Alexander Kalchauer

Monitoring of software process KPIs using a web-enabled dashboard

Master's Thesis

Graz University of Technology

Institute for Softwaretechnology
Head: Slany, Wolfgang, Univ.-Prof. Dipl.-Ing. Dr.techn.

Supervisor: Wotawa, Franz, Univ.-Prof. Dipl.-Ing. Dr.techn.

Graz, May 2013

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ____________________ ____________________

Date Signature

Eidesstattliche Erklaerung[1]

Ich erklaere an Eides statt, dass ich die vorliegende Arbeit selbststaendig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen woertlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ____________________ ____________________

Datum Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Contents

List of Figures

Abstract

Quality management is a very important aspect in the development of software applications nowadays. One part of the quality management is the measurement of quality attributes in software applications and processes, that influence the development of software. The most important measurable numbers in a company are key performance indicators (KPI). Tools that can measure and monitor those KPIs and present them to the user are called Software Dashboards or Software Cockpits.

This master thesis gives an overview, how to measure an issue tracking process in a web enabled Software Cockpit. The issue tracking process is part of the application life-cycle management is software companies. It is described, in which way the chosen tracking process interacts with other processes in the life-cycle of a software application. The measurement of the quality of this process is done by using quality models. Those models show how to break down abstract quality terms to metrics. Metrics are rules how to measure one attribute of a software product or process. The metrics are implemented in Software Cockpits, that use the data of this process with the rules how to measure the process, to monitor it. Several Software Cockpits and other monitoring tools are compared to give an overview over the current market in this area.

In the practial part an existing Software Cockpit is introduced and it is explained how to change the architecture of this cockpit, to implement several new metrics. The metrics and the model of the issue process are given by a partner company in the field of the automotive software producing industry. The queries, that implement the metrics are splinted into several sub-groups and described. Finally a case study about the issue tracking process of the partner company is done. The result of this case study shows the behavior of the issue tracking process of the partner company. This study presents the practical business value of the implemented Software Cockpit.

Kurzzusammenfassung

Qualitätsmanagement ist ein wichtiger Bestandteil in der heutigen Entwicklung von Software Applikationen. Ein Aspekt des Qualitätsmanagement von Software Applikation und der beteiligten Software-Entwicklungsprozessen ist die Messung deren Qualitätsattribute. Die wichtigsten messbaren Einheiten sind eines Unternehmens sind seine Leistungskennzahlen. Tools die diese Zahlen messen und überwachen können werden Software Dashboards oder Software Cockpits genannt.
Diese Masterarbeit zeigt auf wie ein Issue-Tracking Prozess mittels eines Software Cockpits überwacht werden kann. Ein Issue Prozess ist die des Application Lifecycle Management. Es werden in dieser Arbeit die Zusammenhänge der verschiedenen Prozesse innerhalb des ALM beschrieben. Qualitätsmodelle werden im Qualitätsmanagement verwendet. Diese Modelle beschreiben, wie man abstrakte Qualitätsmerkmale auf sogenannte Metriken projizieren kann. Metriken sind Regeln, die definieren wie man einzelne Merkmale eines Softwareprodukt oder Softwareprozesses messen kann. Metriken werden in den Software Cockpits, die ebenso die Daten über den Prozess oder das Produkt enthalten, implementiert, um das Produkt oder den Prozess zu überwachen. Nach dieser Einführung werden mehrere Software Cockpits miteinander verglichen, um einen Marktüberblick zu geben.
Im praktischen Teil wird ein bereits existierendes Software Cockpit beschrieben. Es wird gezeigt wie die Architektur dieses Software Cockpits verändert werden kann, um neue vorgegebene Metriken zu implementieren. Die Querys, die diese Metriken implementieren werden in Untergruppen eingeteilt und beschrieben. Abschließend werden noch die Ergebnisse einer Fallstudie über das Verhalten eines Issue-Tracking Prozesses präsentiert, die in Zusammenarbeit mit einer Partnerfirma aus der Automotive Softwareentwicklungsbranche durchgeführt wurde. Die Fallstudie zeigt den praktischen Wert des Software Cockpits auf.

1 Introduction

This master thesis describes the monitoring of software process KPIs using a web-enabled dashboard. Section 1.1 of this introduction gives a short motivation example about the importance of quality measurement and measurement of software. ISO and industry quality models are introduced. Section 1.2 introduces the Software Dashboard and shows how Software Dashboards can be used to measure the quality of a business product. Section 1.3 describes the goals of this thesis.

1.1 Software quality measurement

Software is a important part in every person's life nowadays and the influence of software is increasing each year more and more. Complex software components are integrated in every car or building. Every person has contact to software every day, by using a smart phone or a mobile computer. The software industry increases and new software is implemented very fast. Nevertheless software failures in the past years created the awareness that the focus of creating software is not only on the fast production of new software, but is also on producing high quality software. The top ten examples of failures in 2011 of software presented by Phil Codd show how much software failures still influence our life and our economy and that software failures are not a problem of the past (*Business Computing World @ONLINE* 2013).

Quality standards like ISO 9001 or industry models like the Capability Maturity Model Integrated (CMMI) and their software oriented sub-categories the standard ISO 9126 and CMMI-DEV are state of the art models that ensure the quality in the production of software applications. Those models define quality dimensions of software applications as well as the processes involved in the production of the software applications and how to improve them. Due to the high complexity and high abstraction level of those models and standards, using one or more quality models is a huge challenge for companies (Mutafelija and Stromberg, 2003). Nevertheless to usage of such quality standards and models increased more and more in the last years . Those quality models are often part of business contracts between companies or the government. More than 1 million companies are ISO 9001 certificated (*ISO 9000 @ONLINE* 2013).

One essential part of those quality models is the measurement of the quality attributes and the measurement of the business processes connected to the production and usage of a software product. This measurement is used by the management to control the production and operation of their software products and processes and take strategic decisions. Due to the fact that the different management levels need different sights on the measurements of software products or processes and the giant amount of data available in companies, tools are needed that can collect, store this data and present this data on different abstraction levels and sights to the management. The storage of the available data and the measurement of all dimensions of a software including the important quality attributes are part of Business Intelligence solutions of software companies (Negash, 2004). The data is stored in data warehouses and presented to the user in so called Software Dashboards.

1.2 Dashboard

Originally a dashboard represents the measurement instruments in the cockpit of a car. It shows the driver of a car the current status of its car, like the current speed of the amount or the fuel in the reservoir. This term has been adopted for the software industry. In the software industry dashboards are the state of the art way to measure a software product or a associated business processes. Software Dashboards can also be called Software Cockpits. Dashboards that measure data of a business processes are also called business dashboards or enterprise dashboards. A Software Cockpit runs queries on a data storage, usually a data warehouse and presents the result with diagrams or reports to the user.

The data warehouse is the data storage system of a Software Dashboard. It collects data of the observed processes or software products and all attributes of the products and processes. The data and meta-data can be from any kind of resource. Usually the resources are software versioning systems like SVN, ticketing systems like issue tracker or social systems. The data warehouse is a helper of the Software Dashboard. It stores the data in a relational database by using the star- or snowflake schema pattern. On top of the data warehouse the data is structured in multidimensional cubes by using the Online Analytical Processing(OLAP) technology. OLAP cubes consist of dimensions hierarchies and measures. The measures are the information about the product or the investigated process in numbers. The dimensions represent the kind of attribute of a specific measure. Hierarchies are abstraction levels of the dimensions. The data in the cube is accessed with multidimensional queries. Queries use the three elements of the cube to give the user the possibility to combine different dimensions and investigate the data on different abstraction levels by using the hierarchies (Chaudhuri and Dayal, 1997). Modern dashboards give the possibility to drag and drop through the dimensions in the cube and through hierarchies. Often Software Dashboards are web-enabled and can be accessed with smart-phones or tablets.

This master thesis shows a way of implementing the business process: issue tracking into a Software Dashboard. An issue tracker tracks all bug and enhancement tasks of one or more software projects. The issue process is defined by its process model, that defines the states and the state changes of the process. The model has an entry state, several exit states and middle states. The states are connected by state change transitions. This graph is called a state-flow graph. The issues are imported from an issue tracking tool. An issue is represented in an issue tracker by one issue ticket. Every issue has a set of attributes like the creation time, the product or person it is connected to and the state in which the issue is at the current time. Each attribute of an issue is one dimension in the data warehouse. Beside of the issue and the attributes that describe the issue process, history based change events are imported into the Software Dashboard, that change those attributes. A change event is defined by the attribute it changes, it is also called activity of this change event, the value of the changed attribute and the date when the change happened. The most common change events are the state change events, but also change events of other attributes are stored. Queries use the process model, the change events, the issues and their attributes to measure the issue tracking process. The queries are implemented according to predefined metrics. Metrics are rules to measure of a property of a software.

1.3 Goals

The main goal of this master thesis is the adoption of an existing Software Cockpit that monitors an issue tracking process. With this extension it will be possible to monitor more complex quality dimensions of this issue process. Several parts in the architecture of the Software Cockpit have to be changed to extend the monitoring of the process by new dimensions and give the possibility to run more complex queries. The Software Cockpit monitors the issue tracking process of an automotive software producing company by using a set of metrics and event data given. The Software Cockpit imports a data-set of nearly 41000 issues and 250000 change events. The Software Cockpit has around 200 queries, they are implemented according to a set of metrics. After the extension of the Software Cockpit a case study is done with the available data.

Three main parts of the Software Cockpit architecture have to be changed, to add new dimensions into the Software Cockpit and add the possibility to create more complex queries:

- The data importation process has to import new history events of dimensions that should be added to the Software Cockpit. For example the clarification dimension has to be imported. The importation

process has to import additionally all history events with a clarification activity. The existing Software Cockpit only imports state change activities. A side goal is to refine the importation for a more consistent importation and the importation of old data. For example the data of person that changed their name is not imported yet.
- The structure of the relational database in the data warehouse has to be extended by the new dimensions. The relational database uses the database star schema pattern. Every dimension is one branch in the star schema. For the new dimensions branches in the star scheme model have to be added.
- The structure of the cube in the data warehouse has to be changed. New dimensions and measures have to be implemented. The structure has to be changed to support queries that investigate issues over several state changes. Therefor state change events of one issue have to be connected.

New queries have to be implemented. Queries are the way to access the data in the data warehouse and structure this data for the end user. The task is to create a framework of queries and templates of different groups of queries. The technical issue process department has to have the possibility to create specific queries for the management level and has to be able to change queries and create new queries fast. The queries in the existing Software Cockpit are able to count single status change activities, connect them to different dimensions like the product of an issue or its milestone and present measures like minimum, maximum or average transition time to the user. The goal is to create new classes of queries that fulfill the following requirements:

- A new group of queries implements the tracking of the change of the state of issues over more than one state change. The independent state change events of one issue have to be connected to each other. Queries have to recognize this connection by using existing query functions. This set of queries does also require additional changes in the architecture of the Software Cockpit.
- A new category are snapshot view queries. The current queries only measure the state change events that happened in an issue. Snapshot queries present the number of issues in a specific state at a given time-

point, instead of the changes done to any issue splinted to the dates when they happened. The focus of these queries is on the state and not on the issues and on the monitoring at a given time-point rather than measures that measure over a certain time-period. Mathematical a snapshot query can be described as the number of issues that entered a certain state minus the amount of issues that left an state at one time. The goal of the snapshot queries is to identify how the process envelops in one state, by observing a state at several time-points.

- One task is to implement new queries that investigate changes of the new dimensions that have been added to the Software Cockpit. The three new dimensions are the *test-failed*, *review-failed* and *clarification* dimension.
- Parametrization of the queries has to be implemented. The management wants to access each query with one HTML link and wants to access different dimensions for example the product or milestone dimension and different levels of one dimension. A example of different levels of a dimension is the higher level of the product dimension a complete product or the view on the level of a component of one product. Without the parametrization the management has to drill manually through the different dimensions and hierarchies. Therefor all queries in all groups for all dimensions have to be parametrized.

After extending the Software Cockpit and implementing the queries a small case study is done with the process model of the issue tracking process and the event data given by the company. The goal is to evaluate the Software Cockpit and interpret the informative value of the results of the queries. The results of the queries are compared to the given issue process model. The model is checked for its conformance with the model and the results of the queries are checked for their correctness and informative value.

1.4 Content of this thesis

The goal of the next two chapter of this thesis is to give the practical work done a theoretical foundation. In the next Chapter 2 it is explained how the issue tracking process is embedded into the application life-cycle management (ALM) of a company. Chapter 3 introduces the way of measuring software. Metrics and Key Performance Indicators (KPIs) are explained. The two quality meta models Goal-Question-Metrics and Factor-Criteria-Metrics show how to use metrics. Chapter 4 is a small market study about available software monitoring tools. Different aspects of the tools are inspected. Chapter 5 and describe the architecture of the implemented Software Cockpit and the queries that were implemented and added to this Software Cockpit. The results of the case study about the issue process is presented in Chapter 7. Chapter 8 presents related work done in the field of this master thesis, while Chapter 9 gives a conclusion and shows how the Software Cockpit can be improved in the future.

