human resources. This means that employees can join or leave the project and their knowledge with them. New employees have to be instructed by senior team members and knowledge is lost if anyone leaves. But new team member can also have a positive influence regarding new inputs and ideas.

- Team size:

  The size of the project team causes also several influences on the project and on quality assurance efforts. The more people work on different modules the more peculiarities regarding to the way of implementation can occur. Therefore the interfaces of the modules have to be planned and checked carefully to avoid any functional problems at the end of the project.

  Documentation of good quality is also more necessary by an increasing number of involved people because the replicability is very important for other team members to understand the work of their colleagues. This is obvious for changes of modules or the interdependency between them.

- Team stability:

  The team stability has also some different effects on the project and its outcome. First we have the social factor. This means that a well attuned team becomes its own dynamics which often has a positive consequence on the outcome of the project. Good team spirit is an important factor for project teams because they are working together very close – closer than “normal” colleagues.

  Second there is the knowledge factor. Every project team has different practices even though they produce similar outputs. Therefore the knowledge of team practices is also important for the progress of any project. Well attuned teams have a higher level of effectiveness than others. It is at hand that new team members have to be introduced in the practices of the team by a senior team member which causes a reduction of effectiveness on both people involved.

  But on the other hand new people bring also new ideas into the group. Sometimes this can have a very positive effect in order to evolve the team

  As we can figure out according to the arguments impacts on the stability of the team can have different effects. So it is not that easy to evaluate the influence of this attribute.

- QA experience of the team:

  This attribute refers to the teams experience with quality assurance methods in general. If the experience is high, it will be far easier to conduct a preselected quality assurance strategy and the project manager does not have to pay that much attention on QA training of the team members.

  The efficiency factor of the used methods will also be higher which means that the detection of a number of errors can be conducted with a lower amount of time and money. This is useful because nowadays most projects are narrowing calculated regarding to quality assurance expenditures.

  The reason why this attribute is referred to QA experience in general is that it can be quantitatively estimated during the evaluation phase. Experience with single quality methods can be hardly quantified and therefore it is better to handle such attributes as qualitative values.

- Project experience of the team:

  Like the value for the QA experience this attribute deals with the overall project experience of the whole team. It is also too difficult and complex to evaluate the experience of every team member especially if the size of the team exceeds a certain number. It is also easier for the project manager to evaluate this attribute for the whole team because normally he knows his team and the software development methods used before.

This value helps to qualify the general project experience. Are there many new team members with no or less experience or is the main structure of the team always the same? Such questions can help to determine this attribute. But it is also important to consider these points in the planning phase of any project.

The handling of less experienced teams is much more difficult for the project manager than working with experienced teams. The project manager has to pay more attention on inexperienced team members because they need guidance regarding their work and the team's practices. Because of that the handling of any project depends on the independency of the team members which grows with the project experience.

- Risk factor:

  It is obvious that software projects normally have more than one potential risk which threatens the successful finishing. This attribute represents the sum of all discovered risks and is therefore the benchmark of all project risks. An experienced project manager is able to estimate the risk level of any project with the help of basic information.

  The risk level depends not only on the success of quality assurance methods because there are some risks which can not be managed with the help of them – e.g. environmental influences like sick leaves of team members or external conditions like delayed shipments of sub suppliers.

  But all risks which can be influenced directly by the project team can be included in the estimation of this attribute. Quality assurance is very important for projects with a high risk level to minimize the possible threats during all project stages. The higher the risk level the easier can severe problems occur. These problems have to be discovered and removed early to minimize the costs and maximize the chance of success.

- New technologies/tools:

  The use of new technologies is a challenge for every project team. On the one hand the people are motivated to experience something new but on the other hand new technologies or tools cause always some risks because the project team is not familiar with the handling of them.

  Therefore the application of new tools requires a well panned quality assurance to consider and check all possible problems. If new technologies are handled with care the project team is able to gain experience during the project and the technologies can be useful for upcoming projects and enhance the value and the power of the whole development team. So the application of something new has to be seen as a chance for the improvement of everyone involved.

- Interfaces to other systems:

  This attribute is relevant to determine the complexity of any project from another point of view because complexity itself can have various dimensions regarding to a software project. Some projects are very large but have only a few number of interfaces to other systems and some small projects have many interfaces to fulfil its functionality.

  In general we can say that the development of interfaces to other systems always must be done carefully to ensure a proper way of communication between the items. Therefore the necessity of a good quality assurance strategy is increasing with an increasing number of interfaces to various tools.

- Number of components/modules:

  The number of components and modules has similar characteristics like the number of interfaces. It also describes the complexity of a project. Software with a small number of components is easier to implement and the possibility of errors is not that high. Therefore the focus on quality assurance strategies increases with a higher number of software components.

  This attribute deals only with the count of the modules and components not with any similarities or the complexity of them. These two characteristics are handled separately.

- Reuse of pre-existing code:

  This value depicts the reuse of already implemented pieces of software. It does not matter if these pieces are implemented in already finished software or in the current project.

  One example for such pieces is a data base connection module for instance. If different pieces of software use a database of the same type the connection manager is normally the same. It can also happen that such modules are written externally and so they are free for use.

  If the reuse of pre-existing code is high the number of potential errors decreases because these pieces of software are already tested and implemented in other systems. The attention must be laid on the accurate implementation in the project regarding to variables and interfaces.

- Complexity of modules:

  This attribute deals with the degree of difficulty of the single modules. The number of different modules does not tell us anything about their complexity. But this is also a very important point for the qualification of a software project.

  For instance if we have a large number of modules which are all easy to implement and they are well structured the working process would not be that complicated for the project team. But we could also have only few modules with a high level of complexity. Which of these two instances is more difficult? So we can see that the number of modules and their complexity have a reverse interdependency regarding to the complexity of the whole project. Therefore both values have to be collected separately to recognize them in the evaluation of the project and the QA strategy.

6.2.2 Qualitative Values

As mentioned before qualitative attributes can not be treated like quantitative values because there are no metrics where qualitative values fit in. Because of that the evaluation of the best fitting quality assurance strategy can not happen fully automated.

But it is also important to remind the project manager of such values during the evaluation process and therefore the prototype has to deal with these attributes too. Since these attributes can not be summarized in any calculation they must be presented in another way.

One qualitative value is considered in the prototype to show a possible way of dealing with these attributes:

- Domain:

  This value represents the field of application which the software is written for. It is a good example for a qualitative attribute because there are several different domains which are possible but they can not be quantified like easy or hard to implement.

  With the help of practical experience some quality assurance methods can be classified as preferred method for a special area of application but this means not that this method is not appropriate for any other possible domain.

  Therefore the domain must be selected in the prototype but its fitting to a strategy will be only presented by qualitative match.

It is at hand that every project has several qualitative attributes. Some other examples are the following:

- Preferred quality assurance methods

- Required quality assurance methods
- Fulfilling of general project constraints
- QA methods for special requirements

These attributes can not be ranked like the quantitative values. Only a direct matching is possible – e.g. fits or does not fit. But we must not forget that these values are also important for the qualification of the project and the evaluation of QA strategies. Therefore they must be handled.

6.2.3 Interdependencies between the attributes

As it is mentioned during the description of the different attributes there are some interdependencies between them. The question is if these interdependencies are good or not. After a detailed look on this topic we can see that these relations are necessary to describe a project properly – especially for less experienced project managers.

First it makes sense to implement interdependencies into the prototype to ensure that the project will be considered from all possible points of view. This helps the project manager to take all attributes into account. With the help of correlations less experienced project managers get the advantage of self control during the evaluation phase.

Second the interdependencies can not be avoided because of the complex characteristics of every project. Especially software projects can look very similar at first glance. But after a more detailed consideration some sever discrepancies can appear – like the complexity of single components. Therefore some correlations regarding to the attributes are necessary to depict the project from all possible points of view. This is obvious to ensure the quality of the prototype.

Third interdependencies can cause some inconsistencies regarding to the evaluation of the project. Therefore the tool has to handle these possible inconsistencies with the scanning of the input parameters. Another possibility is to choose the reference values of correlated attributes carefully to ensure an outcome of good quality even if the input parameters contradict each other.

Some examples of correlated attributes form the prototype are the following:

- *Risk factor* and *number of new technologies*:

  As in the description of "number of new technologies" is written the risk of a project increases with a higher number of new tools. So the risk level should be higher if there is a vast number of new technologies except the project manager do not qualify them as very risky.

- *Number of components* and *reuse of pre-existing code*:

If many components are necessary the complexity of the project increases. But if it is possible to use already existing pieces of code or whole modules the complexity level decreases.

As we can see by these examples the topic of interdependencies is very sophisticated and its relevance grows with an increasing number of attributes. It is a very sensitive parte because the correlations take influence on the output of the evaluation tool. We have to take care of this during the implementation of the prototype and possible enhancements of the tool in the future.

6.3 MAPPING QUALITY ASSURANCE METHODS TO SOFTWARE PROCESSES

The attributes mentioned in the chapter before were introduced as variables to describe project types. But if we take another point of view on this attributes we realize that they can also be used to describe quality assurance methods.

Let us look at some examples. The project size is a very important characteristic for software projects. But it says also something about the assignment of quality methods. The bigger a project is the more critical is the assembling of the different modules because the interdependency between them is far more complex than in smaller software projects. Therefore the size can tell us something about the importance of QA methods in a project.

But there is another argument which must be mentioned – the applicability of different quality assurance methods. Not every method is as good as any other for large software projects. It depends on the characteristics of the QA method. For instance if the preparation for the control of every single module is very high the effort and the costs grow rapidly which can be a problem according to the narrow budgets for today’s software projects.

Similar things can be observed for other attributes. Let’s have a look at the “QA experience of the team”. As the name says the value deals with QA methods and their application. Teams with good QA experience can apply them easier which helps to safe time. The second meaning refers at the complexity and applicability of these methods. For instance if audits are used to check the design documents of a software project and the team has no experience with this QA method it is quite hard to communicate them that the goal of the audit is only the qualification of the document and not their personal work. An experienced team regarding to the application of quality assurance methods can handle such things easier than inexperienced teams. But this depends on the style of conducting a QA method. A test process for pieces of code also detects errors but no one will be pilloried in the presence of the whole team.

Further on quality assurance methods can be seen as very small projects themselves. They need to be planned, have an own budget, they need human resources and they are scheduled in an appropriate way. They have an effect on further stages of the software project and they have sustainable consequences. These are also arguments for characterizing these methods like software development processes. It is also helpful that QA methods are very different according to their way of detecting errors and their style of application. That makes it easier to qualify them with the help of attributes.

So we can see that the attributes are useful to characterize both – software development processes and quality assurance methods. With the help of the attributes described in chapter 6.2 a matching between these two groups can be conducted.

All mentioned Attributes can be used to evaluate the matching between software processes and quality assurance methods. The most applicable way is to build a vector of attributes and calculate the deviation between the different values. Further information about the matching process can be found in chapter 7.1.2 where the calculation sheet of the prototype is described.

7 IMPLEMENTATION OF THE CONCEPT – A PROTOTYPE

This chapter deals with the implementation of the concept of the past chapter. Because of the fact that we are dealing with a vast amount of different software project environments which are not always identical with the standard procedures of several software development processes the most suitable implementation approach is to let the user insert the values of the project attributes and match them with the reference values of the quality assurance methods.