2 Application Life-cycle Management

Application Life-cycle Management (ALM) is a very common phrase in the information technology (IT) nowadays. Nevertheless there is no general definition, the general idea is that ALM expands the generic idea of life-cycle management to the production of software applications in the IT industry. The center of the ALM is the software development life-cycle (SDLC). SDLC includes all processes that are directly connected to the production of software. ALM furthermore expands the SDLC and includes all tools and processes in the entire time where an organization spends money on this asset, from the initial idea to the end of the applications life (Chappell et al., 2008). Since the idea of ALM is not defined generally every application has its own sight on application life-cycle parts and its common processes and patterns. ALM consists of processes from the business side as well as from the engineering side. It includes all tools and patterns to fulfill the goal of handling the application from its beginning to its end. This chapter describes ALM process models and the part of issue management. Afterward ALM tools are presented.

2.1 ALM process models

Several process models describe the processes and phases in ALM. The circle model and linear model describe all parts of the ALM. The focus of the traditional sequential model and the modern agile model is the SDLC. The first two models and the SDLC models can be combined with each other. For example can both SDLC models be part of the development phase of the linear model and describe this phase more detailed. Afterward the task or issue process, as one important process in ALM is described.

2.1.1 Circle ALM model

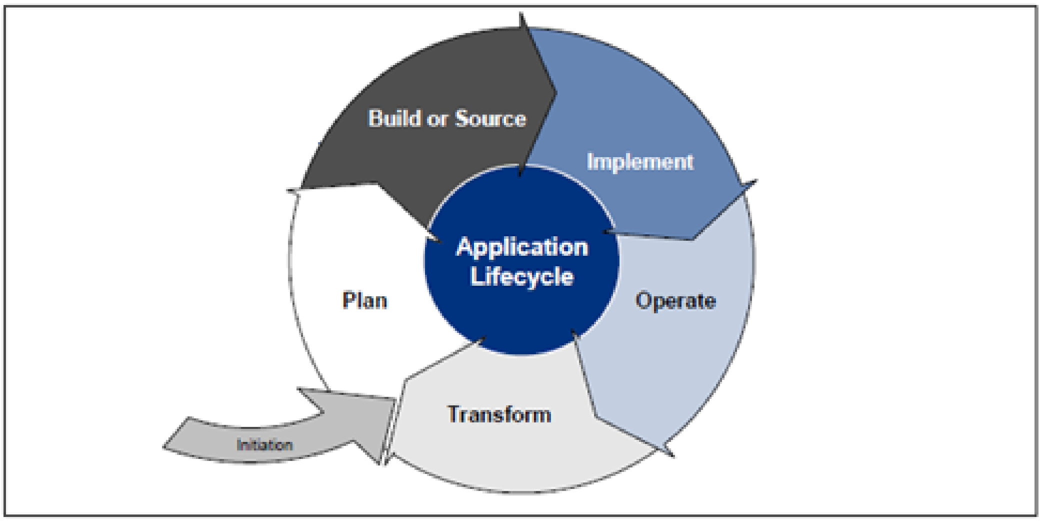


Figure 2.1: ALM Life-cycle (*http://innovationcenter.deteconusa.com/ @ONLINE* 2013)

One possible view on the ALM processes is the circle view (Rossberg, 2009). In this view the processes are splinted into 5 parts. The circle with the 5 parts is shown in the Figure 2.1. This circle model has an entry point at the planning phase where the idea is generated and an exit point after an operations phase.

The first part of the ALM life-cycle circle is the planning part, it includes the Requirement Management process and the concertizing of the initial idea. Often first market studies are done in this part of the circle and business plans are created. External consultants can be used to evaluate the plan in this stage to avoid early failures in projects.

In the build or analysis part the application is designed and the architecture of the application is created. Technical decisions are taken at this stage, for example which type of SDLC is chosen in this project and which platforms will be used.

The implementation part consists of the implementation, the customizing and the testing. This part of the ALM contains the main parts of the SDLC. Several SDLC runs are done according to version and milestone decisions done in the previous parts.

In the operation part the application is already implemented, but it can still need maintenance, bug fixing or a redesign. This part has a important business aspect, the application is now an asset of the company and asset management needs to be done.

In the transformation phase the application can get into retirement or a new application idea can trigger a restart of the circle.

2.1.2 Linear ALM model

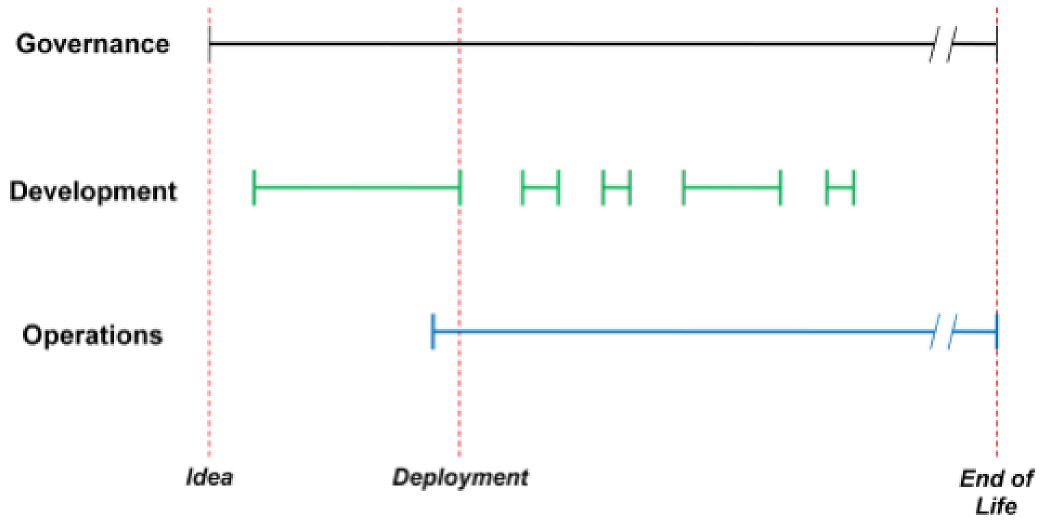


Figure 2.2: ALM can be viewed as having three aspects (Chappell et al., 2008)

Chappell has a more straight view on ALM. He speaks of three overlapping linear phases: the governance, the development and the operation phase. Figure 2.2 shows this linear model.

The governance phase controls the business aspects of the ALM. At the beginning the business model and business plans are created. Later on, it ensures that the business needs are fulfilled and the business plan is followed. From the beginning until the application is deployed, the application is in the project portfolio of a company. When the application is deployed it moves to the application portfolio management and is handled as asset of the company. Governance is the only phase that consists from the initial idea until the end of the lifetime without a break.

The development phase consists of several SDLC turns. The turns are defined through different versions or milestones of application. If there is more than one SDLC, it is possible that between those SDLC phases maintenance steps happen. In Figure 2.2 the long green lines are SDLC runs and the short green lines show maintenance phases. The development phase starts when the initial idea is already concertized and ends when the application is not extended anymore.

The operation phase starts for the first time just before the application is deployed. It handles monitoring and management aspects of the project. The operation phase is closely connected to the development phase, because every new SDLC turn and maintenance turn triggers a new operation event. Nevertheless after the application is deployed for the first time the operation phase never ends until the end of the applications lifetime (Chappell et al., 2008).

This model states in a good way the importance of both the development oriented processes, especially the SDLC processes, and the processes of the business side and their interaction.

2.1.3 Sequential SDLC model

Sequential models are also called linear or step-wise models. They consist of several following phases. Every phase can only begin, if the previous is completed. The different phases should not overlap. In adapted models it is possible to redo steps and jump to earlier phases of the model. Those models are not complete linear models (Royce, 1970).

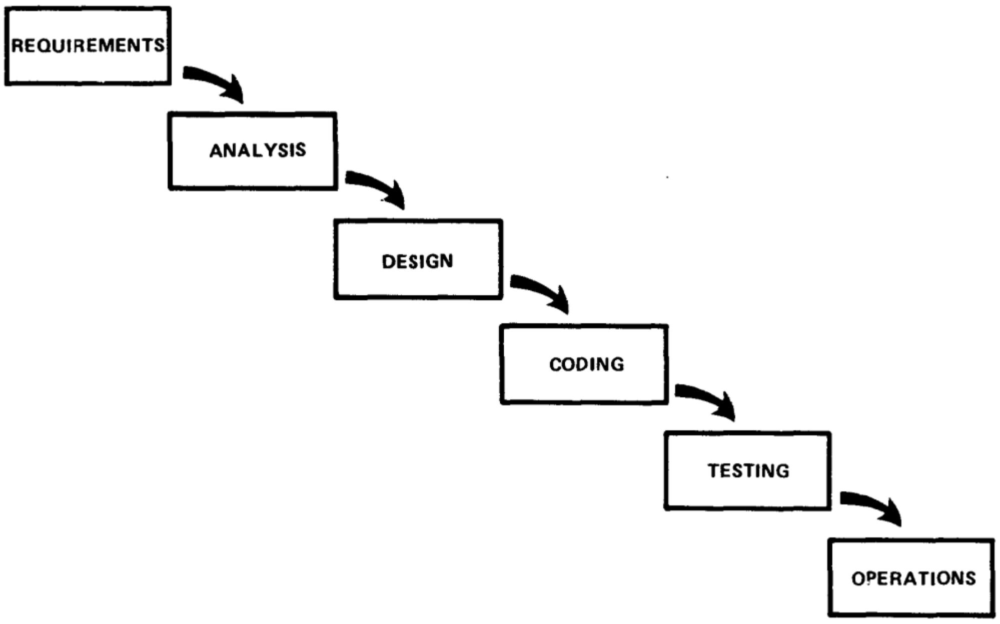


Figure 2.3: Advanced Waterfall Model (Royce, 1970)

The earliest and one of the most common and well defined sequential models is the waterfall model. There exist many waterfall models with steps on different abstractions levels and advanced models where it is possible to jump steps back and redo steps. Figure 2.3 shows a waterfall model that consists of the six following steps:

- At the beginning the requirements of the application are defined and documented.
- Analysis defines the general requirements more detailed, including market analysis and technical aspects. At the end of the analysis the requirements have to be exactly fixed for the design.

- Design structures the fixed requirements. Often model languages like Unified Modeling Language are used to create a technical design.
- Coding translates the design into a computer readable and executable language.
- In testing phase the created program is tested for its requirements.
- Operation phase handles the execution and the maintenance of the program (Royce, 1970).

One of the biggest disadvantages of this model is its lack of flexibility. The waterfall model requires fully elaborated documents as completion criteria for early requirements and design. In modern projects many requirements are often not documented or in many cases not known. In the early parts of a project. In projects where the flexibility is not needed, because the requirements are well defined and change seldom, sequential models are still very popular. One example of projects that use often sequential models are embedded systems.

Sequential SDLC models are easy to integrate in the other general models since one SDLC turn is one completed process and can be treated as one entity with a defined input and output.

2.1.4 Agile SDLC model

The less flexibility of the sequential SDLC models lead to the creation of more flexible models. Agile development, tries to solve the problem of fast changing requirements, by a new model of iterative and incremental development. It is based on four principles:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan (*agilemanifesto @ONLINE* 2013)

Although agile development is originally a SDLC model, all four principles have impact on the other ALM parts in both ALM models.

The importance of individuals and interactions requires flexible tools and processes. All phases in the ALM have to be changed if necessary. A continuous working software requires a flexible build management with tools that offer software versioning, continuous integration and nightly builds. The goal is that there exists always a running software application, which is enhanced step by step. Therefor only small changes and small iterations are done. In Scrum, as a agile example, there exist daily, weekly and monthly iterations. Each iteration contains small changes and enhancements done to a software application. This influences the implementation as well as the operation phase of ALM. Costumer collaboration requires a close interaction with the customer. Social tools make the interaction between developer and customer easier. The customer needs also access to the ticketing systems and has to be always up to date. At every time the customer can change the requirements. The customer is integrated in all ALM phases. Also responding to change, requires that processes and tools can be changed at any time.

Due to the fact that agile development influences ALM a lot, the definitions agile-ALM and agile-ALM tools were created (Goth, 2009).