The prototype shall represent a first solution approach of our existing problem to make software development processes and quality assurance methods comparable on the basis of scientific data and experiences with QA methods of software development companies.

7.1 INTRODUCTION OF THE PROTOTYPE

The prototype of this decision tool is realised in Excel. Its purpose is to show one possible solution for the problem in a clear and understandable way in order to prove its usefulness. It consists of four sheets which are separated because of their different functionalities and meaning to the user of the tool. The sheets are described in the following subchapters. The sheets deal with the input and output of the tool, the calculation of the comparable values, the collection of reference values and the presentation of the possible strategies.

The prototype shall cover the most important functions which have to be implemented to guarantee the most accurate decision supporting tool for the users. It is at hand that the first implementation does not cover all conceivable details and attributes. The main intention of it is to show the possibility of filling the research gap mentioned before and to show examples for the solution of several different problems during the matching process like the different behaviour of some attributes which can not be brought into any scale.

To understand the functionality of the prototype the following figure is provided. It starts with the fact that a new software project has to be developed. With the help of the direct requirements and other direct and indirect conditions an evaluation of the project attributes can be conducted form the project manager with the help of the project team.

These variables are inserted into the prototype and then automatically compared with the reference values of the quality assurance methods. The amount of the reference values proceed from scientific literature and some practical experience.

As soon as the insertion of the project attributes is completed the preferred quality assurance methods and strategies are depicted. This outcome can be used as basis for the choice of the most appropriate QA strategy considering the experience of the company and its best practises.

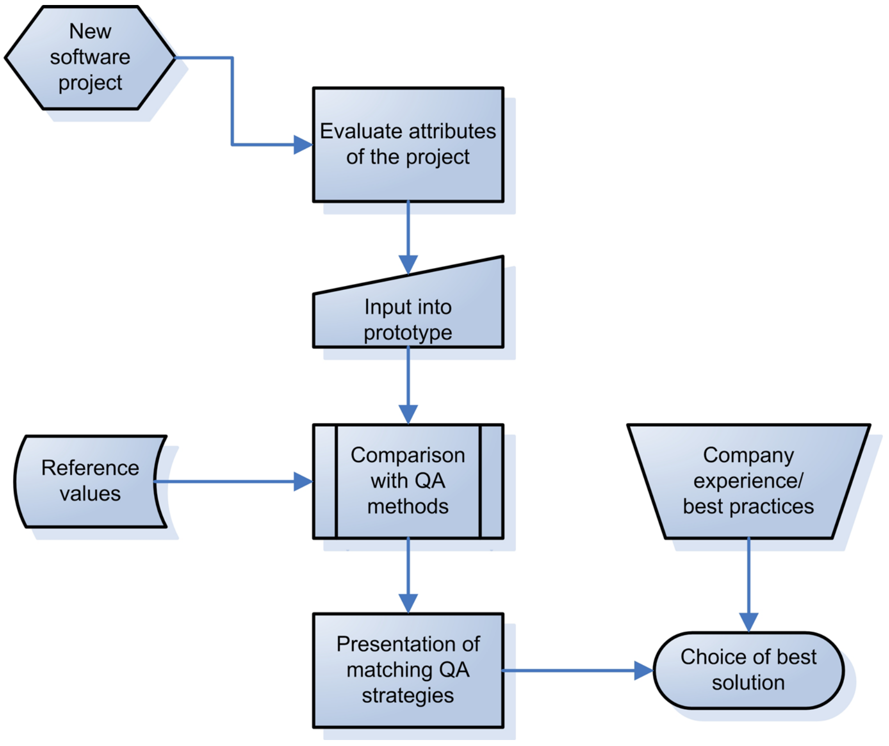

New software project
Evaluate attributes of the project
Input into prototype
Reference values
Comparison with QA methods
Company experience/ best practices
Presentation of matching QA strategies
Choice of best solution

Fig. 18: Flow of the working process of the prototype

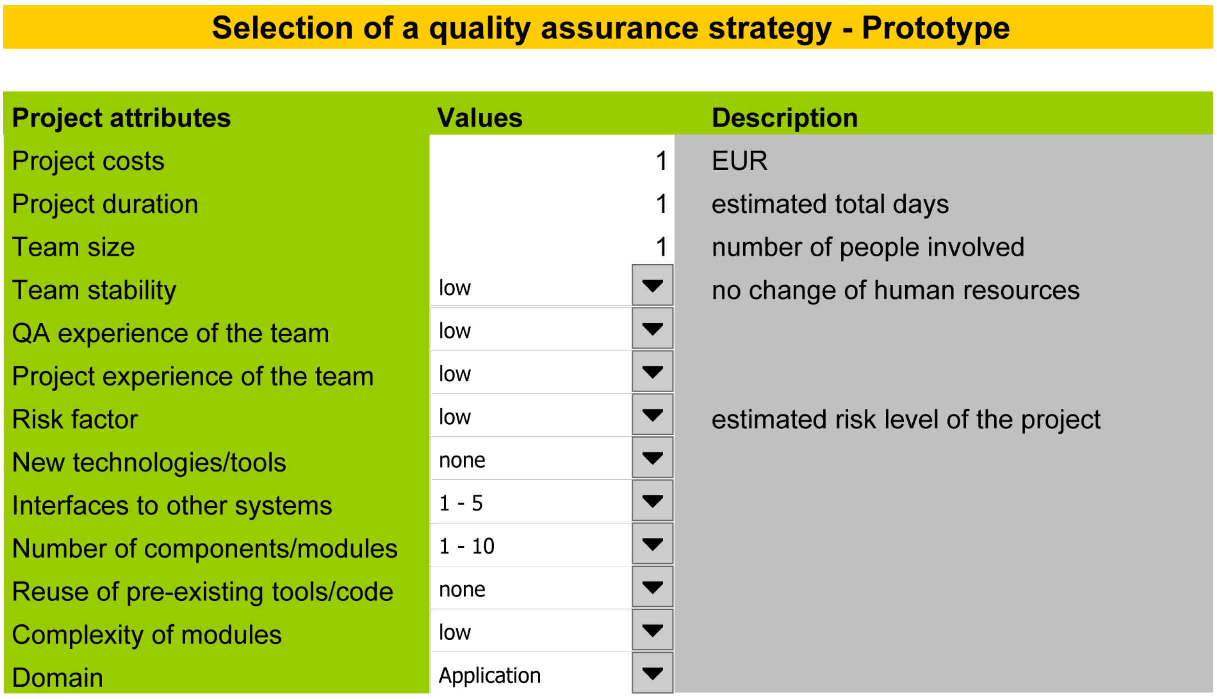
7.1.1 Input and output of the tool

The Input/Output sheet of the prototype is the logical beginning of it. With its help the user can enter the values of the attributes according to his project details. Therefore all attributes are listed here. The first three values can be inserted directly so the user doesn't have to categorize them. The other values can be entered with the help of drop down boxes in order to have predefined proposals for the values. This way of selection makes it easier for the user to qualify these attributes and helps to avoid failures of the prototype.

Behind the values there is a description field to give the user further information about some variables to ensure that the understanding of the attributes is clear.

During the input of the values the calculation and the matching are done automatically so that the output values are changing constantly. This gives the user the advantage to use the tool also in the pre project phase. At this stage he has the possibility to adjust the project attributes a little bit to achieve a quality strategy which possibly fits better to the environmental conditions of the target project. Therefore the tool has an additional possibility of application.

The figure below shows the input fields of the prototype:



Selection of a quality assurance strategy - Prototype

Project attributes	Values	Description
Project costs	1	EUR
Project duration	1	estimated total days
Team size	1	number of people involved
Team stability	low	no change of human resources
QA experience of the team	low	
Project experience of the team	low	
Risk factor	low	estimated risk level of the project
New technologies/tools	none	
Interfaces to other systems	1 - 5	
Number of components/modules	1 - 10	
Reuse of pre-existing tools/code	none	
Complexity of modules	low	
Domain	Application	

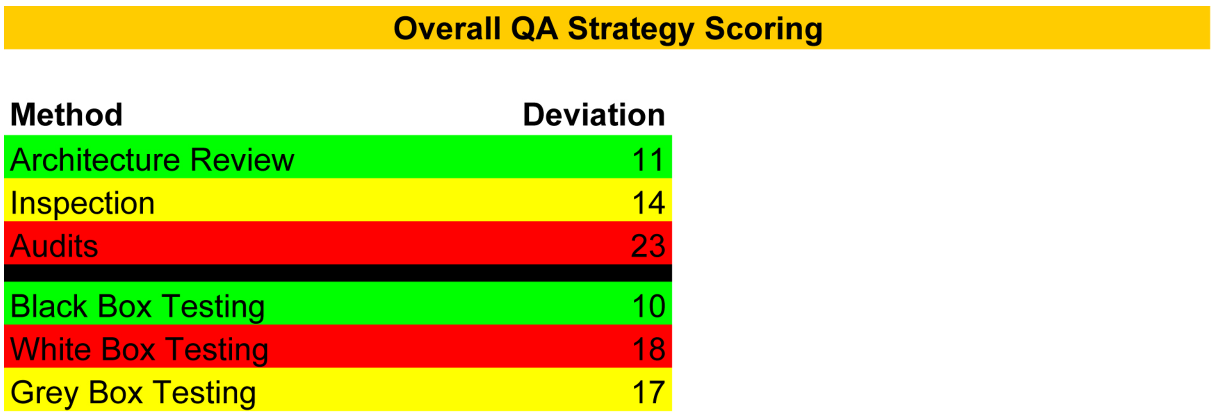
Fig. 19: Input of the different project attributes

On the same page of the tool the first qualification of the quality assurance methods are depicted which helps the user during the adjustment of the variables. The presentation of complete QA strategies is done on a separate sheet which will be described in chapter 6.4.3. This first calculation shall give the user an overview of the possible alternatives. Early conducted QA methods are not combined with late QA methods on this sheet. Here they are treated separately.

Beside the methods the deviation of the method in comparison with the inserted project values is depicted. The best fitting quality assurance method of each project phase is highlighted green, the second best method is highlighted yellow and the third method is highlighted red. These colors are used according to the functionality of traffic lights.

The green method hast the smallest deviation as we can see in the picture below. In addition the green highlighting shall help the user to identify the best method without the need to compare the deviations. But if he wants to check the value of the deviations he has the possibility to do it also on this page.

The exact deviation is far more interesting in consideration of the whole quality assurance strategies because there the values diverge much more. The figure below shows the first output of the prototype which is located below the input section on the same page.



Overall QA Strategy Scoring

Method	Deviation
Architecture Review	11
Inspection	14
Audits	23
Black Box Testing	10
White Box Testing	18
Grey Box Testing	17

Fig. 20: First Scoring of QA strategies

7.1.2 The Calculation Process

The calculation of the tool is realized on a separate sheet because of the reason that the user of the tool must not influence this process. There the input values are scaled to comparable values which are necessary to make the tool work according to the concept.

The first three values (team size, duration, costs) are split into five categories which distinguish between very small and very large. The borders of theses categories can be accommodated which makes is easier to customize the tool. The scaling to comparable values is realized with the help of the "vlookup" function of Excel.

Following description of this function can be found in the Excel 2003 online help from Microsoft:

"Searches for a value in the first column of a table array and returns a value in the same row from another column in the table array." (http://office.microsoft.com/en-us/excel/HP052093351033.aspx)

If we have a look at the team size value in the figure below the input value in the blue field is coming from the input sheet of the prototype. This is also the reference value for the lookup. It will be compared with the input column and the scale column provides the according scale which is also displayed in the blue box. The values of the input column represent the lower borders of the ranges. This must be considered if the experienced user wants to change the five ranges. The duration and the costs of the project are handled in the same way.

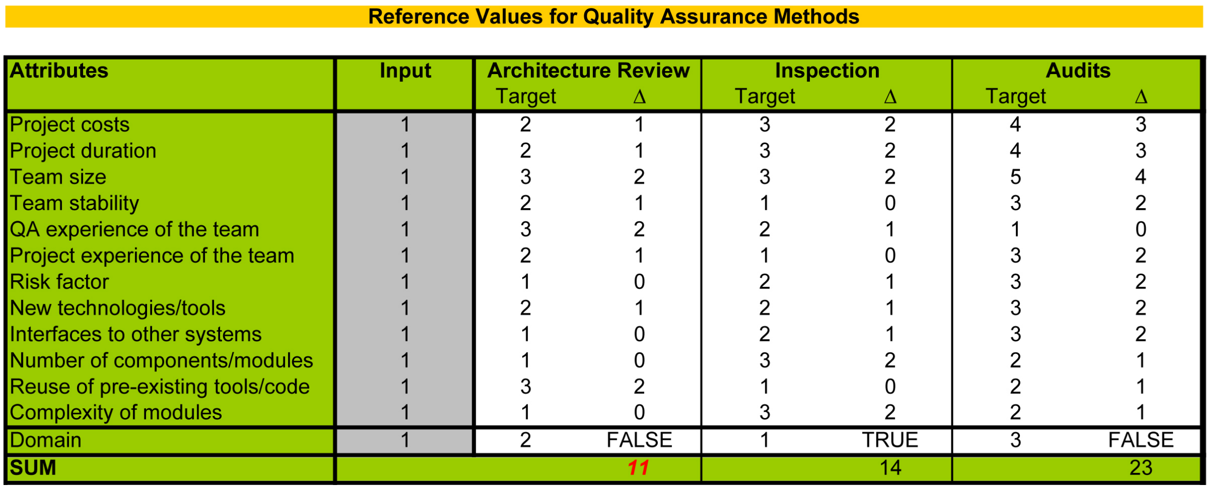
All other values are handled over the functionality of the drop down boxes from Excel. These boxes have implemented a similar function as the vlookup. The values of the input column are depicted in the different drop down fields of the user interface and the scale values are selected over the input values. Because of the fact that this type of selection needs no further scaling the input values are directly used for the matching procedure.

Scaling				
	input	scale		
Team size	1	1	input value	8
	6	2		
	14	3	scale	2
	30	4		
	50	5		
Duration	1	1	input value	105
	30	2		
	90	3	scale	3
	180	4		
	360	5		
Team Stability	low	1	input value	3
	medium	2		
	high	3		
Costs	1	1	input value	41.900
	20000	2		
	50000	3	scale	2
	12500	4		
	25000	5		
QA Experience	low	1	input value	2
	medium	2		
	high	3		
Project Experience	low	1	input value	3
	medium	2		
	high	3		
Risk Level	low	1	input value	3
	medium	2		
	high	3		
Complexity of modules	low	1	input value	3
	medium	2		
	high	3		
New technologies/ tools	none	1	input value	2
	few	2		
	many	3		
Interfaces to other Systems	1 - 5	1	input value	1
	6 - 15	2		
	16 - 30	3		
	over 30	4		
Reuse of pre-existing tools/ code	none	1	input value	3
	low	2		
	high	3		
	very high	4		
Number of components/ modules	1 - 10	1	input value	3
	11 - 25	2		
	26 - 50	3		
	over 50	4		
Domain	Application	1	input value	2
	Web Application	2		
	Data Base System	3		

Fig. 21: Calculation of scaled values

It is also important to mention that some attributes have a different range of values – e.g. “Team Size” has one to five whilst “Team Stability” has only one to three. These differences are made with intent because they are affecting the outcome of the prototype. In other words the weighting of the attributes can be handled over the range of the values. This argument will be explained in the following chapter when the matching procedure is described.

7.1.3 The reference sheet including single deviations



Reference Values for Quality Assurance Methods

Attributes	**Input**	**Architecture Review**		**Inspection**		**Audits**	
		Target	Δ	Target	Δ	Target	Δ
Project costs	1	2	1	3	2	4	3
Project duration	1	2	1	3	2	4	3
Team size	1	3	2	3	2	5	4
Team stability	1	2	1	1	0	3	2
QA experience of the team	1	3	2	2	1	1	0
Project experience of the team	1	2	1	1	0	3	2
Risk factor	1	1	0	2	1	3	2
New technologies/tools	1	2	1	2	1	3	2
Interfaces to other systems	1	1	0	2	1	3	2
Number of components/modules	1	1	0	3	2	2	1
Reuse of pre-existing tools/code	1	3	2	1	0	2	1
Complexity of modules	1	1	0	3	2	2	1
Domain	1	2	FALSE	1	TRUE	3	FALSE
SUM			*11*		14		23

Fig. 22: References for early phase QA methods

The picture above shows the three possible quality assurance methods for early project stages considered in the prototype. These methods are also mentioned in the theoretical chapter of this thesis to give the reader an overview of these methods.

The attributes are also listed here to make the adjustment of the target values of the quality assurance methods easier. The input column gets its values directly from the scaling sheet. Every quality assurance method disposes of two columns – one for the target values and one for the deviation which is represented by the delta in the figure. There the actual matching takes place and works as it follows:

The target value is subtracted from the input value and the absolute value of this calculation is inserted in the deviation field. Therefore the deviation of the matching attributes is calculated separately which makes it easier to implement additional ones. All deviations are summed up and the sum is displayed at the bottom of the deviation column. Because of that the absolute values must be used to ensure that deviations with different algebraic signs do not annul each other which would have a negative effect on the outcome of the matching.

The “Domain” which represents the example for qualitative attributes is not considered in the calculation of the overall deviation. The matching value for this type of variables can only be “true” or “false”. Because of that the value of this variable is first necessary on the presentation sheet of this tool but for the sake of completeness the matching takes place here.

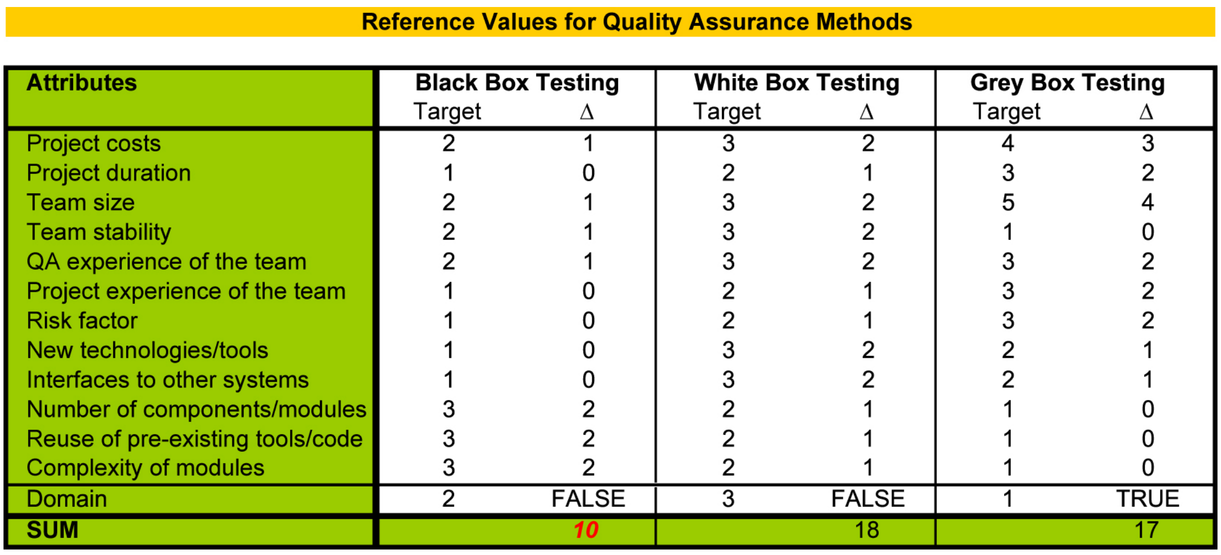
The sum represents the total deviation of all quantitative attributes between the selected project vector and the respective reference vector form a quality assurance strategy. The lower the value is the bigger is the matching between the two vectors. For better understanding all input attributes are set to one. As we can see in the picture below Black Box Testing has the lowest deviation sum which is highlighted red. Therefore this method would be the preferred testing procedure if all input parameters are set to one. The optimal case would be that the sum of deviation is zero which means that the input parameters match completely with a quality assurance method. Therefore the sum also gives an input about the grade of deviation between the compared values. This is important for the selection of the best quality strategy which will be described later.

The target values are estimated with the help of references in the scientific literature and the experience of a software development company which don't want to be mentioned here directly. But anyway these values have to be accommodated during several practical insets of the tool.

The picture below shows the different testing methods which are commonly used in late project stages after the finish of the first module codes. Both the lowest deviation sum of the first three methods and the one of the three used in the late project stages are highlighted red because they can not be compared directly according to the different field of application during the project. All other calculation and comparison methods are similar.

Just for the testing methods the target values depend on the company experience with these QA tools. It depends also on the level of automation of the testing tools. If the test tools are nearly full automated the differences between the methods are not that high. This fact shows the importance of customized adjustment of the target values.

But this sheet is nearly unimportant for the normal user of this prototype because the necessary information is shown on the Input/Output sheet and the presentation sheet.



Reference Values for Quality Assurance Methods

Attributes	**Black Box Testing**		**White Box Testing**		**Grey Box Testing**	
	Target	Δ	Target	Δ	Target	Δ
Project costs	2	1	3	2	4	3
Project duration	1	0	2	1	3	2
Team size	2	1	3	2	5	4
Team stability	2	1	3	2	1	0
QA experience of the team	2	1	3	2	3	2
Project experience of the team	1	0	2	1	3	2
Risk factor	1	0	2	1	3	2
New technologies/tools	1	0	3	2	2	1
Interfaces to other systems	1	0	3	2	2	1
Number of components/modules	3	2	2	1	1	0
Reuse of pre-existing tools/code	3	2	2	1	1	0
Complexity of modules	3	2	2	1	1	0
Domain	2	FALSE	3	FALSE	1	TRUE
SUM		***10***		18		17

Fig. 23: References for late phase QA methods

The Reference sheet is necessary to adjust the target values for the matching which is normally done by a specialist employee of the company and not by every project manager.