2.1.5 Issue management processes

This master thesis focuses on one specific process of the ALM: the issue management process. Issue management is also know as task management in ALM. The issue process can be divided into two different processes the bug tracking process and the enhancement tracking process. Both processes are part of ALM in the different models. This section describes these processes and associates the processes with the previous introduced models. The bug tracking is also known as defect management, the enhancement tracking as technical task management. Both processes influence and are influenced by the quality management, the build management and the requirement management.

Bug tracking process

The bug tracking process handles the know faults, failures and errors in a software application, from the first occurrence until they are solved. An error is a human action that produces an incorrect result. A fault is the condition that causes the software to fail. The failure is the termination of the ability of a product to perform a required function or its inability to perform within previously specified limits. A failure may occur due to one or more faults in a software application. ("IEEE Standard Classification for Software Anomalies" 2010) Faults, failures and errors are named bugs and are treated same in the bug tracking process. This way bugs can depend on each other.

Each bug in a software is documented as a bug ticket. Bug tickets can be physical papers or virtual in a bug tracking system. A bug ticket contains relevant information and meta data about the bug. The information is for example: what happened, under which conditions did the bug happen or how can the bug be reproduced. The meta information is: who identified the bug or when did the bug happen. The ticket is generated by a developer or software application support. Bug tickets are the input of the bug tracking process. In the bug tracking process the bug is analyzed, assigned to a developer and solved by a developer. The output of the bug tracking process is that the bug ticket is marked as solved. It is also possible that the bug is rejected, for example when the bug is already solved or is not solvable. Chapter 7 describes a bug tracking process and compares two different bug tracking processes.

Bugs are recognized and furthermore bug tickets are created in two different phases of the ALM. If the test of a software fails, the bug is recognized in the testing phase. In the circle model the testing is part of the implementation phase. In the linear model it is part of the development phase. This is often controlled by the test management.
If a bug is recognized by a user of the software, the bug ticket is created in the operation phase of the ALM.
Bugs are solved in maintenance steps or in SDLC phases of a new version

of the software. In the circle model the maintenance is part of the operation phase, in the linear model maintenance steps happen between the different SDLC turns.

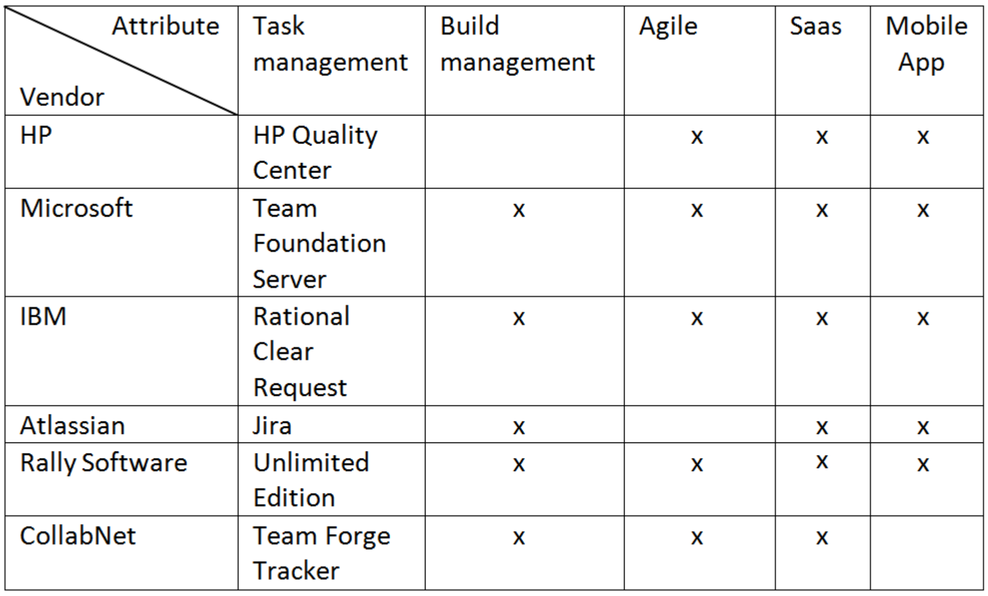
Enhancement tracking process

The enhancement process is closely connected to the bug tracking process. The enhancement process also stores tickets with relevant information about the enhancement and its meta data. Often the same tool is used in both processes to store the bug and enhancement tickets. The difference between those two types of processes is the creation of the ticket. The enhancement tickets are created regarding to the requirements of a software application. The enhancement tickets describe new features of a software application or other tasks of the developer. In sequential SDLC all enhancement tickets are created before the implementation starts. In agile development the enhancement tickets are created at the beginning of each iteration. They are solved in the implementation phase of ALM.

2.2 Application Life-cycle Management Tools

In the past there existed no specific ALM tools, but many small tools which handled one aspect of ALM. Nowadays many companies offer complete tool-sets that cover all parts of ALM. In the most companies there is a core tool that is extended by several other tools that cover the other aspects of ALM. Those tool-set are presented as complete ALM solutions.

Gartner describes in their yearly report about ALM tools five companies as leaders in the production of complete ALM tool-sets: Microsoft, IBM, Atlassian, Rally Software and CollabNet. They fulfill all of the following ALM requirements: requirements management, project management, quality management, defect management, build management, release management and task management. Their first challenger is HP, who has a market leader



Attribute Vendor	Task management	Build management	Agile	Saas	Mobile App
HP	HP Quality Center		x	x	x
Microsoft	Team Foundation Server	x	x	x	x
IBM	Rational Clear Request	x	x	x	x
Atlassian	Jira	x		x	x
Rally Software	Unlimited Edition	x	x	x	x
CollabNet	Team Forge Tracker	x	x	x	

Figure 2.4: ALM products

position in quality and test management, but still has not integrated all aspects of the SDLC like the build management. The leading vendors have all aspects of ALM in their tool-sets. Most of them have a core tool and built other tools that cover the other aspects of ALM (*Gartner @ONLINE* 2013).

Those six ALM tool solutions are analyzed closer in the aspects of task management, agile development, Software-as-a-Service(Saas) and mobile apps 2.4.

The Microsoft ALM solution is build around their Visual Studio Ultimate edition, which includes their Team Foundation Server and Share Point. Visual Studio Ultimate is sold as Microsoft ALM solution, all other editions do not have all parts of the ALM tool-set. A task tracker is integrated in the Team Foundation Server, which is part of Visual Studio Ultimate. They offer a agile process template for Visual Studio ALM. This can be loaded into the Visual Studio process tools. Several tools of the Team Foundation Server and the Share Point are available as Saas. Perfecto Mobile is a mobile extension of the Team Foundation Server, that gives mobile access to the Team Foundation Server and helps developing mobile applications (*Microsoft ALM tools @ONLINE* 2013).

IBM has a core set of six featured ALM products, with the prefix Rational: Rational TeamConcert, Rational ClearCase, Rational Buildforge, Rational Requirements Composer, Rational DOORS and Rational Quality Manager. Beside the six main tools several other small tools are offered by IBM with the focus on ALM. The task tracker is included in the Rational ClearRequest. Rational TeamConcert focus is agile ALM in the development process. All products are available as Saas. Rational Mobile is a not core tool that gives the possibility to control other products via a mobile phone and offers special features for mobile development, like mobile multi platform development (*IBM ALM @ONLINE* 2013).

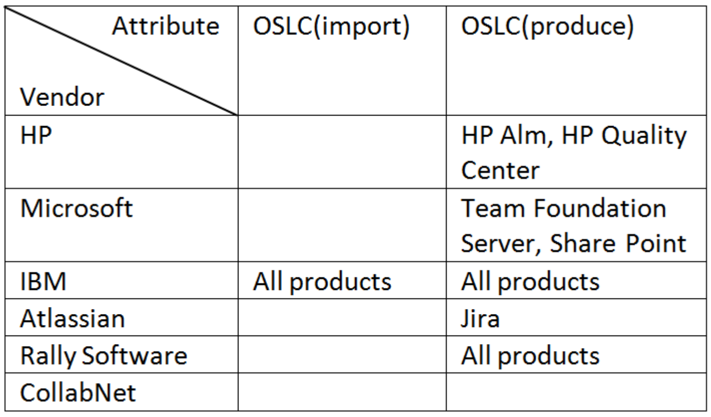
Atlassian offers around their market leading Project- and Issue-tracker Jira, several other tools in all fields of ALM. All other tools are designed to be able to work with Jira. They offer a huge amount of plug-ins and add-ons to Jira, which give the possibility to connect their tools with other non-Atlassian ALM tools (*Atlassian ALM @ONLINE* 2013).

Rally Software has a focus on agile development and Saas only. The task management process, which is the backlog in the agile development, can be connected with all other issue tracker like Jira, ClearQuest, through their apps and integration center. They offer apps for mobile project management and mobile development (*Rally ALM @ONLINE* 2013).

CollabNet's main focus is also Saas products with focus on agile development. Their core products are agile SDLC products: software versioning systems and continuous integration systems. Around those core products they build tools that handle all other parts of the ALM. They also offer integration solutions to integrate other issue tracker into their agile backlog solution in their task management (*Collab ALM @ONLINE* 2013).

HP ALM is based on HPs core strength the HP Quality Center. It is supported by the HP Requirements Manager, the HP Defects Management, the HP Business Process Testing and HP ALM. HP ALM combines all other tools to one ALM solution. All tools are available as Saas. The task and defect management is included into the HP Quality Center. In agile teams the HP Agile Manager can be included. (*HP ALM @ONLINE* 2013)

Open Service for Lifecycle Collaboration (OSLC)



Attribute Vendor	OSLC(import)	OSLC(produce)
HP		HP Alm, HP Quality Center
Microsoft		Team Foundation Server, Share Point
IBM	All products	All products
Atlassian		Jira
Rally Software		All products
CollabNet		

Figure 2.5: OSLC products

Although most companies have their own solutions and interfaces to import data from tools of other companies, the OSLC foundation was founded in 2008 to create a standard for the exchange of data between ALM tools. The standard is based on the W3C Linked Data specification and has the goal to be a open standard for the interaction between all ALM tools. It is hugely supported by IBM, but is also supported by more than 30 other user and producer of ALM tools. Nevertheless the only tools that support the importation of OSLC data yet are IBM tools. But several other tools support already the production of different OSLC based data, they are presented in Figure 2.5.
This gives IBM a huge advantage in the interaction between different ALM tools. Other tools like the Atlassian tool-set provide a huge amount of APIs to support the integration of the data of other tools. The advantage of a

open standard is, that it is possible to import any data from old or new implemented ALM tools, Atlassian and other ALM tool vendors have to provide APIs for every new supported tool.

The OSLC initiative has different work-groups for the different areas of OSLC data, like requirements management data, quality management data or change management data and a core work-group that is responsible for the interaction from data between the different data areas. Figure 2.6 shows the example of work-groups in the different areas and the core work-group (*OSLC @ONLINE* 2013; Stücka, 2013).

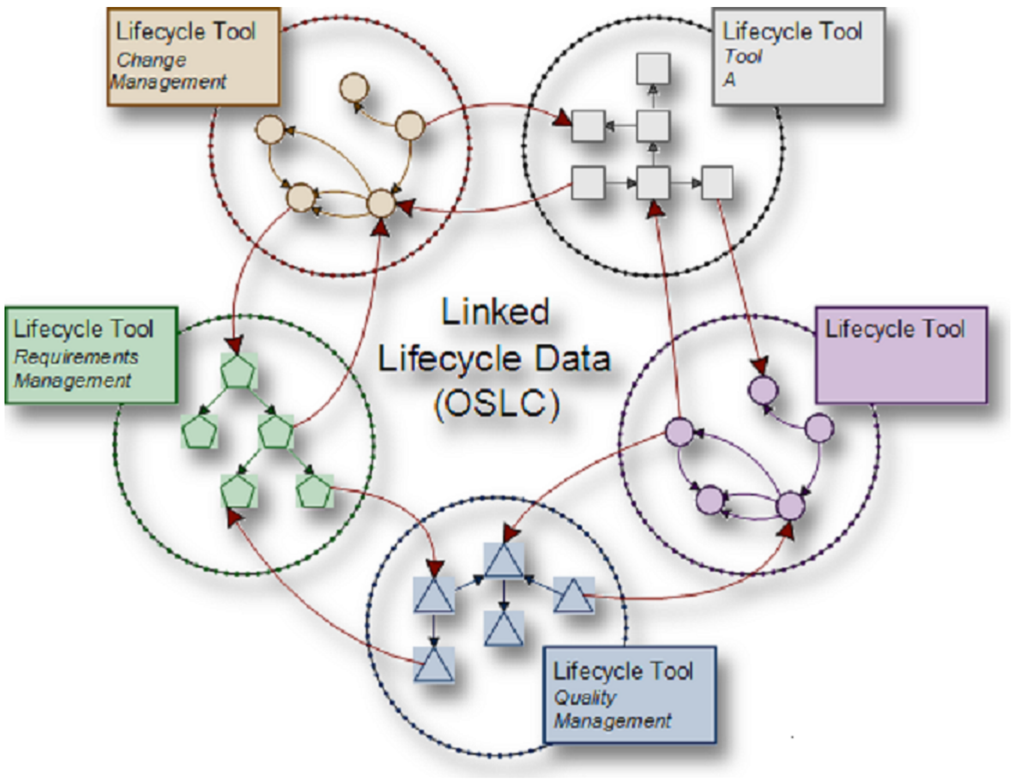


Figure 2.6: OSLC diagram (*OSLC @ONLINE* 2013)

2.3 Advantaged and Disadvantages

As described in this chapter ALM is an important factor in the production of software applications. All big players in the field of software producing and supporting tools offer complete ALM tool-sets. In those tools big data-sets with much information are created, stored and transferred between different ALM tools. This data can be used to monitor software processes and software products and improve the quality aspects of them. Therefor this data has to be measured. Chapter 3 introduces the measuring of software products and processes.