Furthermore the different range of the single values comes into play. The bigger the range of the values the more deviation can be created by this attribute. Because of the fact that each variable has a direct effect on the sum of deviation the possible influence of values with a bigger range is greater. For instance, if we have a rage of three the maximum deviation for this variable is two. In case of a range of five the maximum deviation is for. Therefore if the deviation sum of two quality assurance methods differ only by three points the change of one variable with a five point range can lead to the opposite outcome while a variable with a three point range can not.

The advantage of this calculation method is that the range of the values can adjusted quite easy so that the importance of variables can be adjusted to the needs of the various companies.

7.1.4 The presentation sheet for comparison

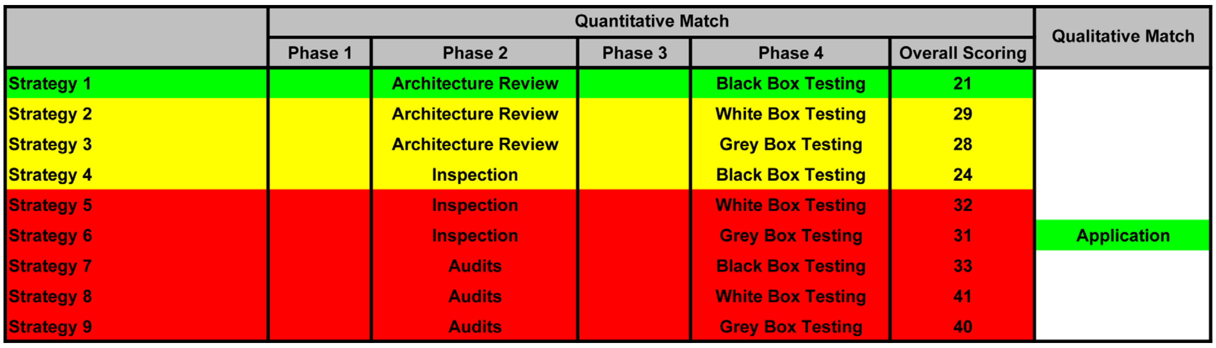
The presentation sheet gives an overview about all possible quality assurance strategies which are possible regarding to the considered methods. AS we can see in the figure below the methods for early project phases are put into the second project phase and the methods for late project stages are put into the fourth project phase.

This classification is useful if we define the four project stages as the following. Phase one represents the pre concept phase which means that the idea for the project exists but there are no further “official” documents except a vision sheet which clarifies the scope of the upcoming project. The second phase will be entered with the official project order which can be seen as the go for further serious work on the project. Here the main documents are created like the project plan, the requirements analysis, the milestone trend analysis, the risk analysis, the use case models or the architecture document. These documents have to be reviewed in order to ensure the quality of these artefacts. Therefore the three QA methods (architecture review, inspection and audits) are fitting into this stage of the project.

Phase three and phase four can show some parallel processes. The classical point of view would be that phase three represents the coding phase and phase four is assembling testing and finishing of the project. But the different software development methods described in chapter two make clear that these phases can be mixed up or in parallel. It also depends on the structure of a company and their best practices how the development process is handled exactly. So this is another point where this tool can be possibly customized for the users.

In the version discussed here the classical linear project structure is used. Therefore the testing methods for the source code of the software are settled in phase four. Phases one and three are intentionally left blank because they shall show that the tool have upgrading possibilities if they are needed but they are not necessary to prove the functionality.

The column for the overall scoring completes the section for quantitative matching. This value is the sum of the deviation of the two QA methods in each row. Like on the input/output sheet the lowest score represents the most accurate strategy.



	Quantitative Match					Qualitative Match
	Phase 1	Phase 2	Phase 3	Phase 4	Overall Scoring	
Strategy 1		Architecture Review		Black Box Testing	21	
Strategy 2		Architecture Review		White Box Testing	29	
Strategy 3		Architecture Review		Grey Box Testing	28	
Strategy 4		Inspection		Black Box Testing	24	
Strategy 5		Inspection		White Box Testing	32	
Strategy 6		Inspection		Grey Box Testing	31	Application
Strategy 7		Audits		Black Box Testing	33	
Strategy 8		Audits		White Box Testing	41	
Strategy 9		Audits		Grey Box Testing	40	

Fig. 24: Presentation of strategy overview

To make it easier for the user the best strategy regarding to the quantitative matching is highlighted green. The strategies which are ranked form the second to the fourth place are yellow coloured and the remaining solutions are inked red. So the most feasible solutions are easy to find in the list. In the figure above “Strategy 1” is the winner followed up by number four.

Let us assume that another strategy has also an overall scoring of twenty one both would be coloured green. The same things happen with the yellow inked rows.

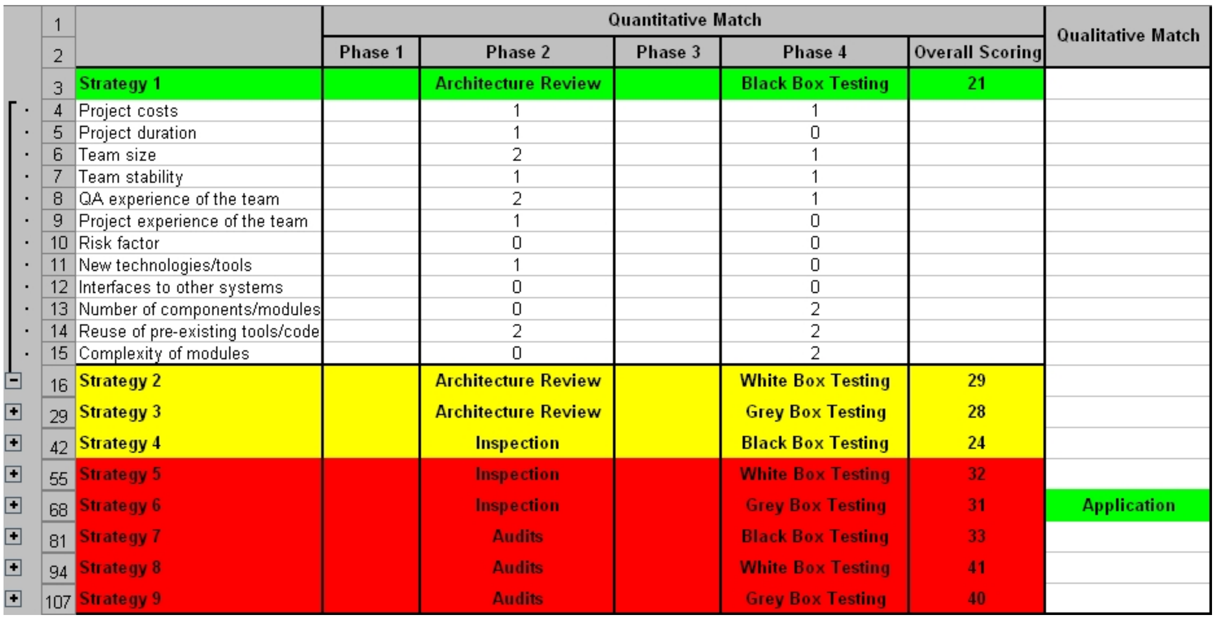
A possible solution for the matching of qualitative attributes is shown in the rightmost column of the picture. If both quality assurance methods in a row are applicable for the selected domain, the name of the domain will be shown besides the overall scoring of this particular row. It is green coloured as well to ensure that it shines out even though it is standing beside a red highlighted score as we can imagine it in the figure above.

As we discussed before, the tool should be used as guidance to identify the most accurate quality assurance strategy but the project manager has to estimate the importance of the qualitative attributes himself because they are not fitting into any reasonable scoring. Because of that the manager of the project needs possibly more information about the deviation of each attribute of his preferred strategy in order to have an idea which values the causes for the deviations are.

To help the user in acquiring this information below every strategy the variables are listed and the deviations of them can be found below the names of the QA methods. The list can be extended by using the “plus” symbols at the very left of the Excel sheet. The opened groups can be hidden with the “minus” symbols. These things are shown in the picture at the next page where the first strategy is extended and the others are not.

The reason why this additional information is normally hidden is that the presentation sheet would be too confusing.

This illustration has the big advantage that all necessary and important information is presented. For instance we can see that the estimation of the risk factor of the project fits perfect to the quality assurance methods of both phases while the reuse of pre existing tools has a deviation of two in both times. So the quality of the single attributes is easy to link with the selected quality strategy which is also necessary for further decisions.



1		Quantitative Match					Qualitative Match
2		Phase 1	Phase 2	Phase 3	Phase 4	Overall Scoring	
3	Strategy 1		Architecture Review		Black Box Testing	21	
4	Project costs		1		1		
5	Project duration		1		0		
6	Team size		2		1		
7	Team stability		1		1		
8	QA experience of the team		2		1		
9	Project experience of the team		1		0		
10	Risk factor		0		0		
11	New technologies/tools		1		0		
12	Interfaces to other systems		0		0		
13	Number of components/modules		0		2		
14	Reuse of pre-existing tools/code		2		2		
15	Complexity of modules		0		2		
16	Strategy 2		Architecture Review		White Box Testing	29	
29	Strategy 3		Architecture Review		Grey Box Testing	28	
42	Strategy 4		Inspection		Black Box Testing	24	
55	Strategy 5		Inspection		White Box Testing	32	
68	Strategy 6		Inspection		Grey Box Testing	31	Application
81	Strategy 7		Audits		Black Box Testing	33	
94	Strategy 8		Audits		White Box Testing	41	
107	Strategy 9		Audits		Grey Box Testing	40	

Fig. 25: Enhanced information for QA Strategies

This first issue of this tool consists of all necessary functions to use it for a test run to evaluate the effectiveness of it. This will be done in chapter seven which basically deals with this topic. But now we have to consider some thoughts about the automation of this prototype and the necessity of some of its functions.

7.2 AUTOMATION APPROACH

As we can see in the chapter above an automated comparison between the software development project and several quality assurance methods can be handled with the help of this prototype. But there must be a distinction between quantitative and qualitative project attributes because the qualitative values can not be ranked like the quantitative ones.

The design of the prototype allows a prompt adaption of the outcome values during the insertion of the input values. This has the advantage that project managers can change single variables in order to see the effect on the outcome.

This fact is advantageous for project managers who have influence on some project parameters. This situation can occur in small companies where employees have more than one responsibility. An example for this is that some project managers have also the responsibility for internal resource planning in small companies.

A second advantage for companies is the possibility of the reference value adaption. As said before every enterprise has different structures and core business values. The employees have also different levels of information regarding to project and quality assurance methods. Therefore it is imperative for the companies that they can adapt this tool to their own special needs which can lead to a better output for further projects.

This could be improved by additional post calculation analysis to evaluate the effectiveness of the prototype. In case of finding needs for improvement of the tool the enhancements can be made by qualified internal employees and do not have to be made by external personnel who are normally very cost intensive.

But this argument for self modulation brings also some negative points from a scientific point of view. If such a decision tool is not handled by an independent party the necessary data form the users are harder to collect for the improvement of the system. If the problem reporting and module management is handled centrally this process would be far easier. A good example for it would be the problem reporting of some operation systems or application systems in the business area. If there any problems occur an automated problem report is sent to the developing company with all necessary information to fix the bug during the next service upgrade.