3 SW-Management in Numbers

Software measurement is an important factor in the production of software applications nowadays. It is a mechanism for creating a corporate memory and an aid in answering a variety of questions associated with the enactment of any software process. It helps support project planning, it allows us to determine the strengths and weaknesses of the current processes and products, it provides a rationale for adopting and refining techniques, it allows us to evaluate the quality of specific processes and products (Caldiera and Rombach, 1994).

According to Sneed (Sneed, Seidl, and Baumgartner, 2010) software is multidimensional artifact with many dimensions. Three of those dimensions are measurable: the quantity, the quality and the complexity dimension. There exist two main possibilities of measuring a software in a time aspect: continuous measuring and one time measuring. The goal of the continuous measuring in the production of the software is to reduce the complexity of the software, raise the quality of the software and change the quantity of a software. One time measuring is suitable to compare two different software products, for example when a manager has to decides, what product to buy or to find out, whether a software product fulfills a given standard or not (Sneed, Seidl, and Baumgartner, 2010).

Especially the measuring of the quality dimension is very complex, due to the high amount of attributes of software quality and the different sights on the quality of software. This chapter describes how to operate the measuring of software quality, by using quality models and software metrics.

3.1 Metrics and KPIs

Metrics are rules or functions how to measure a software on the source code level. They rely on the countables of a software application, like the lines-of-code, statements, procedures or input-output operations. The goal of measurement may never be the measurement itself, but has to serve a specific goal (Wagner et al., 2010). Those goals are defined through the business needs of a company. A business need can be to produce high quality software, to reduce the maintenance costs in the bug fixing of the software and satisfy the costumer. A company defines their most important goals as key performance indicators (KPI).

3.1.1 Metric

The IEEE Standard 1061 defines software metrics as a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality (Committee et al., 1998). Singh defines two main categories of software metrics: product metrics and process metrics (G. Singh, D. Singh, and V. Singh, 2011). Product metrics measure the attributes of a software product. Process metrics describe the attributes of the processes in the development of software applications. Those metrics quantify any step in a software process. Both categories can be divided in several subcategories.

For product metrics Sneed defines three main dimensions of metrics: the quantity metrics, the complexity metrics and the quality metrics. They are based on the three main dimensions of a software product (Sneed, Seidl, and Baumgartner, 2010).

- The quantity metrics describe any sizes of a software product. This can be the amount of lines in a database, the number of files or number of tests. A very commonly used base metric in this group is the Lines of Code (LOC) or number of executions metrics (Albrecht and Gaffney Jr, 1983). Those metrics are often connected with the required number of features in business metrics like the return on investment per line of code, or line of code per person. Due to the big amount of possibilities to define the size of a software, one big problem is to define which metric is relevant for which reason.

- The next metrics are metrics of the complexity dimension of a software application. Complexity metrics measures the connections between different part and elements of a software. It measures the amount of connections and how strong the connections between those elements are. Well known metrics of the complexity are the Halsted metrics , the McCabe metrics (McCabe, 1976) or the Sneed branch density metric (Sneed, Seidl, and Baumgartner, 2010). Again a problem is to define the relevant and irrelevant types of connections between elements of a software product to define the complexity.
- The last metric group measures the quality dimension. Quality metrics are difficult to define since there are a lot of aspects which define what quality is. For example many errors in a software product can state that the software product has less quality, or a good quality if all errors are known classified. A lot of changes in the software developing process can mean that this process has a bad quality and has to be improved or that it has a good quality because it is currently improved. It is difficult to define when a software product has too many bugs and how it is correlated to the quality of this product, this number is probably different in every product. Due to this fact general rules have to be considered as well as quality contracts between the costumer and the developer have to be signed. In this way a metric can check, if the code fulfills the expectations of the costumer in sense of cost and quality or the contract is failed. Quality metrics are a important part of the quality models defined in the next section.

ISO 1061 standard describes three different kinds of software product metrics: internal, external and quality in use metrics.

- Internal metrics do not rely on the execution of the code, for example a static code analysis.
- External metrics are applicable on the execution of an application and depend on input and influences.
- Quality in use metrics measures the numbers in the usage under real conditions. The external and the quality in use metrics should usually

depend on each other on correspond.

Software process metrics can be divided in three different groups: the software inventory, the work in process and the velocity group (Loper and Schmidt, 2005).

- The software inventory is the amount of work done in a software project. This group includes new features implemented in a software project as well as the work done to test a software application and solve bugs.
- Work-in-process are all ideas that are brought into an application. These metrics are important to measure the progress in a software project.
- Velocity is the effort that needs to be done to produce a working software. Those metrics are important to predict future effort and costs in software projects.

3.1.2 KPI

KPI represent a set of measures focusing of those aspects of organizational performance that are the most critical for the current and future success of the organization (Parmenter, 2010). KPIs are very critical numbers for the management of an organization they are usually known and identified by the management but sometimes unknown. The management of a company uses the KPIs to take strategical decisions. The connection between the metrics and the KPIs is that metrics measure on a lower abstraction level if the goal for a specific KPI is reached. The KPIs are a subset of the result of the most critical metrics available in the company. One way to break down the business goals like the KPIs, which are on a high abstraction level, down to software metrics, on a lower abstraction level, can be done by using the goal question metrics model (V. Basili et al., 2007; V. R. Basili, Heidrich, et al., 2007).

3.2 Software quality models

Quality is a important factor in the production of software applications. A way to master the high complexity of software quality in software producing companies, quality models are used to refine the abstraction of quality into defined and measureable quality attributes. There exist many different types of quality models, with different strengths and weaknesses. One possibility to categorize the different quality models is the Definition Assessment Prediction (DAP) classification Figure 3.1. In this classification there exist three different purposes of quality models (Deissenboeck et al., 2009):

- Definition: A quality model has to define, what the quality in the software products or processes in a company is or how it is defined. This purpose is high important in the management level of companies.
- Assesment: The abstract term quality has to be carried over into measurable entities. Measurement rules are called metrics. The assesment concernes in the development departments in companies.
- Prediction: A quality model with prediction purpose defines rules that can forecast the output of the measurable entities. Prediction is used in the project planning in companies.

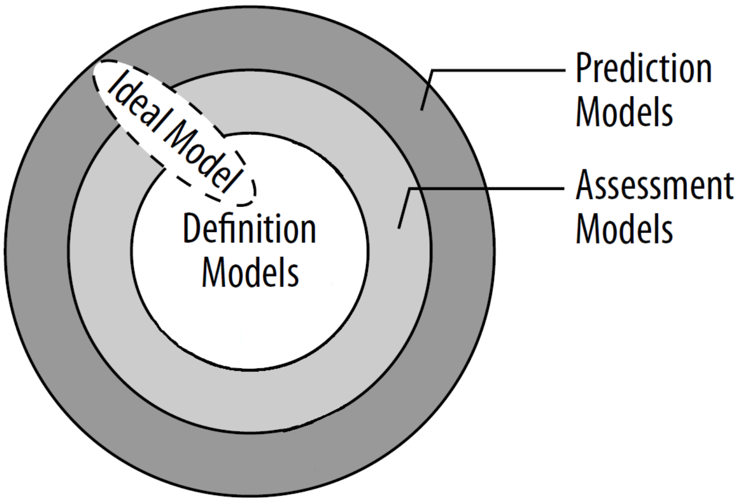


Figure 3.1: DAP Classification for Q-Models (Deissenboeck et al., 2009)

The different purposes of quality models depend on each other. It is not possible to create the assessment of one quality definition without knowing the quality definition. Furthermore it is not possible to forecast the output of a metric without knowing anything about the metric. This dependency is illustrated in Figure 3.1. Models that do not implement the purpose they depends on, need a model that implements this purpose. The ideal model would implement all three categories, but those models are very rare and cover usually only a small aspect of the quality in a company (Deissenboeck et al., 2009).

Especially in environments where software applications change fast, quality models have to be very flexible and adaptable. Therefor it is not possible to create one specific model with predefined definitions and assessment rules. A model that gives a framework to define a quality model, by giving constructs and rules needed to build such a quality model, is called quality meta model. Companies can use quality meta models to define their own quality model for their own needs. The two most know, widely used and adopted meta models are the Goal-Question-Metric and the Factors-Criteria-Metrics models (Wagner et al., 2010). Both models are meta models for the definition and especially for the assessment of quality attributes. Those meta models are introduced in the following section.

3.2.1 Goal-Question Metric (GQM)

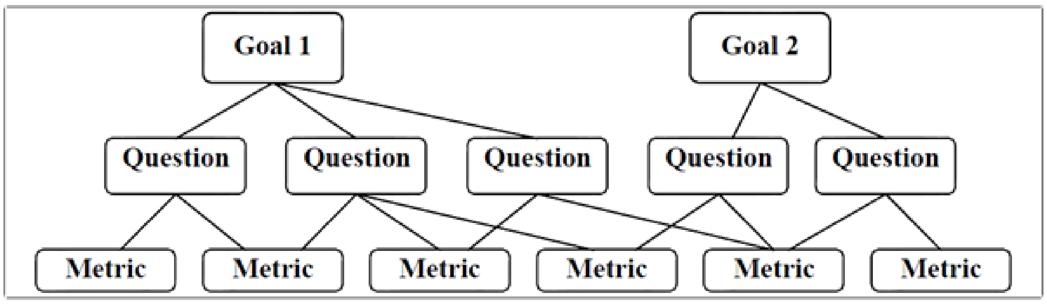


Figure 3.2: Goal Question Metric hierarchy (Caldiera and Rombach, 1994)

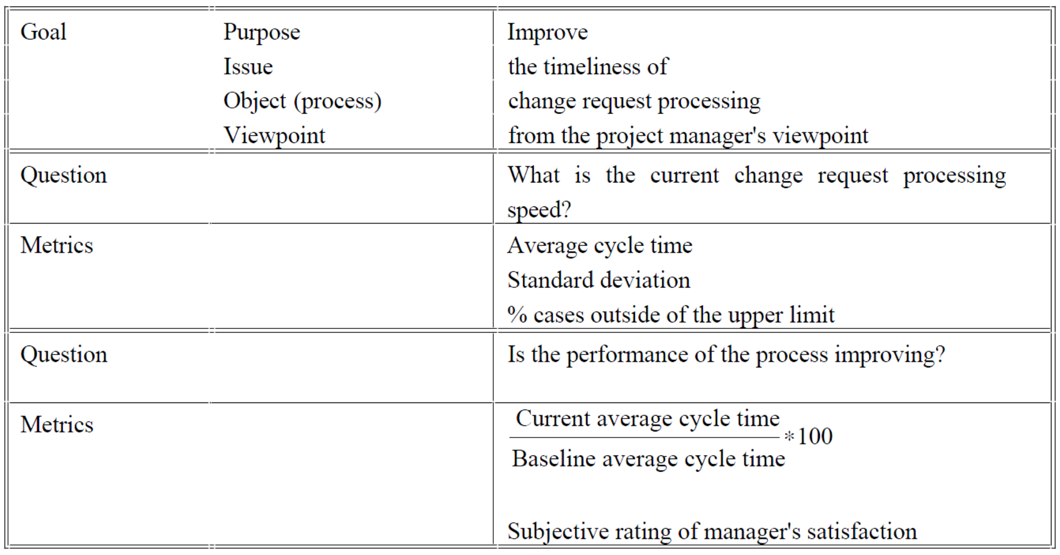
The GQM approach is first defined by Caldiera and describes a process to define goals of a software application and define step-wise metrics from the abstract goals (Caldiera and Rombach, 1994). The metrics can be product, resource or process oriented. The GQM bases on a NASA project to detect defects in a set of projects (V. R. Basili and Selby, 1984).