The same reason is applicable to the scientific improvements of such decision tools. The development of new decision models or improvements for the matching of qualitative attributes for instance would be easier if the tool developers are able to gain information from the users because the practical approach should always be the input for theoretical improvements. The value of scientific objectives is only given if the can be practically used.

Another important point is that a wide spread collection of information about problems and improvements can lead to a broad variation of new enhancement approaches which possibly not considered by the developers because they can not have the insight to all different business situations of the users. So it is very important to integrate the using companies into the improvement process to the tool.

Last but not least there are also commercial arguments for the protection of some adjustment tools and other modules. If this decision tool turns out to be successful the commercial exploitation should not be forgotten because the selling of this tool can finance the further development. It is absolutely clear that the prototype is an untradeable version. The goal of the current version is to evaluate the feasibility of this decision theoretical approach and the success of the matching strategy which underlies this approach to a solution.

8 PROTOTYPE EVALUATION

This chapter deals with the weighting of the reference values and several different approaches which can lead to a high sophisticated system of reference values. The second part of this chapter is about a practical example for the usage of the prototype to receive a better impression if this approach is useful or not. The third section presents the results of the evaluation and the effectiveness of the prototype described in the previous chapters.

8.1 WEIGHTING OF REFERENCE VALUES

The weighting of the reference values is imperative for the operability of the decision tool because these values are compared with the input values of the user. There are two main influences for these values. On the one hand we have the weighting according to scientific reports and business experiences and on the other we have the customization according to the single business structure of a company that uses this tool.

This subchapter deals with the first group of influences because with this kind of information the prototype or improved versions of it can be adjusted. The scientific literature of quality assurance methods and their application is quite vast. But this whole bunch of literature only touches this specific area of interest. In other words there exists no detailed information of weighting different quality assurance methods against each other or different project types.

Because of that the weighting of the comparison parameters is not that easy as it seams to be. We have to read between the lines of many papers which deal with this topic to gain an overview of the advantages and disadvantages of different methods. Let us take a look at a practical example of this problem out of scientific literature:

"However, the scope of this survey is limited in so far as we do not consider the work on other static analysis techniques, such as audits or walkthroughs, although these techniques may be equally or in some situations even more effective for achieving the stated goal." (*[33], p.1*)

As we can see it in this statement a direct comparison of pros and cons is rarely made and in this case the example deals with quite comparable methods. It is much more difficult to find comparisons between quality assurance methods which are absolutely not related to each other.

Therefore it is necessary to find reports from the practical side of this topic or to obtain help from companies which are working in the software development area and using QA methods. The estimation for the values of the reference parameters for the prototype proceeds from several scientific reports and books of quality assurance methods and the practical experience from a small software development company which does not want to be mentioned here directly. The outcome of the estimation process is depicted in the following figure.

Reference values for Project parameters						
	Architecture Review	Inspection	Audits	Black Box Testing	White Box Testing	Grey Box Testing
Project costs	2	3	4	2	3	4
Project duration	2	3	4	1	2	3
Team size	3	3	5	2	3	5
Team stability	2	1	3	2	3	1
QA experience of the team	3	2	1	2	3	3
Project experience of the team	2	1	3	1	2	3
Risk factor	1	2	3	1	2	3
New technologies/tools	2	2	3	1	3	2
Interfaces to other systems	1	2	3	1	3	2
Number of components/modules	1	3	2	3	2	1
Reuse of pre-existing tools/code	3	1	2	3	2	1
Complexity of modules	1	3	2	3	2	1

Fig. 26: Reference values

All parameters are inside the same range like the input values can be. If the value of an attribute would set to high or to low the field changes the colour form white to red to show the maintainer of the tool that the particular value is out of range. If all parameters are within the range a theoretical minimum deviation of zero is possible. That implies a perfect match between the selected project parameters and one of the QA methods. Since the reference values can not be defined exactly a perfect match would not mean that the highlighted method is “absolutely” perfect for the selected project. Therefore the importance of the reference values is at hand and this fact has to be remembered during the usage of this tool.

Another theoretical approach could be that the reference values intentionally have a different range than the input values. This idea has effect that the values can be used to regulate the importance of the comparison parameters not over the rage of the input values but with the help of the reference values.

The second feature would be that the theoretical possibility of a zero deviation is not possible. This has the advantage that the user of the tool is never in the situation to believe that he has found the absolutely perfect quality assurance method for his special type of project which can be a psychological advantage for the success of a decision tool like that. If the user is of the opinion that he found the perfect QA method he could easily forget that this tool is designed to support his decisions and not to decide instead of him.

But this alternative approach is not realized in the first version of the prototype. Nevertheless it is important to consider all possible ways of adjustment and improvement to overview all theoretical variations in order to be aware of the complexity of this research area.

A possible disadvantage of the current rage of the reference values could be that the small distances create also small distances between the different quality assurance strategies. These small gaps could convey the wrong impression by the user that the differences between the single strategies is not that big which can result in the delusion that the tool is not expressive enough.

But he can realize by a detailed consideration that these small gaps are only the result from the small distances between the reference values. Therefore it is important to give the user the impression that the numeric values of the strategies have not the same severity as the ranking of them. If it turns out that the users are too fixed on the numeric values the alternative of greater distances between the reverence values could be quite valuable.

8.2 EXAMPLE FOR THE FUNCTION OF THE PROTOTYPE

To show the practical relevance of this prototype let us have a look at the following example taken from a situation which happened in a small company to ensure that such a case could happen in real life. In the first part of this section the some details of the company and the specific project are provided to get a better overview of this practical case.

It deals with the implementation of a web application for a service company. Their business is the lending out of media of films and computer games like videos or DVD's. The company wants to have an online reservation system for games and films with the following requirements:

- User accounts with their personal history and lending conditions
- Product management with stock number and lending history
- Billing with several different possibilities like credit card or pay box
- Employee accounts for managing orders and statistical analysis
- Employee interface for the preparation of orders
- Secure connection for users and employees to the system
- High level of failure safety
- Access over standard internet, mobile phones and smart phones

These requirements are the basis for the offer of the vendor. This offer was laid by a small company with nine regular employees. Additional human resources were bought in addition during times of high workload or big orders. But these external resources work quite often for the company. Therefore they know their business processes and their project development culture.

The internal evaluation of the project had the following output: The estimated working hours for this project are approximately six hundred with a margin of ten percent up and down. Due to the fact that the company is working on several projects at the same time the sales manager estimated a project duration of four to five months with a team size of eight employees. According to an internal hourly rate of 64 Euros the costs of the working time are about 38400 Euros.

The implementation requires special programs and test equipment to fulfil all requirements of the customer. These additional costs will be about 2500 Euros. The Standard project costs regarding printing of documentation, travelling costs and miscellaneous are calculated with approximately 1000 Euros.

All positions were summed up result with a sum of 41900 Euros. This value is necessary for the prototype input. It is imperative to use the internal project costs for the calculation with the prototype because these are the real costs the company has to face and to work with. The price of sale includes a gross margin which is necessary for the company to gain profit. Therefore the selling price is inapplicable for reasonable calculations. Regarding to the input parameters of the tool following information is also important:

The team stability of this company is very high because they have only one new employee who is permanently appointed and the freelancers work regularly for the company.

The company has experience regarding inspection and reviews. Audits are normally not conducted. The testing procedures are dimensioned for black box testing and white box testing. Grey box testing is not realised. Some of the testing procedures are automated. Therefore they prefer the used procedures. Because of the experience with these different quality assurance methods the value for this input parameter can be set to medium.

The company exists nearly ten years and its core business is customized software for small and medium sized customers. This business did not change since the foundation of the company. Therefore the project experience is high because everyone knows his business and the way to deal with projects and the customers.

Basically the risk level of every project is at least medium because the software products of this company are customized and the clients always want to have a special fitted product. In case of this web application the risk level should be set to high because there are some new technologies where the company do not have many experience with them.

As we can read out of the last paragraph there are some new technologies which have to be used in this project to satisfy the customer. Therefore the value for new technologies should be set to “few” to achieve consistency regarding to the input parameters.

The software must be updated regularly with the new products of the customer. There should also be an interface between the internal accounting software and the web application. Because of that the count for the interfaces to other systems must be set to 1-5.

The overall number of components and modules for this web application can be estimated between 26-50 including all supporting modules for data base connection, data security, privacy, backup and the basic functions.

In this case of application the reuse of pre-existing tools can be set to high because there are some standard functions for the handling of web applications. The customer did not specify some particular data base system. Therefore the preferred system of the company can be used so this circumstance safes time and money for all involved parties. The same preconditions can be observed by some other modules. If there are no modules or code fragments which can be used from former projects the development effort must be calculated higher than it is done above. This fact effects project costs, duration and the parameter of new technologies.

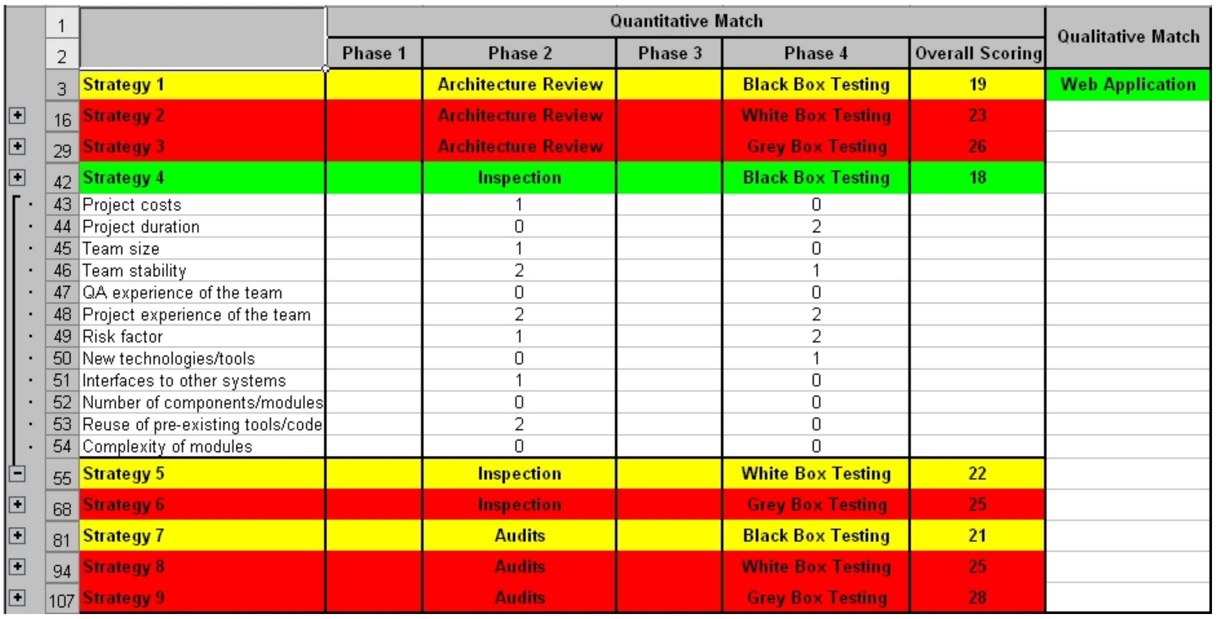
Last but not least we have to qualify the complexity of the modules which have to be developed during the project. Because of the fact that some new technologies have to be used and that some requirements need special interest to guarantee safety issues the complexity of the modules should be qualified as high.