The way of generating the metrics with the GQM method consists of 3 levels:

- In the conceptual level the goals are defined. A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. The object can be a product, process or a resource. Ideally the goals are related to the business goals of a company.
- In the question level the goals are specified by creating questions that define the goals. A set of questions is used to characterize the way the assessment or achievement of a specific goal is going to be performed based on some characterizing model. Questions try to characterize the object of measurement (product, process, resource) with respect to a selected quality issue and to determine its quality from the selected viewpoint.
- In the metric level the data is connected to the questions. Metrics are created that quantify and measure the questions. The metric can have an objective or subjective scope. A objective metric depends only on the measured object, a subjective metric depends on the measured object and on a viewpoint, it is taken from. An objective metric can be the size of the code, a subjective metric can be the the readability of this code. (Caldiera and Rombach, 1994)

All three levels are ordered hierarchically, the hierarchy is shown in Figure 3.2. Each goal can have multiple questions and each question can be connected to multiple metrics. Metrics can answer more than one question, to avoid multiple implementations of metrics and make them reusable. The questions should define the goals as completely as possible to ensure that every relevant aspect is considered.

One output of the GQM model is a table for each goal. The purpose, issue, object and viewpoint of one goal are described. Each question for this goal is written below and under the metrics The metrics have to answer these questions. The Figure 3.3 shows an example of a bug/issue goal.



Goal	Purpose Issue Object (process) Viewpoint	Improve the timeliness of change request processing from the project manager's viewpoint
Question		What is the current change request processing speed?
Metrics		Average cycle time Standard deviation % cases outside of the upper limit
Question		Is the performance of the process improving?
Metrics		$\frac{\text{Current average cycle time}}{\text{Baseline average cycle time}} * 100$ Subjective rating of manager's satisfaction

Figure 3.3: Goal Question Metric table (Caldiera and Rombach, 1994)

Caldiera describes when measurement and furthermore the goals in the GQM method can be effective. Three objectives have to be considered:

- The model has to focus on specific goals. The purpose, the issue, the object and the viewpoint of the goal 3.3 should be described as complete as possible. This way the goals and the questions can be put easier in conjunction.
- It has to be applied to all life-cycle products, processes and resources. A goal is often influenced by all aspects of the life-cycle like products, processes and resources, if only one of those aspects is considered the goal can be miss-interpreted.
- The interpretation has to base on characterization and understanding of the organizational context, environment and goals. The quality model may never serve only itself, it has to serve a specific purpose,

this way the goals in the GQM goals have to set in context with those points.

Doran describes that a good goal has to have five attributes. The goal has to be specific, measurable, ambitious, realistic and terminated (Doran, 1981). In business companies this method is well known as the SMART method. Goals defined this way can be used in the first layer of the GQM method.

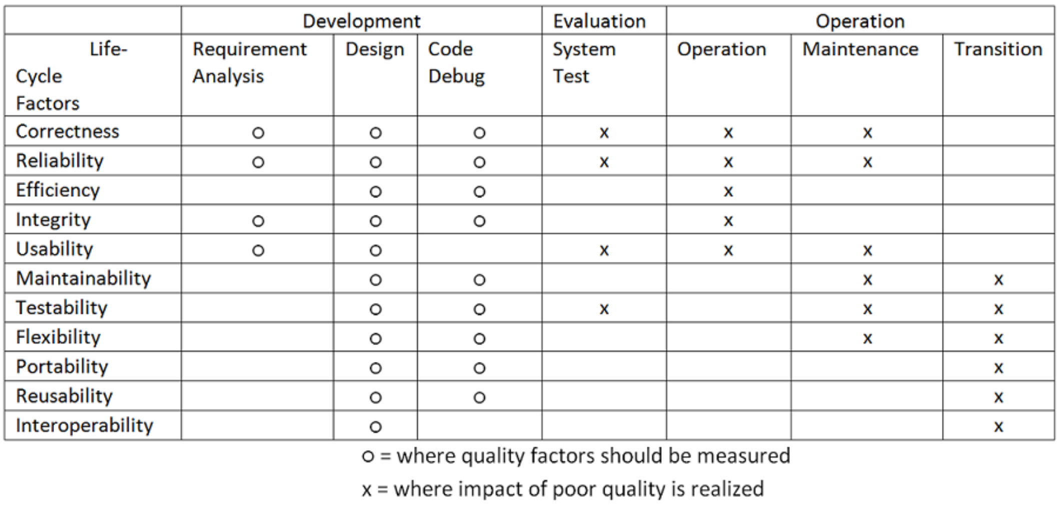
The GQM model and the way from the goals, to the questions and metrics can be considered as a bottom down model. Koziolek (Koziolek, 2008) enhances the model by the interpretation of goals in a bottom up way. Each metric has to be checked, if it answers the question. The questions have to be evaluated, if they are able to reach the goal and completely define the goal. In the bottom up step the questions and metrics are enhanced or adopted.

3.2.2 Factor-Criteria-Metrics (FCM)

The Factor-Criteria-Metric(FCM) model is defined for first time in 1977 by McCall in a document for the US Air-force to define important factors for their software application (McCall, Richards, and Walters, 1977). The quality factors defined in this meta model have a huge impact on the ISO-9126 metric quality model.
The FCM model is a hierarchical model. It is structured in the McCall quality tree with three layers. The root elements are the factors, they describe abstract software quality attributes in a business oriented way. Those factors are on a high business abstraction level and can easily be used as goals in business plans. The second layer consists of the criteria, they splint the factors into software oriented sub categories. In the last layer there are the metrics they describe how to measure the software according to the criteria. Each factor can have several criteria and each criterion can be measured by several metrics. Also one metric can be used for more than one criterion and one criterion can be used in more than one factor. One example of a definition of factors and their criteria is given following:

CORRECTNESS: Extent to which a program satisfies its specifications and fulfills the user's mission objectives.
Criteria: Traceability, Consistency, Completeness
MAINTAINABILITY: Effort required to locate and fix an error in an operational program.
Criteria: Consistency, Simplicity, Conciseness, Modularity, Self-Descriptiveness
TESTABILITY: Effort required to test a program to insure it performs its intended function.
Criteria: Simplicity, Modularity, Instrumentation, Self-Descriptiveness

A factor or criterion can have a positive, negative or no impact on other factors or criteria. It is very important to check those dependencies in the creation of the metrics. The metrics are created according to the criteria which describe the software quality attribute. Criteria can have also sub-criteria and sub-sub-criteria. McCall also describes the time, when a quality attribute has an impact on the software application and when the metric has to measure the criterion. All these dependencies are shown in the Table 3.4.



	Development			Evaluation	Operation		
Life-Cycle Factors	Requirement Analysis	Design	Code Debug	System Test	Operation	Maintenance	Transition
Correctness	o	o	o	x	x	x	
Reliability	o	o	o	x	x	x	
Efficiency		o	o		x		
Integrity	o	o	o		x		
Usability	o	o		x	x	x	
Maintainability		o	o			x	x
Testability		o	o	x		x	x
Flexibility		o	o			x	x
Portability		o	o				x
Reusability		o	o				x
Interoperability		o					x

o = where quality factors should be measured
x = where impact of poor quality is realized

Figure 3.4: Software Quality Metrics (McCall, Richards, and Walters, 1977)

The metrics should start measuring the criteria, at the time point in the life-cycle marked with a circle. The impact of a fault or wrong metric will be seen at the time point marked with an x. The longer an unknown fault exists and the higher the impact of the fault is the higher will be the costs of fixing the software. At this point the product metrics, which measure the criteria of the factors, can be connected with process metrics. The process metrics can break down the process to single steps and define at which point an application is. The process metric shows in which step of the process which product metric has to be applied.

The result of metrics is presented to the user. Applications that clean up, merge and present metrics to the user are called dashboards. A dashboard is one type of software monitoring tool. The following chapter gives a short overview about existing software monitoring tools.

4 Monitoring Tools

Monitoring a software application through the whole lifetime of the application is a very important part in every company nowadays. Monitoring of software products in every step from the creation to the end of the lifetime is part of the measuring of software and furthermore part of the quality management and is essential in the production of high quality products. Especially in software producing companies, where software is developed over a long time in several release cycles and one software product is the major part of the company's product portfolio, it is important to monitor this product - in the developing process as well as in the operation.

This chapter gives an overview on monitoring tools available in the market. It describes Business Intelligence tools, that use Online Analytical Processing(OLAP) as one way to monitor processes, products and resources in software producing companies. It also compares monitoring tools in the field of code quality and code testing. Finally one project is introduced that combines a business intelligence OLAP and a code quality and testing solution.

4.1 Business Intelligence tools based on OLAP

In the monitoring of software production and software projects it is important to investigate a huge amount of data from many sources of the SDLC, that interact with each other. It is necessary to observe this data in the aspect of the different dimensions of the software and in the aspect of different abstraction levels of the dimensions. Business Intelligence (BI) is the technology that focuses on this. It includes applications, infrastructure, tools, and best practices that enable access to and analysis of this data (*Business Intelligence @ONLINE* 2013).

One category of tools that implement BI is OLAP. The OLAP model is a way to present multidimensional data to the user and gives the possibility to interact with this data. The data is aggregated and presented to the user on a graphical user-interface, the dashboard. The data is represented in multidimensional cubes. The main functionality is that the cubes are drill-, slice- and dice-able. More information about the cubes is given in Chapter 5. The OLAP engine gets the data from data-warehouses by using multidimensional queries. It is possible to split the information of the data-warehouse into data-marts. Data-marts are a subgroups of the data-warehouse at a lower level. The data-warehouse can be filled in with any information from any source by using Extraction Transform Load (ETL) processes. The advantages of the OLAP model is that the user can browse online through all dimensions of the current data and history data.

This section provides information about the commercial BI OLAP tools of SAP, IBM Cognos and the open source tools of Jaspersoft, Pentaho.

4.1.1 SAP

SAP is a company founded in 1972 in Germany for System Analysis and Program Development. In their big product portfolio they also have a huge focus on Business Intelligence. One of their main monitoring product is their Business Warehouse in the SAP NetWeaver product portfolio, the SAP NetWeaver Business Warehouse. They also have separated reporting and presentation tools in product portfolio Business Intelligence. The reporting tools are the Crystal Report packages, the dashboard, the web Intelligence, the Visual Intelligence and some more.

The 4 main layers of the SAP BI solution are the ETL services, the Storage Services, the Analysis and Access Services and the Presentation services.

The ETL services can be a SAP ETL service but also third party implementations, with SAP certificate are allowed. They support extraction from several relation based databases as well as from external files, Extensible Markup Language(XML) data, Business Object Data Services and all SAP data services through different services and APIs. The transformation step transform sources like the DataObjects, DataSources and Info Cubes into the internal structure of the SAP BW by using several different transformation rules and routines. In the load step the data is loaded into the final objects called InfoProviders. The InfoProviders are predefined data objects. Those objects are stored into the DataStorage layer of the SAP BW.

The DataStorage layer has several manager to create and control data cubes and aggregation tables. The analytic and access services give the access to the data storage and provides all planning, analysis and navigational functions. It is possible to run any kind of transformations and run queries to the database. The access services gives also third party solutions the possibility to access the data in the Data Storage layer and run queries. Every query always accesses one InfoProvider. SAP offers the BEx query to design the queries, which supports drag and drop solutions to create queries of several dimensions. Queries give a multidimensional result, while the BDx Report solution returns more flat models. The presentation services uses the BEx Analyser, Report Designer, Web Analyzer and Web App Designer to create the visual output form the requested data. It supports several dashboards or the Crystal tool-set. It is also possible to present the information in own or third party dashboards or reports. (McDonald et al., 2002; *SAP BI @ONLINE* 2013)

4.1.2 IBM Cognos

IBM is one of the market leader in Business Intelligence tools and frameworks. The ETL service bases on the Cognos DataManager. It can be connected to any relational based data system and several other data systems. It also supports the importation of data from Microsoft EXCEL, or XML files.

Cognos supports, beside of the standard multidimensional cubes two special cube solutions. Cognos PowerCubes contain calculated and aggregated data that is organized as dimensions and measures you can view with Cognos Enterprise. Easier and faster access to precalculated summary data enables quick analysis. Dynamic cubes extend Dynamic Query in-memory acceleration to drive performance for dimensional analysis. Cognos Cube Designer can be used to create a cube definition from a relational data warehouse. Dynamic cubes help you optimize the value of enterprise data warehouses, which often have exploding data volumes. At the front-end Cognos supports several different kind of dashboards, scorecard solutions and analyzing tools, including the possibility to access those tools online and on mobile devices.