All the information on the last pages can be used to feed the prototype. To cut a long story short here are the input values for the tool:

- Project costs: 41900 EUR
- Project duration: 105 days
- Team size: 8 people
- Team stability: high
- QA experience of the team: medium
- Project experience of the team: high
- Risk factor: high
- New technologies/tools: few
- Interfaces to other systems: 1 – 5
- Number of components/modules: 26 – 50
- Reuse of pre-existing tools/code: high
- Complexity of modules: high
- Domain: Web application

These input parameters are used to configure the prototype of the decision tool for the selection of the most suitable quality assurance strategy.

The outcome of the prototype is discussed with the aid of the presentation sheet:



		Quantitative Match					Qualitative Match
		Phase 1	Phase 2	Phase 3	Phase 4	Overall Scoring	
3	Strategy 1		Architecture Review		Black Box Testing	19	Web Application
16	Strategy 2		Architecture Review		White Box Testing	23	
29	Strategy 3		Architecture Review		Grey Box Testing	26	
42	Strategy 4		Inspection		Black Box Testing	18	
43	Project costs		1		0		
44	Project duration		0		2		
45	Team size		1		0		
46	Team stability		2		1		
47	QA experience of the team		0		0		
48	Project experience of the team		2		2		
49	Risk factor		1		2		
50	New technologies/tools		0		1		
51	Interfaces to other systems		1		0		
52	Number of components/modules		0		0		
53	Reuse of pre-existing tools/code		2		0		
54	Complexity of modules		0		0		
55	Strategy 5		Inspection		White Box Testing	22	
68	Strategy 6		Inspection		Grey Box Testing	25	
81	Strategy 7		Audits		Black Box Testing	21	
94	Strategy 8		Audits		White Box Testing	25	
107	Strategy 9		Audits		Grey Box Testing	28	

Fig. 27: Result of the calculation example

As we can see strategy four is ranked first followed by strategies one, seven and five. The matching of the qualitative parameter rates strategy one as best solution because both quality assurance methods of this strategy are appropriate for web applications.

The picture shows that the overall scoring of the best ranked strategies is very close. The gap between number one and number four are only four deviation points. But this shall not convey that the fitting of these strategies are nearly equal. As said before the low deviations come from the small ranges of the comparison values. This can also be seen if we look at the gap between the best and the worst strategy which is only ten score points.

If we have a look at the deviations of the reference variables we can see that black box testing fits better to the selected values than inspection but these methods are not directly comparable because their target is a different phase of the project. But both QA methods have a small sum of deviation therefore both methods qualify this strategy to be the preferred one.

With this information the project manager can develop the best fitting strategy for his project. This evaluation is necessary because the gap of the quantitative scoring between the two best strategies is only one point but the second ranked strategy matches the qualitative requirements. So the project manager has to assess the importance of the qualitative match in contrast to the quantitative scoring. Furthermore he has to consider the environmental conditions of his company and the experience of his project team regarding architecture reviews and inspection. Black box testing appears in both cases so he does not have to take care about the testing phase of his project. Even the strategy ranked on the third place consists of black box testing in phase three. In this case he can be certainly sure that this testing method is the best for this kind of project. The decision for the best quality assurance method during phase one depends therefore on considerations which are not directly measurable.

As described at the beginning of this section the project was already conducted. The team used the best fitting strategy intuitively because these methods are part of the best practices of the company. But the output of the prototype is not surprising because the reference values of the current configuration of the tool are influenced by the experiences of the company.

This fact proves us that the basic considerations regarding to the logic behind the prototype are working properly. But this situation can be treacherous too because if the reference parameters are adjusted too much to the preferences of one company the tool puts out what the company wants to have and so the output is not objective anymore. To avoid this problem the basis of the reference values has to be a wide spectrum of different experiences from different companies to guarantee the objectivity of the tool.

8.3 PRESENTATION OF RESULTS

As we have seen in chapters six and seven the practicability regarding the comparison between parameters of a certain software development process and several different quality assurance methods is possible. But we have to consider the fact that it is not a simple matching process of some parameters.

The theory behind this prototype is the development of a vector of attributes which can characterize both software development projects and quality assurance methods in order to be able to conduct a matching process between them. Here the matching takes place with the help of the calculation of an overall deviation between two vectors of attributes. Because of that the attributes have to be scaled to discrete values to make this calculation operation possible.

Because of the fact that we are dealing with deviations the lowest deviation of the reference vectors from the quality assurance methods is ranked first in the presentation of results.

During this scaling process some attributes were found which can not be scaled because they have no possibilities ranking them. These attributes are called qualitative attributes and are handled separately with the help of a direct matching process.

The idea of best fitting quality assurance strategies is realized with the help of splitting the project into several phases and assigning all QA methods which are fitting into this phase. The best strategy is selected over the lowest sum of deviations of reference vectors in all phases.

Because of the fact that this tool should support the project manager in his decision of choosing the right quality assurance methods for every phase of the project, all possible combinations and the corresponding deviations are depicted on the presentation sheet. Therefore the project manager has the possibility to compare the different strategies by himself and furthermore he has the possibility to take other arguments into account which are not considered in the decision tool. So the project manager does not lose his flexibility by using the decision tool.

Anyway the tool supports the project manager in his decisions but it does not replace him. The last power of decision has the project manager or his superiors.

Because of the fact that the area of research is a wide spread field with complex routines and behaviours in it we can observe that it is quite difficult to find the right adjustment of the prototype in order to ensure a useful outcome.

There are various different aspects of necessary tool adjustment to provide a correct result:

1. Basic adjustment

   The current version of the tool is adjusted according to the knowledge of a small software development company and some arguments out of the specialist literature of this research area. As we can see in the practical example of the prototype more sources for adjustment of reference values must be taken into account to ensure a basic objectivity of the decision tool. Otherwise the outcome of the tool is predictable because the reference values are only based on the experiences of the using company and this is not the idea of a decision support tool. So an experience base must be developed to guarantee an independent decision base.

2. Individual adjustment

   Although we described in the last paragraph that we need a broad reference base, individual adjustment is necessary to be able to handle the peculiarities of any company. The tool should be useful for every type of software development projects. Therefore it is necessary to meet the requirements of all companies from small to big ones. It is at hand that a smaller company does not have the possibilities to handle every different project circumstances and is also not able to conduct every type of quality assurance methods.

   Because of those facts the tool should have the possibility of making pre selections in an advanced version. An example could be that some quality assurance methods can be disabled. This would be helpful because the user of the decision tool would not get unsuitable recommendations. This makes the decision process much easier for project managers in such situations.

   Such functionalities should be adjustable by the users because the business environment or the structure of companies can change and the tool has to be that flexible to can be adjusted according these changes.

3. Adjustment according to experience

   The third step of adjustment is the adaptation based on the experience of working with the tool is useful to develop it further. This is necessary because the area of software development is changing and evolving permanently. So the decision tool should be able to react on these changes to keep it up to date.

   The possibility of self adaptation with the help of experiences from users is very attractive because it would be a continuous improvement process independent from the advancements of the inventors of the tool. So the grade of flexibility would increase.

   A possible way of conduction of these adjustments could be that the advanced version of the tool can be synchronized with a database which is available over the internet. The advantage would be that every version which is in use provides information for this central database and the number of gained experience would be very high. A second advantage of this solution is that the distribution of the information happens automatically and therefore the tool does not need a manual update or service packs.

Furthermore qualitative attributes playing a very crucial role in this research area because they describe several environmental conditions of a project but they can not be compared like other parameters as for instance project duration or costs. During the practical work it turned out that the most suitable way to compare qualitative attributes is a direct matching in a binary way which means that only two possible conditions of this matching process exists – match or no match.

It is absolutely clear that it is a bit difficult to evaluate the quality of several alternatives when there are only two output conditions. But it is easier to handle this problem by comparing a greater number of qualitative attributes. In this case the chance of matching outputs increase and the user of the tool can orientate himself by the number of positive matches.

Here is a short example to illustrate this approach. Let us assume we have fifteen qualitative attributes with three to five different states in our decision model and there are five quality assurance methods at choice. The project manager inserts his parameters into the decision tool especially the qualitative parameters. Because of the fact that we have many attributes and five different methods the chance is very high to have a different number of matching parameters by the various methods. Therefore the project manager can choose the method with the most hits of the qualitative matching process.

This course of action implements that the qualitative and the quantitative part of matching are two separate parts which can not be compared directly because of two main arguments: First the matching process is different and second the scope is also different. In case of further research on this topic this is a major point for the development of new procedures to make the interpretation of the output easier for the user.

9 DISCUSSION AND CONCLUSION

This chapter shall provide an overview of the whole topic to link the theoretical and practical sections with the research questions. In the final chapter the conclusion of this approach is presented and linked with future possibilities of research in this area. It shall also combine the most important perceptions and results to build a bridge to the conclusion sections of this chapter. The last two sections of this chapter shall help to connect the core problem with the results and give an outlook to future areas of work in case of the extension of this research topic.

Let us go a bit deeper into detail and have a look on the research questions again to compare them with the cognitions of the practical parts in order to depict the result of the practical work in relation to the questions at the beginning of the workflow. The outcome is compared with the questions in the following three sections.

9.1 BASIC RELATIONS

Now let us remember the first research question about the basic relations between software development projects and quality assurance methods. The question was if there is any possibility of finding attributes which describe software development projects as well as quality assurance methods in order to have a possibility to describe both practices with the help of a common interface.

As we can see in chapters six and seven there are many attributes which can be used to develop such an interface with the objective to compare both practices to find the best fitting quality assurance methods for a given software development environment. These attributes must be split into two groups because the first group of attributes can be ranked and the second group can not. This problem is described further in subchapter 9.4.

With the help of the prototype and the practical example we can see that the realization of this questioning is possible. The decision routine works if we have an adjustable part which can be used to insert the environmental conditions of the project and a reference part which represents the basis for the matching process. Therefore the basic relation between software development projects and quality assurance methods can be constructed with the help of a decision tool which brings us to the following research questions.

9.2 REQUIREMENTS

The second research question deals with the requirements of such a decision process that should be automated. So what are the requirements for an experience based decision model for the selection of the best fitting quality assurance method? Software development projects are normally categorized with the help of several project characteristics. Furthermore we have theoretical and practical experience concerning the usage of different quality assurance methods. The trick is to bring all these attributes and characteristics into a scale which makes it possible to compare these values and bring them into a ranking afterwards.

But as mentioned before some attributes can not be brought into any scale because the choices of these attributes are not quantifiable. The prototype presents the following solution of this problem. The application domain of software projects are a very good example for a qualitative attribute. The only possibility of comparing this information of the user with the reference values of the tool is to look if the selection of the user can be found in a reference model or not.

Thus the matching of this qualitative attributes contributes nothing to the ranking process. This information must be seen as a bonus for the project manager which tells him that a special quality method fits to his selected application domain.

Last but not least we had to find a possibility to provide the user of the decision tool a proposal for a reasonable quality assurance strategy which consists of two or more QA methods that are used during different project stages. To cover this problem the prototype has two types of quality assurance methods implemented. First we have reviewing methods which are used for document approval in the first project stages and second we have testing methods which are needed in the last project stages to inspect the code for correctness. With the help of this separation a proposal for a suitable quality assurance strategy can be given to the user.