4.1.3 Pentaho BI Suite Community Edition (CE):

Pentaho is a software producing company with the main focus on Business Intelligence. Their core framework is based on open source, but they also offer commercial applications. Their commercial products are for example the bug and issue tracker Jiira with a direct interface to their business intelligence products or the Business Analytic and Big Data Analytic tools. Those tools are set on top of their core open source framework and offer a higher usability experience with drag and drop possibilities at the front-end, better and more chart possibilities and the connection to mobile devices. They also have the commercial support and training sessions of the open source tools in their Business Model. Their open source portfolio consists of the projects Kettle, Reporting, BI Platform, Mondrian, Weka, CDF, CBF, Saiku Analytics, CDA and Big Data (*Pentaho Community @ONLINE* 2013).

Kettle also know as Pentaho Data Integration Community Edition (PDI CE) is a ETL tool. It offers interfaces to many tools like their own bug tracker Jiira, but also open bug tracker like Bugzilla. Also they offer interfaces to many well known databases, for example MySQL (*MySQL @ONLINE* 2013) and Oracle. The user can define many transformation processes and manual jobs by himself. Kettle uses a meta-data driven approach to integrate the data into the BI Platform.

Mondrian is an OLAP engine written in Java. It executes queries written in the MultiDimensional eXpressions(MDX) language, reading data from a relational database (RDBMS), and presents the results in a multidimensional format via a Java API (*Pentaho Community @ONLINE* 2013). MDX is a multidimensional query language. It is a read only language and uses same syntax as SQL. It should not mixed or compared with SQL. MDX was introduced by Microsoft. Mondrian does not support all possibilities of the Microsoft language, but most of them. Many times in the forums and documentations the reference to the Microsoft MDX reference page. The cubes are defined in XML file and are directly connected to the relational database. It is possible to create several cubes in one file, but each query only accesses one cube. By using the MDX queries the cube is drill and slice-able. The elements like sets and members are described in Chapter 5. Saiku Analytics is another open source front-end and dashboard of Pentaho. Before the Saiku project started most projects had to use JPivot as front-end. It is licensed under Apache 2 and free available. It was former known as PAT (pentahoanalysistool) and is now a separated project. It is active under development and has a good community. Since it is still early in development a few features are still missing, compared to JPivot. Saiku offers the possibility to drag and drop dimensions to queries on run-time. Queries can be saved and loaded easily, the stored format is an XML format. It is not possible to parametrize queries. The application is Asynchronous JavaScript And XML(AJAX) based and multiple queries can be called during one session. Saiku has all diagram options, which are available in JPivot (*Saiku Analytics @ONLINE* 2013).

Reporting is a also known as Pentaho Reporting Community Edition (CE) and includes the Pentaho Report Designer, Pentaho Reporting Engine, Pentaho Reporting SDK and some libraries shared with the whole BI project. This suite of open-source reporting tools allows you to create relational and analytical reports from a wide range of data-sources and output types including: PDF, Excel, HTML, Text, Rich-Text-File and XML and CSV outputs of your data (*Pentaho BI @ONLINE* 2013).

4.1.4 Jaspersoft

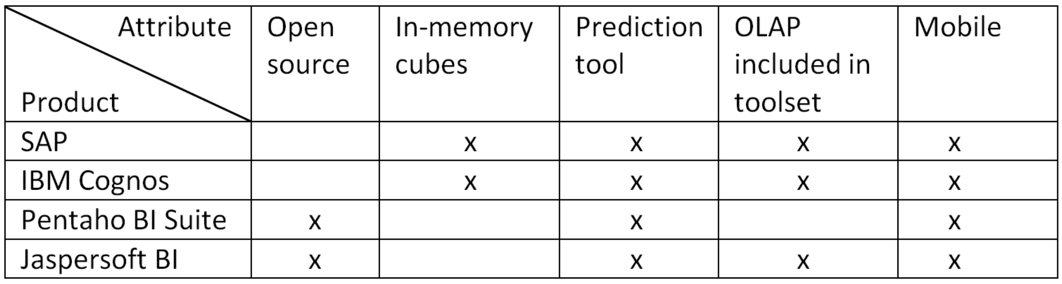
Jaspersoft is a commercial company, which has for every product a open source community edition of its products. The business model is similar to the one of Pentaho, they also have commercial features and commercial support and workshops, additionally to their open source products. Compared to Pentaho, Jaspersoft more focuses on adding commercial functionality to their products. Jaspersoft has currently the program Jaspersoft:European tour as their "The Open Source BI Conference". This way they try to supports open source in the BI world. Jaspersoft has 5 products in their portfolio as commercial as well as community edition. They are JasperReports Server, JasperReports Library, Jaspersoft ETL, Jaspersoft Studio and iReport Designer. The OLAP engine of Jaspersoft bases on the Mondrian 3.0.1 engine of Pentaho, they adopted this engine at a few parts. Jaspersoft ETL is the ETL tool of the Jaspersoft framework. It has the possibility to add data from more than 500 connectors and components. It is possible to add cron jobs to the importation process. The ETL tool adds the data to the Jaspersoft data-warehouse, where it is also possible to add the data to data-marts. The commercial version adds a dashboard to control the data importation and a data viewer. Furthermore a real-time synchronization is available.

The JasperReports Library is a web-enable JasperReports library it produces reports in PDF, XML, HTML, TXT and more formats. The commercial version adds the possibility to create charts, maps and widgets in flash. JasperReports Server is the report engine of Jaspersoft. It creates reports and is available as standalone version as well as a mobile and web version. The user-interface is customizable. The commercial version adds an ad-hoc web designer and the possibility to analyze relational and non-relational data.

Jaspersoft Studio is a eclipse-based framework designed to create own reports and implement own reports or customize the look of the reports. Jaspersoft Studio as no commercial version yet. iReports Designer is a netbeans-based framework designed to create own reports and implement own reports or customize the look of the reports. The commercial version supports the creation of flash based reports (*Jaspersoft BI @ONLINE* 2013).

4.1.5 Summary

The market of BI tools is very big and many tools are offered, open source software as well as commercial products. Often the commercial products are more optimized in memory usage and time to execute queries, by using special cubes that are created dynamically and can be stored in the random access memory. This is also an advantage, if much data has to be investigated. All investigated products have a at least small tool that can be used to predict numbers and results. Japspersoft, SAP and IBM Cognos have their OLAP engine included in their final product, Pentaho has a very famous open source engine that is also included in Jaspersofts engine, but does not combine it in a final product. It is already a market standard that BI companies have to offer mobile apps to access the data everywhere. A summary of the attributes of BI OLAP tools is given in Figure 4.1.



Attribute / Product	Open source	In-memory cubes	Prediction tool	OLAP included in toolset	Mobile
SAP		x	x	x	x
IBM Cognos		x	x	x	x
Pentaho BI Suite	x		x		x
Jaspersoft BI	x		x	x	x

Figure 4.1: BI OLAP tools

4.2 SW-qualtiy monitoring tools

This section introduces three tools that are used for software testing and include at least a set of software metrics. Modern quality monitoring tools include sets of software metrics to evaluate the tests and compares the metrics with metrics that do not depend on testing the code. Those tools need the source code for static analysis as well as a build management system for dynamic testing. Most testing tools can interact with versioning systems and can compare the results of the metric over time by storing the results of the tests and metrics.

4.2.1 SonarQube

SonarQube, previously known as Sonar, is a code analysis tool with the goal to manage code quality, offering visual reporting on and across projects and enabling to replay the past to follow metrics evolution. It is a open source project and covers seven aspects of code quality and its measurement: Architecture and Design, Duplications, Unit tests, Complexity, Potential Bugs, Coding Rules and Comments. It consists of several components. It can be connected to several continuous integration tools to check out code and enable continuous monitoring also in agile development. It includes build management systems, like Apache Maven or Apache Ant, this way the code can be monitored with static analysis and dynamic testing. It has a relational database in the back-end to store the results of the metrics and rules and it has a dashboard to present the result of the metrics and rules to the user. Most standard metrics, more than 600 coding rules, pattern and anti-pattern are implemented.
The dashboard is web-enabled. The result of the metrics, coding rules, pattern and anti-pattern can be monitored in the dashboard as well as the results of the build processes and the unit tests. Right now SonarQube supports more than 20 programming languages. Additional programming languages, patterns and metrics can be added by using the plug-in system. The metrics can be represented with several different diagrams, the dashboard has a helicopter overview as start page where the most important metrics are presented. All other metrics can be accessed through the navigation list. The dashboard includes a so called TimeMaschine, this tool helps to replay the past and shows the user how quality metrics evolve in time (*Sonarqube @ONLINE* 2013).

4.2.2 Parasoft

Parasoft is a software producing company, with the focus on software test tools. They claim that the majority of Fortune 500 companies rely on Parasoft in order to produce top-quality software consistently and efficiently. Their main products are Parasoft Test, Parasoft Virtualize and Parasoft Concerto. Parasoft Virtualize and Parasoft Concerto are responsible for the test management, environ management and the virtualization on different platforms. Parasoft Concerto enables the measuring of requirements, the task management and the project planning. It includes the Parasoft Process Center, which helps users to configure work-flows for business processes and measure those processes. Parasoft Test consists of the parts JTest for the programming language Java, dotTest for the .Net framework, C/C++Test for C and C++. Each tool part contains the possibility to do static analysis for patterns flow and metrics, unit testing, development testing, coverage testing and code review. Additionally for the programming language C and C++ Insure++ exists, which adds the possibility to test the software for memory errors. The tools SOATest and LOADTest give the user the possibility to create load tests and do cloud or web testing.
The result of the metrics and other test tools are presented to the user in a dashboard, with the possibility to create different diagrams. The results are stored in a database to create history based diagrams. The most important metrics are presented on one screen as architecture dashboard (*Parasoft @ONLINE* 2013).

4.2.3 Coverity

The main product of Coverity bases on a open source framework Stanford Checker, but is yet a commercial product. The center tool is the Coverity test engine, it is connected with CoveritySave a static analysis verification engine and the analysis pack, which contains dynamic testing, architecture analysis and analysis integration. The coding languages that are supported mainly are C, C++ and Java ,additional languages can be added through the integration tools. Through those integration sets can also third party metrics, code coverage rules, test execution and other tools integrated.

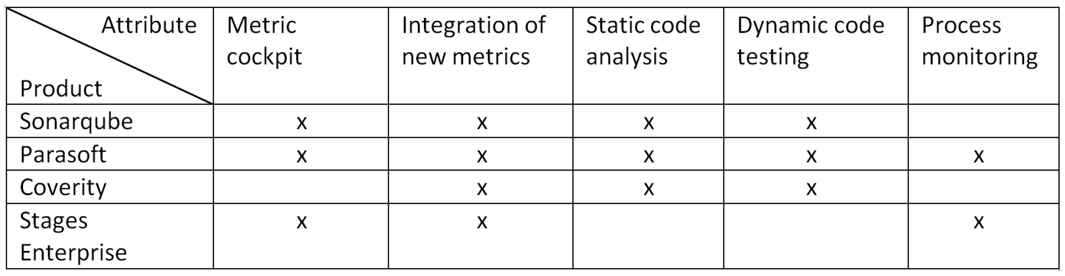
Coverity Connect is the center of the monitoring. It contains an issue manager it automatically includes all issues that are found through the test, code review or metric tools. It is possible to filter and prioritize the issues and navigate through the issues. This tool can assign issues automatically to free developer and calculates dependencies between issues. It has also a history database to improve the issue process and calculate several process metrics of the issue process.
Coverity Connect does not have a central cockpit integrated in the dashboard. In the Integrity Manager or Policy Manager tool in the dashboard section it is possible to create simple charts and reports for the metrics and result of other tests, but there is no cockpit in the center, that provides the most important information in one place. The Policy Manager can calculate different metrics and can send alert messages if the test policies are violated (*Coverity @ONLINE* 2013).

4.2.4 Stages Enterprise

Stages Enterprise is a tool from the software producing company method park and has the focus on monitoring and measuring business processes. Several different data sources can be connected to Stages Enterprise, with this data and the process models, process metrics are calculated. They have a huge set of process metrics included. Stages has a software dashboard and several reporting tools. Additional metrics can be included by using the plug-in center. A special feature is the possibility to connect Stage Enterprise to IBM Rational Team Concert. The processes can be designed by using different standard models and can be checked for compliance with models like CMMI, Automotive SPICE or ISO 26262. (*Stages @ONLINE* 2013)

4.2.5 Summary

Four software and software process quality analysing tools were investigated, which use software metrics to interpret the results and present them to the user. The first three tools use static analysis as well as dynamic testing and include build management tools. All tools have a huge amount of implemented metrics included and offer interfaces to include more metrics. The programming languages Java, C, C++ are supported, but the tools can be extended for other languages. Sonarqube and Parasoft have a cockpit where the most important metrics can be analyzed fast, this is an important helper for the management to get a fast overview of software applications and all projects. In Coverity diagrams and reports have to be generated manually. The metrics can show how the tests and metrics envolved in the past, therefor the tools provide a possibility to store results in databases. The tools Parasoft and Stages Enterprise give the possibility to model software business processes and monitore those processes. Both tools have a set of process metrics included, which can be enhanced and have a dashboard to present the results of the process metrics to the user. A summary of the attributes of test tools is given in Figure 4.2.



Attribute Product	Metric cockpit	Integration of new metrics	Static code analysis	Dynamic code testing	Process monitoring
Sonarqube	x	x	x	x	
Parasoft	x	x	x	x	x
Coverity		x	x	x	
Stages Enterprise	x	x			x

Figure 4.2: Metric based testing tools

4.3 Softnet Cockpit

The Softnet Softcockpit is BI project of the Softnet competence center in Hagenberg. It is a cooperation project between this competence center the university of Hagenberg and several business and industry partners. The prototype was tested in cooperation with Siemens at the SiTEMPPO(Success in Test Execution, Management, Planning, and rePorting Organizer) project. The goal of this project was to create a Sofware Cockpit, which is able to control the test management and quality management of a software product as well as the process of creating the software application. It should cover both, the advantages of test management and metric tools and and the advantages of the BI OLAP solutions. The project used only open source tools and own software solutions, which where integrated to a complete Business Intelligence product. The application consists of three main parts: The Data Adapters, The Data Warehouse and The user interface. (Albrecht and Gaffney Jr, 1983).

The adapter get the data from several sources periodical. The source can be for example bug trackers like Bugzilla for test process management, source code version control systems like CSV or Mercurial to get product based data to get metrics like Lines of Code(LOC). The adapters periodically extract relevant data from different repositories and databases, e.g., Bugzilla's issue database or the change log of CVS. The data is transformed to a standard data structure and stored in the central data warehouse.
The data warehouse organizes the data as cubes amenable for OLAP. The data schema supports recording the project history for analyzing the evolution and forecasting of trends.
The user interface of the cockpit visualizes aggregated information and offers the flexibility to customize views, metrics and models. The web-based implementation provides easy access to visual representation of the integrated data.The adapter is a ETL process for every source that should be integrated or one big ETL process which implements interfaces to all other applications. The data-warehouse organizes the data as cubes with an OLAP technology. The application used the open source Mondrian engine (Beer, 2009).
The user interface was implemented with Java Servlet Pages and the tool JPivot to integrate Mondrian.
A solution based on this Softnet SoftCockpit is used in the practical part of the master thesis explained in Chapter 5. In this chapter it is described how to implement a issue tracking process and metrics that measure this process in this Softcockpit.

5 Softnet Cockpit

The goal of the of the Softnet Cockpit is to combine the advantage of a software metric dashboard in the software quality management and the advantages of a BI solution based on the OLAP technology. The advantages of the metric dashboard is that prepared quality metrics are available very fast and can be observed by the management easily. The advantages of the BI solution is that a huge amount of data can be used for the measurement and it is easy to include more data sources in the future. The multidimensional OLAP model allows also different sights on the dimensions of the data as well as on different abstraction levels of the data.

In the first section a short overview of the architecture of the Softnet Cockpit is given. The second section describes the input data and requirements given by the partner company. The last part describes the implementation of the Softnet Cockpit more detailed and the changes done in practical part of this master thesis, which base on the input and requirements.

5.1 General Architecture

The architecture of the Softnet Cockpit solution consists of three parts: the data-warehouse, the multidimensional OLAP engine and a graphical front-end. The Softnet Cockpit uses a MySQL database with a multidimensional relational star schema as data-warehouse. As OLAP engine the open source tool Mondrian is embedded. It is possible to create multidimensional cubes and run MDX-queries on this cube, with this tool. As front-end JPivot is used. It is embedded in java servlet pages. This way the JPivot elements are accessible through the user on a web page. JPivot can take MDX queries and can execute them on the Mondrian engine. It creates results as multi-dimensional cubes and diagrams. It can create reports in the pdf or excel format. A overview of the architecture is given in Figure 5.1.

5.1.1 Front-end

The Software Cockpit dashboard runs on an Apache server. The entry points are several Java Servlet Pages (JSP) files. In these files the front-end library JPivot is embedded. JPivot is a JSP custom tag library that renders an OLAP table and let users perform typical OLAP navigation like slice and dice, drill down and roll up. JPivot also supports XMLA (XML for Analysis) data-source access. The JPivot multidimensional result table is embedded into the Softnet Cockpit data-warehouses JSPs. It supports the direct manipulation of the MDX queries inside the browser. A generation of diagrams forms the multidimensional output and the generation of reports in either PDF or Excel format. JPivot is not supported and developed anymore and Pentaho already uses Saiku as front-end successor. Also JPivot is older and does not support features like drag and drop it still has some advantages like the parametrization of queries, which is not supported in Saiku. (*JPivot @ONLINE* 2013)
The web-page consists of several navigation pages for each group of queries one page and for the later included snapshot queries, test queries with product groups and diagram optimized queries also one page. Queries are always in two possible versions available one full parametrized and one with the possibility to choose between several dimensions, which are shown as second dimensions beside the always shown product dimension. By clicking on a query link on the navigation page the selected query is loaded by JPivot the parameters are replaced and the query is executed by using the Mondrian engine. The result is presented by JPivot in a multidimensional result table and if chosen in a diagram. This page also includes the possibility to change and rerun a MDX query. The MDX queries are also stored in JSP files on the server. The functionality and construction of the queries will be explained in Chapter 6.

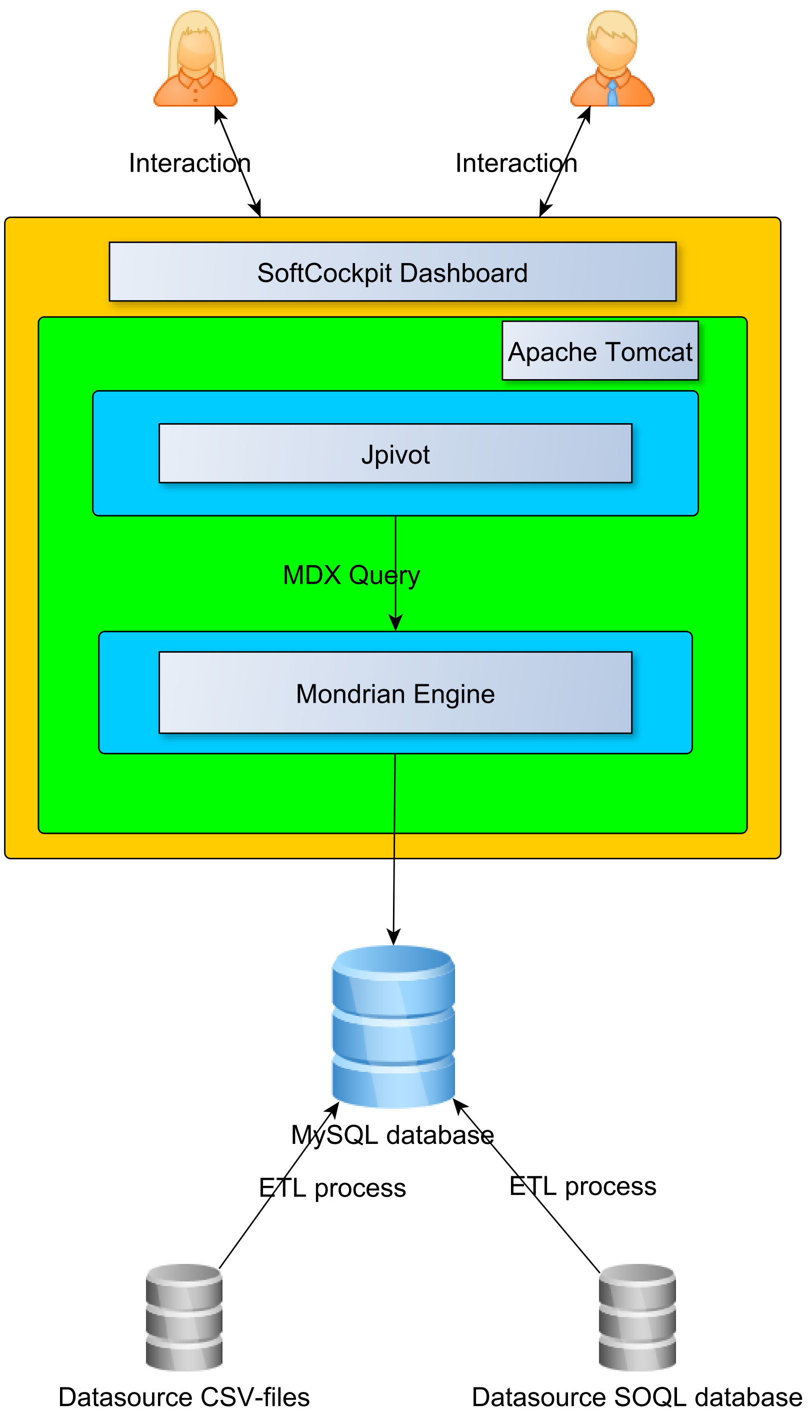


Figure 5.1: Architecture

5.1.2 Multidimensional OLAP engine

Mondrian is used as multidimensional OLAP engine and runs also on a Apache web server. It is set on top of the relational data-warehouse. Mondrian creates the multidimensional cubes and can execute MDX queries.

The multidimensional cube does consist of measures and dimensions (Datta and Thomas, 1999). Measures are the elements inside of the cube and base on the dimensions, that give additional information. Figure 5.2 has three dimensions: time, issues and severity and the measure elements: state change and product change. Dimensions consist of hierarchies, member, sets and tupels. The dimensions and cubes for the Mondrian engine are defined in a XML file. In this file the dimensions and cubes are connected to the tables in the data-warehouse The dimensions are defined in one part and linked to the cubes by a definition of the dimension usage.

A dimension consists of levels which are ordered in hierarchies. The hierarchies are used to drill through the different abstraction levels in the cubes.

A member is one item in one dimension. Every dimension consists only of its members. Each member has one value. It can be accessed in a query by a full qualified or by a member function. Members can be defined in a query, or predefined as predefined member in the cube definition, if only a subset of members in the same level is needed.

A set is an ordered collection of zero to n tupels. Like a member a set can be defined in the query or predefined in the definition of the cube. Many functions are offered in the library, which calculate operations over all elements of a set.

A tuple is composed of members of different dimensions. A simple tuple which contains only one element is also a member of the single dimension.

The cubes are drill-, slice-, dice- and pivot-able (Datta and Thomas, 1999).

- Slicing refers to selecting the dimensions used to view the cube.
- Dice refers to selecting actual positions or values on a dimension. Slice and dice together have the effect of reducing the dimensionality of the cube.
- Drill-down refers to decreasing the level of aggregation along one or more dimensional hierarchies, whereas roll-up refers to increasing the level of aggregation.
- Pivot refers to aggregating over one or more dimensions and producing a new cube having an attribute for each dimension and an additional attribute for the aggregated measure

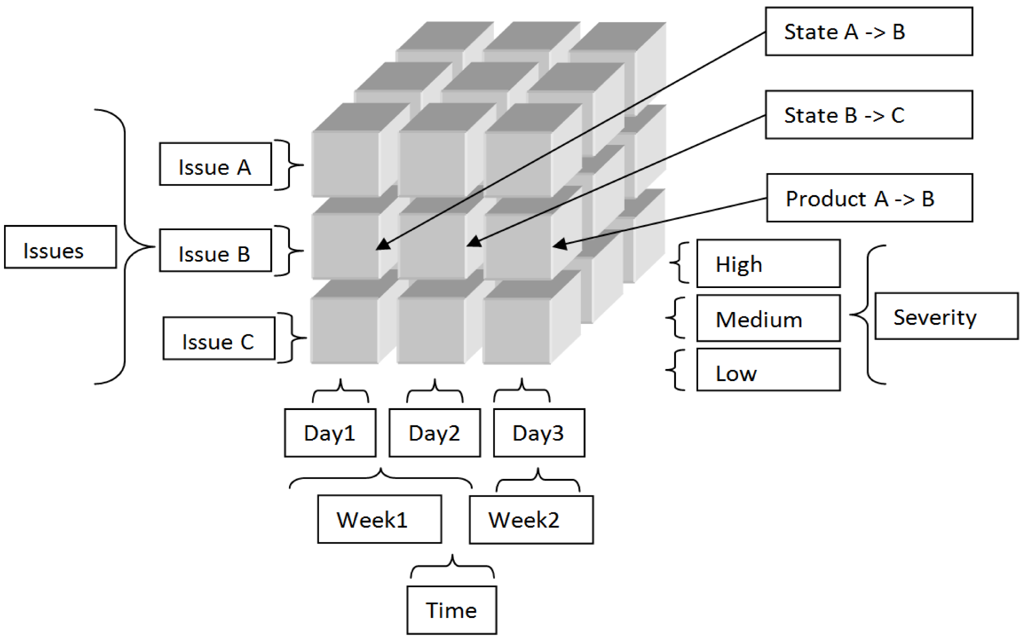


Figure 5.2: Cube

5.1.3 Data-warehouse

The data-warehouse is the data storage of the Softnet Cockpit. It consists of a relational MySQL database. It uses one of the most frequently used database pattern in the creation of multidimensional relational databases: the star schema pattern (Pedersen and Jensen, 2001). The star schema has one main table in center, which is called fact table. The fact table has one record for each discrete measurement. In the case of issue process measurement of the Softnet Cockpit, each discrete measurement is one change that happened in the issue process. One change is also called history change. It contains the whole information about the issue that changed and the changed variables. Some numeric information directly available in the fact table is also called measures. For further information the fact table is connected to a set of dimension tables. These tables describe, what is known in the context of each measurement record. For example information about the product, the issue or the severity of the issue. The history events describe the past, this way this data-warehouse is numerical and historical oriented.