So we have defined the necessary requirements to develop a decision tool for the evaluation of the best quality assurance strategy.

9.3 DECISION CRITERIA

The last research question deals with the choice of the right decision criteria and the course of action of the prototype. As we have seen before several decision criteria can be found but how we are able to qualify the reference methods. The reference values of the prototype are adjusted with the help of several pieces of literature of QA methods which describe pros and cons between various quality assurance methods. The second origin of the reference values is a practical grading of the different methods. As described in chapter seven, a wider practical evaluation of reference values is necessary to guarantee the objectivity of the tool. This argument is described further in the previous subchapter.

Now the prototype has several attributes which can be used for the comparison and the reference values are adjusted to ensure a useful outcome of the tool. The last question is how the process of comparison and evaluation works. The prototype realizes this problem with the help of a so called decision vector. All attributes can be subsumed in a vector. This vector has always the same structure to ensure that the correct attributes are compared.

Now the decision vector is compared with all available reference vectors of the quality assurance methods. This is possible because the values of the attributes are scaled into a predefined range before. If an element of this the reference vector is different to the same element of the decision vector, the deviation is calculated and stored. After all deviations are determined the sum of deviation of all reference vectors is computed and goes directly into the ranking procedure. The lower the sum of deviation is the better is the matching between the decision vector and the reference vector.

The output of this calculation process is depicted directly on the input page of the user the give him an overview of the effects of changing an input parameter. The presentation of the different quality assurance strategies is done on a separate sheet which shows the strategies, the associated quality assurance methods and the single attributes by request. All these pieces have depicted the sum of deviation in order to guarantee a good overview for the project manager.

Last but not least the ranking of the different methods and strategies is realized with the help of a traffic light system. The best strategy is highlighted green whilst the others are highlighted yellow or red which depends on their rank.

With the help of answering the research questions we have created the requirements to develop a prototype of a decision tool for this research problem. The following chapter deal with the conclusion of this approach and an outlook on further research on this topic.

9.4 COMPARISON OF THE CORE PROBLEM WITH THE CONCLUSION OF THIS THESIS

After taking a look into the theoretical and practical topics of this research area it is necessary to qualify the outcome of the proposal for such a decision tool. As we can see in chapter seven and eight, the basic idea of matching attributes between software projects and quality assurance methods is very useful to find the best methods and strategies for a given software project environment.

The sensitive point of this problem is the proper qualification of the quality assurance methods with the help of all available qualification attributes. According to the fact that many QA methods have very similar characteristics this qualification is the deciding point between success and defeat of such a decision tool.

Some people of this business are of the opinion that this qualification and classification have to happen with the help of practical experience from already conducted software projects because in the software project business many things can not be planned theoretically. Therefore further research on this topic should be conducted in cooperation with some business partners in the area of software development.

Furthermore this decision tool can be very helpful for large companies which have experience in the conduction of several different quality assurance methods because a well adjusted tool can help them to maximize the quality of their output by spending an equal amount of time and money. It is at hand that this way of selecting and conducting quality assurance efforts have to be understood and lived by the management and the employees of software development companies because the decision tool is only the support to find effective quality strategies.

The proper execution of quality assurance and the way of dealing with and learning from failures depends on the corporate culture and the attitude of every employee of a company. This problem can be observed by many large companies in all business areas. The basic attitude of all employees must be changed to the point of view that a failure is not a mistake from an employee or a department; it is rather a chance to find problems in the company’s structure and to conduct preventive actions to avoid the problem in the future.

Therefore the implementation of this decision tool is only one step to open minded business thinking. It has to be combined with the proper conduction of the selected QA strategy and follow up discussions about the success and the effectiveness to ensure that the company is moving forward and the tool gets the right inputs for better adjustment to the needs of the company.

9.5 FURTHER RESEARCH TOPICS OF THE AREA

During the work on the prototype and the following evaluation phase some aspects appeared which certainly need further research to improve the effectiveness of the decision tool and make it applicable for business usage. The following points cover some of the steps to the operational readiness of the prototype. But it is obvious that the completion is far more difficult than it seems to be because many different stakeholders have to be involved and these points are just ideas which come up during the work on this solution approach. It is worth to mention these research topics to complete the cognitions which are made during this thesis.

- Enhancement of reference values

  To prove the functionality of the prototype twelve quantitative variables and one qualitative variable is used in the prototype. Further variables of both categories must be added to enhance the level of detail. But the focus should be laid on the qualitative attributes because they describe many environmental conditions of the project although they are difficult to compare.

  But we must not forget that a too detailed description of a project can also cause problems. First the acceptance of the tool can decrease if too many attributes are to define because this would lower the usability of the decision tool for less experienced users and for small projects.

One solution of this problem could be a selection of two modes of the tool – one for small projects and one for complex projects. The version for small projects has only the basic project attributes and adjustment functions whilst the advanced version covers the whole functionality of the tool.

Second some attributes can not be defined at the beginning of the project because some environmental factors are not clear at this time. Therefore the decision finding process possibly needs a couple of iterations to bring all necessary aspects into account. This can also be a part of an advanced decision tool.

- Implementation of further quality assurance methods

  The current version of the prototype consists of six different quality assurance methods. The next step in this field should be the implementation of all popular quality assurance methods which are used in recent projects. To model them properly information from various business partners is necessary. The more strategies are implemented in the decision tool the wider is the range of possible quality assurance strategies. Therefore the companies have more options to find the best solution for their environmental conditions.

- Detailed definition of quality assurance strategies and standardization

  The quality assurance methods are separated into two project phases to show the possibility of quality assurance strategy evaluation. The prototype points out all possible combinations between the methods of the two phases. The other two phases have to be implemented too and the possibility of switching methods from one phase to another is also important to gain a flexible decision tool.

  A further possibility would be the implementation of standard strategies which consist of harmonising QA methods. The advantage of this approach is that the outputs from the preceding method can be used as input for the following QA method. A similar structure can be observed by audit circles. All detected weaknesses of an audit are reported. They are used for the creation of improvement actions and these actions are checked and evaluated during the following audit. Such a circle could be useful for quality assurance strategies too. In order to implement this approach into QA strategies standards must be defined to be able to create interfaces between the different methods.

- Implementation of preferred strategies for specialized companies

  Another possibility of improvement would be the pre selection of preferred methods and strategies. This can be useful for smaller companies which does not have experience in all different quality assurance methods. So they can select only the methods they are familiar with and the tool can consider this during the calculations. This feature increases the usability of the tool for smaller companies which are specializing in only a few number of QA methods because the tool would not create strategies which are not applicable for these companies.

This feature can also be used for resource planning matters. Let us assume a software development company has some specialist employees for every different quality assurance method. If some departments have an overflow of work the decision tool can be adjusted to this situation with the help of this feature.

- Implementation of practical reference values from several companies

  It is at hand that a decision tool needs real experiences to adjust the references properly. This makes an informed choice possible. The most valuable information can be obtained by the companies which using quality assurance methods because the employees have practical experience in usage of them. They can evaluate the necessary attributes in an appropriate way.

  Therefore it is necessary to implement several practical weightings of reverence values into the prototype to ensure useful decision automation. To realise this some options are possible. One way could be the collection of practical values from different companies and an overall weighting according to the size of these companies. In this case the advantage is that additional references can be implemented easily.

  Another way is the reference calculation outside the tool. This has the disadvantage that the adjustment of the tool is more difficult and it would also lose usability. All in all we can say that practical experience is very important for a decision tool like this and it is absolutely necessary to implement practical knowledge to gain an efficient tool.

- Enhanced weighting of reference values

  The current prototype has implemented three different ranges for the matching process to prioritize some values in relation to the others. In chapter seven the application example showed that it is a philosophical question if the ranges should be expanded or not. The current setup has the outcome that the scoring of the strategies was very close together. This fact has the disadvantage that users could be of the opinion that the strategies are nearly have the same level of quality for the selected input parameters.

  Therefore the next logical step is the enhancement of the ranges. But it is very important evaluate the importance of the reference values and their interdependencies separately in order to have an effective improvement. If we have two parameters for example which affect the same project characteristics and there are no other interdependencies the rage of values must be the same to guarantee a correct calculation.

This is a very important point for the decision tool because as soon as the user is able to compare values at the end of the calculation process they have to look reasonable just to make sure that the authenticity of the tool is demonstrated for him. But that is not the only important point. The range of the reference values determines the maximal possible deviation for an attribute. Due to the fact that the deviation influences the outcome of the matching process directly with the help of this method of calculation the adjustments of these values must be done very carefully.

- Adjustable reference values according to the output of the tool and the experiences of the users

  Another useful enhancement of the tool could be that the users of the tool can insert a qualification of the last usage of the tool. This could happen like a feedback report which includes the input values and the satisfaction with the output. Due to the fact that this function is not implemented yet new methods must be developed to realise this feature. The idea of this enhancement is grown during the practical evaluation of the tool but the prototype does not provide the necessary requirements to implement this function.

  But this enhancement can be very useful regarding the fact that the tool would become "self learning". This is also a way to bring practical experiences into the decision process and helps to improve the tool continuously. And the goal of continuous improvement is necessary in this area of research as we discussed in previous sections to adopt the tool to the requirements of software development which are changing permanently.

  Another argument why this function is not implemented into the prototype is that the goal of the prototype was the general possibility of comparing software development characteristics with attributes of quality assurance methods. If this goal had failed the implementation of this function would be useless.

- Further research on qualitative attributes and their matching possibilities

  The research on matching possibilities of qualitative attributes is a quite big area. It is not easy to compare qualitative arguments because the can not be scaled to discrete values as discussed in the previous sections. The possibility of depicting qualitative attributes for fitting strategies is implemented in the prototype. But if there is a greater number of qualitative attributes which have to be compared, this solution could be a bit confusing. In this case one solution could be that the number of qualitative attributes which meet the requirements is depicted.

Another possibility could be a graphical solution like a diagram of matching attributes. But it is absolutely clear that this is not an easy topic. The importance of qualitative attributes depends also on the environmental conditions of the project and therefore the project manager is the last authority to qualify them. All things considered the matching of qualitative attributes is a very interesting but complex topic and it is worth to conduct further research on this topic.

If we have a look at all these points of improvement we are able to realize the wide area of research we are dealing with. It is absolutely clear that the market maturity of this decision tool needs much more additional work and inputs from various stakeholders to ensure the coverage of all necessary aspects.

Because of the fact that we are dealing with a decision support tool the importance of practical experience data from companies must not be forgotten.

The conclusion of this first approach is that such a decision support is needed in the quality assurance area of software development projects. Furthermore it turned out to be a very powerful tool with high potential if the way of this approach will be continued and improved. Therefore the basic approach which underlies this thesis is very good and it is worth to conduct further research on this topic.

With the help of this tool and further research, standardization efforts in the software development area can be extended to the closely linked field of quality assurance and its strategies. It would help to distribute experience of software development companies and to improve knowledge of effective usage of quality strategies in order to improve the quality level of software products in general.

10 BIBLIOGRAPHY

[1] Abdelnabi, Z., Cantone, G., Ciolkowski, M. and Rombach, D. "Comparing code reading techniques applied to object-oriented software frameworks with regard to effectiveness and defect detection rate" *International Symposium on Empirical Software Engineering, ISESE '04. Proceedings*, 2004, pp. 239 - 248.

[2] Advanced Development Methods, I. "Controlled Chaos: Living on the Edge" *White Paper*, 1996, pp. 10.

[3] Anderson, P., Reps, T. and Teitelbaum, T. "Design and implementation of a fine-grained software inspection tool" *IEEE Transactions on Software Engineering* (29), 2003, pp. 721 - 733.

[4] Andersson, C., Thelin, T., Runeson, P. and Dzamashvili, N. "An experimental evaluation of inspection and testing for detection of design faults" *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings*, 2003, pp. 174 - 184.

[5] Baah, G. K., Gray, A. and Harrold, M. J. "On-line anomaly detection of deployed software: a statistical machine learning approach" *Proceedings of the 3rd international workshop on Software quality assurance*, 2006, pp. 70 - 77.

[6] Bahsoon, R. and Emmerich, W. "Evaluating Software Architectures: Development, Stability, and Evolution" *Proceedings of ACS/IEEE Int. Conf. on Computer Systems and Applications, Tunis, Tunisia*, 2003, pp. 10.

[7] Bartolomei, T. T., Garcia, A., Sant'Anna, C. and Figueiredo, E. "Towards a unified coupling framework for measuring aspect-oriented programs" *Proceedings of the 3rd international workshop on Software quality assurance*, 2006, pp. 46 - 53.

[8] Basili, V. and Caldiera, G. "The Experience Factory" *Encyclopedia of Software Engineering* (2), 1994, pp. 469-476.

[9] Biffl, S. "Software Inspection Techniques to support Project and Quality Management", 2001.

[10] Biffl, S. "Lesetechniken für die Inspektion von Software-Anforderungsdokumenten" *Informatik - Forschung und Entwicklung* (16), 2001, pp. 145-158.

[11] Biffl, S., Babar, M. A. and Winkler, D. "Impact of Experience and Team Size on the Quality of Scenarios for Architecture Evaluation," *Conference on Evaluation and Assessment in Software Engineering (EASE), Bari, Italy*, 2008, pp. 10.

[12] Biffl, S., Denger, C., Elberzhager, F. and Winkler, D. "A Quality Assurance Strategy Tradeoff Analysis Method" *Euromicro Software Engineering and Advanced Applications (SEAA)*, 2007.

[13]Brykczynski, B. "A survey of software inspection checklists" *ACM SIGSOFT Software Engineering Notes* (24), 1999, pp. 82.

[14] Chaar, J., Halliday, M., Bhandari, I. and Cihillarege, R. "In-process evaluation for software inspection and test" *Software Engineering, IEEE Transactions on* (19), 1993, pp. 1055 - 1070.

[15] Colquitt, D. and Leaney, J. "Expanding the View on Complexity within the Architecture Trade-off Analysis Method" *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, 2007, pp. 45 - 54.

[16] Cooper, D., von Konsky, B., Robey, M. and McMeekin, D. "Obstacles to Comprehension in Usage Based Reading" *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, 2007, pp. 233 - 244.

[17] Denger, C. and Elberzhager, F. "Basic Concepts to Define a Customized Quality Assurance Strategy" *IESE-Report*, 2007, pp. 42 pp.

[18] DiFatta, G., Leue, S. and Stegantova, E. "Discriminative pattern mining in software fault detection" *Proceedings of the 3rd international workshop on Software quality assurance*, 2006, pp. 62 - 69.

[19] Doyle, J. and Thomason, R. H. "Background to Qualitative Decision Theory" *AI Magazine* (20), 1999, pp. 35.

[20] Freimut, B., Klein, B., Laitenberger, O. and Ruhe, G. "Measurable Software Quality Improvement through Innovative Software Inspection Technologies at Allianz Life Assurance", 2003.

[21] Gilb, T., Graham, D. and Finzi, S., (eds.) *Software Inspection*, Addison - Wesley, 1993.

[22] Hu, P., Zhang, Z., Chan, W. K. and Tse, T. H. "An empirical comparison between direct and indirect test result checking approaches" *Proceedings of the 3rd international workshop on Software quality assurance*, 2006, pp. 6 - 13.

[23] ISO/IEC 12207 *Information Technology - Software Life Cycle Processes*, First Edition ISO/IEC, 1995.

[24] Kazman, R., Barbacci, M., Klein, M., Carrih-e, S. J. and Woods, S. G. "Experience with performing architecture tradeoff analysis" *Proceedings of the 1999 International Conference on Software Engineering*, 1999, pp. 54 - 63.

[25] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H. and Carriere, J. "The Architecture Tradeoff Analysis Method" *Fourth IEEE International Conference on Engineering of Complex Computer Systems, ICECCS '98. Proceedings.*, 1998, pp. 22.

[26] Kazman, R., Klein, M. and Clements, P. "ATAM: Method for Architecture Evaluation"

(CMU/SEI-2000-TR-004ESC-TR-2000-004), 2000, pp. 83.

[27] Kelsey, R. B. "Integrating a defect typology with containment metrics" *ACM SIGSOFT Software Engineering Notes* (22), 1997, pp. 64 - 67.

[28] Kirchler, E. and Schrott, A. *Entscheidungen in Organisationen*, WUV, 2003.

[29] Krebs, J. "IBM Rational software - The Rational Edge", 2005, pp. 5.

[30] Kruchten, P. "How the Rational Unified Process Supports ISO 12207", 2002, pp. 7.

[31] Kruchten, P., The rational unified process, Addison-Wesley, 1999.

[32] Laitenberger, O. "A Survey of Software Inspection Technologies", 2001.

[33] Laitenberger, O. and DeBaud, J. "An encompassing life cycle centric survey of software inspection" *The Journal of Systems and Software* (50:1), 2000, pp. 5-31.

[34] Laux, H. *Entscheidungstheorie*, Vol. 3. Auflage, Springer, 1995.

[35] Macdonald, F. and Miller, J. "A Comparison of Tool-Based and Paper-Based Software Inspection" *Empirical Software Engineering* (3), 1998, pp. 233-253.

[36] Mashayekhi, V., Drake, J., Tsai, W. and Riedl, J. "Distributed, collaborative software inspection" *Software, IEEE* (10), 1993, pp. 66 - 75.

[37] Pareto, L. and Boquist, U. "A quality model for design documentation in model-centric projects" *Proceedings of the 3rd international workshop on Software quality assurance*, 2006, pp. 30 - 37.

[38] Parnas, D. L. and Lawford, M. "The Role of Inspection in Software Quality Assurance" *IEEE Transactions on Software Engineering* (29:8), 2003, pp. 674 - 676.

[39] Rosenblum, D. S. "Formal methods and testing: why the state-of-the art is not the state-of-the practice" *ACM SIGSOFT Software Engineering Notes* (21), 1996, pp. 64 - 66.

[40] Services, P. E. "PASA", Web. http://www.perfeng.com/pasa.htm

[41] Singh, R. "An introduction to International Standard ISO/IEC 12207", Technical report, Federal Aviation Administration, 1999.

[42] Singh, R. "International Standard ISO/IEC 12207 Software Life Cycle Processes" *IEEE* (), 1996, pp. 18.

[43] Stobie, K. "Too darned big to test" *Queue* (3), 2005, pp. 30 - 37.

[44] Tan, S. W. and Pearl, J. "Qualitative Decision Theory" *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994, pp. 928 - 933.

[45] Tanaka, T., Aizawa, M., Ogasawara, H. and Yamada, A. "Software quality analysis and measurement service activity in the company" *Proceedings of the 20th international conference on Software engineering*, 1998, pp. 426 - 429.

[46] Thelin, T., Andersson, C., Runeson, P. and Dzamashvili-Fogelström, N. "A replicated experiment of usage-based and checklist-based reading"*, Proceedings. 10th International Symposium on Software Metrics*, 2004, pp. 246 - 256.

[47] Thelin, T., Runeson, P. and Wohlin, C. "An experimental comparison of usage-based and checklist-based reading" *Software Engineering, IEEE Transactions on* (), 2003, pp. 687 - 704.

[48] Thelin, T., Runeson, P., Wohlin, C., Olsson, T. and Andersson, C. "How much information is needed for usage-based reading? A series of experiments"*, Proceedings. 2002 International Symposium on Empirical Software Engineering*, 2002, pp. 127- 138.

[49] Wagner, S. and Meisinger, M. "Integrating a model of analytical quality assurance into the V-Modell XT" *Proceedings of the 3rd international workshop on Software quality*

assurance (), 2006, pp. 38 - 45.

[50] Wheeler, S. and Duggins, S. "Improving software quality" *Proceedings of the 36th annual Southeast regional conference* (), 1998, pp. 300 - 309.

[51] Winkler, D., Denger, C., Elberzhager, F. and Biffl, S. "QATAM: ein Szenario-basierter Ansatz zur Evaluierung von Qualitätssicherungsstrategien," *Presentation*, 2008, pp. 19.

[52] Winkler, D., Denger, C., Elberzhager, F. and Biffl, S. "Quality Assurance Tradeof Analysis Method (QATAM) - An Empirical Quality Assurance Planning and Evaluation Framework", Technical report, Technische Universität Wien, 2007.

[53] Winkler, D., Riedl, B. and Biffl, S. "Improvement of design specifications with inspection and testing"*, 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, 2005, pp. 222 - 230.

[54] Winkler, D., Thurnher, B. and Biffl, S. "Early Software Product Improvement with Sequential Inspection Sessions: An Empirical Investigation of Inspector Capability and Learning Effects"*, 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, 2007, pp. 245 - 254.

[55] *V-Modell XT*, 2006. http://v-modell.iabg.de/index.php?option=com_docman&task=doc_details&gid=39&Itemid=30

[56] www.agilealliance.org "Agile Processes - Emergence of Essential Systems" *White Paper*, 2001, pp. 3.

[57] www.agilealliance.org "Agile Processes and Self-Organization" *White Paper*, 2001, pp. 3.

[58] Barbacci, M., Longstaff, T. H., Klein, M. H. and Weinstock, C. B. "Quality Attributes", Technical report, CMU/SEI-95-TR-021, ESC-TR-95-021, 2005.

[59] http://en.wikipedia.org/wiki/Software_Lifecycle_Processes

[60] http://en.wikipedia.org/wiki/Decision_theory

[61] http://www.v-modell-xt.de/

[62] http://www.v-modell.iabg.de/

[63] https://www.uni-hohenheim.de/i410a/skriptal/etheorie.html (Online)

[64] http://www.rsf.uni-greifswald.de/bwl/gesundheit/entscheidungstheorie.htm

[65] http://www.sei.cmu.edu/architecture/ata_method.html

[66] Winkler, D. Software Engineering and Project Management, *Presentation*, 2004

[67] http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process

[68] Myers, G. *The Art of Software Testing*, 1979

[69] http://en.wikipedia.org/wiki/Scrum_(development)

[70] Beck, K. Test – Driven Development by Example, Addison Wesley, 2003

11 LIST OF FIGURES