The data is imported to the data-warehouse from other data sources by using Extraction, Transformation, Load (ETL) processes. These processes extract data from different data sources, transforms it into a form that fits into the data schema data-warehouse and loads it into the data-warehouse. This way the data of several sources can be imported into one data-warehouse (Vassiliadis, Simitsis, and Skiadopoulos, 2002).

5.2 Requirements for the practical part

This section describes the new requirements. The input given by the partner company can be divided in three parts. The process and the model of the process, the data correlated to the process and the metrics that should be implemented. This section describes these parts and gives additional information about requirements given by the company.

5.2.1 Issue process

The issue tracking process of the partner company is printed in Figure 5.3. It contains all states and the most important state changes, state changes that are not printed in the diagram can happen also. The process starts in the state *New* and ends in one of the final states: *Verified*, *Rejected*, *Duplicate* or *Elaboration*. The issue process can be divided in two main parts one includes the inflow state the other one the outflow states. All states until the state *Assigned* are inflow states, the states after *Assigned* are outflow states. In the inflow states the issue is checked and classified, in the outflow states the issue is worked on. When the issue changes from the inflow to the outflow part it has to be classified to a product and a developer, also attributes like the severity of the predicted development time have to be set.

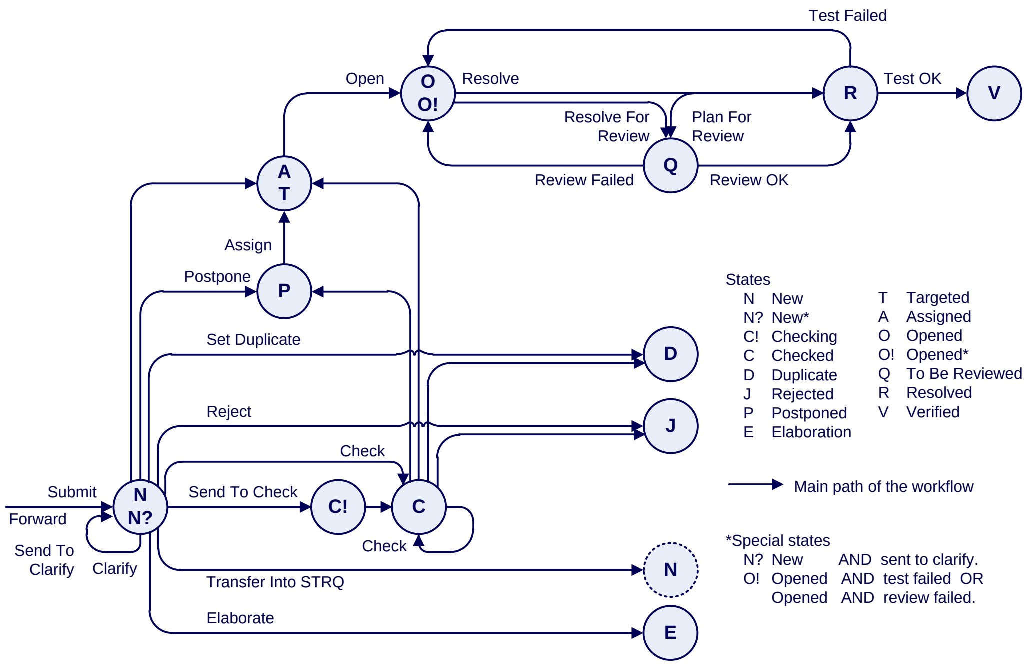


Figure 5.3: Model of the issue process of the partner company

Inflow states

An issue enters the process to the state *New* by using the created activity or by a state change from the empty state or a later state back to the state *New*. There are three final states that define that an issues is worked on more. *Duplicate* is the final state, for issues which are already reported. Issues in *Rejected* have been rejected for any reason. Issues in *Elaboration* need more work and will be opened later.

There are two possibilities, if more investigations are needed: the clarification and the checking. Issues in *Checking* state are sent there from the developer to the project leader or manager to solve open questions. If the questions are solved the project leader sends the issue into the *Checked* state. From *Checked* the issue can be send to *Postpone* if it can not be *Assigned* to a developer yet, to *Assigned*, or any of the final inflow states.

The other possibility is the clarification. The clarification process extends the main process with the clarification flag. This flag is treated like an additional state. This flag can have three different states: *Clarify*, *Sent* and empty. The initial value is always the empty, this means that the issue does not need to be clarified. A clarification activity happens at the creation of an issue if more information is needed to solve or identify the bug or enhancement. The manager or developer sets the issue as sent to be clarified and gets the clarification from the reporter. The difference between the checking states in the main process and the clarification side process is that the checking states occur internally between developer and project leader and the clarification process externally between reporter and the developer. After the clarification the issue is in the state *New* and can be send to the states *Assigned*, *Postponed*, *Checking*, or any final inflow state.

Outflow states

The issue is put from the state *Assigned* to *Opened*, if it is currently worked on. Issues may skip the state *Opened* if their estimated implementation time is very small, then, after the issue is finished, it is put directly into *Resolved*.

In the state *Resolved* the issue is tested, if the test fails the issue is put back into *Re-Open*, otherwise it is put into the final state *Verified*. *Re-Open* is a special Open state, if a state change to Open and a test fail, or a review fail happened. In the data-warehouse this state is treated as normal *Opened* state. The issue is put into the state *ToBeReviewed*, if a code review is necessary. After that the issue is put into *Re-Open* or *Resolved*.

Requirements

Clarification, test failed and review failed states and activities are not implemented in any part of the Softnet Cockpit and have to be implemented in all parts.

5.2.2 Process input data

The data that is read into the data-warehouse is read from data files. They are available as comma separated values (csv) files. After changes in the requirements of the input data, the data is now read directly from a Salesforce Object Query Language database. Each file is read from one database table. There are two different purposes of data files, one is to describe the mapping of ids to products or user, which are used in the other data files and to describe the mapping through different abstraction levels, like product levels. The second is to store the information and attributes of the issues like the creation time. The last one contains the history events that store the information about the changes, that happened in issues.

At the beginning of this project the four files UserFile, StructureFile, IssueFile, HistoryFile were used in this project. After problems with old data the two additional files: UserFileOld and StructureFileOld, were also included to the set of observable input data. This is the information in those data files:

- **UserFile**
  This file contains the whole set of all person involved into the issue process. Each person has one unique id, this id is used in the other data files.
- **UserFileOld**
  This file contains the names of person that changed their name during the run-time of the issue process. The old name is mapped to the new name. The mapping is necessary because the old name is still used in the history events.
- **StructureFile**
  All products are stored in this file. There exist several main products, each product can consist of several product components and each component can consist of several product component parts. Each combination of product, product component and product component part has one unique id.
- **StructureFileOld**
  All combinations of product, product component and product component part, that changed their name during the run-time of the issue process are mapped in this file, to their new name.
- **IssueFile:**
  This file contains all information available about one issue at the current time. About 50 attributes to one unique id of the issue. The attributes are for example, the owner of the issue, the creation time, the product it is mapped to, the severity or the current state.
- **HistoryFile**
  All history events of the issues are stored in this file. A history event has a unique id, its creation time, the name of the attribute that should be changed also called activity, the issue that is changed, the value of the attribute before and the after the history event. There are nine activities: created, state change, owner change, assignees change, product component part change, clarification change, test failed, review failed, and issue type change.

Requirements

The history events of the activities: product component part change, clarification change, test failed, review failed are not implemented and have to be included in the Softnet Cockpit. The requirements described at the beginning of this section were added during the project due to grown experience in the data importation.

5.2.3 Metrics

The initial requirements was to implement the metrics of the following groups. Those groups are explained raw. They contain also the metrics for clarification, owner change, test failed and review failed. Every metric contains a simple description in the goal question metric format, a mathematical explanation and a set of dimensions that have to be implemented for this metric. In the run-time of the project the parametrization became also a requirement. The queries had to be parametrized as well as the front-end pages of the Softnet Cockpit. Also snapshot metrics were added to the requirements, they add a additional sight on the time aspect of the metrics.

Metric groups

- **Number of issues to inflow states**
  This group contains the metrics that count all history events that lead to a state in the inflow section. It contains all metrics that count all history events that leave a state of the inflow section. This group contains also metrics that count the different clarification history events.
- **Number of issues to outflow states**
  This group contains the metrics that count all history events that lead to a state in the outflow section. It contains all metrics that count all history events that leave a state of the outflow section. This group contains also metrics that count the test or review failed history events. Two more complex metrics in this group calculate, if a issue has used the *Opened* state properly. If the estimated effort time is below a certain

time an issue may skip this state, if it is above this time the issue has to use the *Opened* state.

- **Elapsed time in inflow states**
  Metrics in this group calculate the minimum, maximum and average transition time a history change needed for the transition of one inflow state to the other one. The transition times of the clarification activities are also calculated.
- **Elapsed time in outflow states**
  Metrics in this group calculate the minimum, maximum and average transition time a history change needed for the transition of one outflow state to the other one. The transition times of the test failed and review failed activities are also calculated, as well as if a re-open happened correct.
- **Processing time until inflow states**
  Metrics in this group calculate the number of issues, minimum, maximum and average transition time of issues over several status changes, with a end state from the inflow states. These metrics are more complex because any amount of history changes with any activity can happen between the issue was in the beginning state and entered the end state.
- **Processing time until outflow states**
  Metrics in this group calculate the number of issues, minimum, maximum and average transition time of issues over several status changes, with a end state from the outflow states. These metrics are more complex because any amount of history changes with any activity can happen between the issue was in the beginning state and entered the end state.

Requirements

Several basic metrics in the first four chapter were implemented, the task is to implement all other metrics and the three most important metrics of the last two groups. This includes the metrics for the new dimensions: clarification, test failed, review failed and the more complex queries like the correct use of the state Open or Re-Open. All metrics have to be parametrized, because they have to be accessible in aspect of different dimensions, hierarchies, levels and values. Several different snapshot metrics have to be implemented, one snapshot metric for every state with different time-lines.

5.2.4 Other requirements

An installation procedure has to be developed that installs the whole application, including the set up of the relational database, the setup of the web-page and data-warehouse and the importation of the external data.

5.3 Implementation and changes to the Softnet Cockpit

This section describes the architecture of the Softnet Cockpit in a more detailed way. It takes also the requirements into consideration and describes what changes were done to the Softnet Cockpit in the practical part of this master thesis.

5.3.1 Relational database schema

The relational multidimensional database is the data-warehouse of the application. Figure 5.4 shows the structure of the relational multidimensional database.

In the center of the star schema is the fact table. The fact table contains the relevant information. A fact in this table is one history change that happened in the issue process or any attribute of an issue, like change of persons or products.

The fact table is connected to the dimension tables. These tables give the fact table additional information. The fact table has the primary key of each dimension table as foreign key, this way they are connected to each other. The initial database contained the dimension tables: Status, Activity, Severity, Milestone, Version, Priority, Issue, Time and Person. It contains also the view of the Status table named: PrevStatus. This view is necessary to identify the state that a issue had been, when a state change happens.

The fact table has also additional attributes that are not linked to dimensions. They are for example the unique Id or the TransitionTime.

5.3.2 Changes to the relational database

The changes done to the relational database star schema in the practical part of this master thesis are marked with a red arrow in Figure 5.4. Small changes have been done to the dim_product table, the last level of the product the product component part was added. Also three values have been added that document the product values for all product levels if the name of one product name changed. In the standard values there is the information about the current product name, in the added values about the old name. This way a metric can filter for issues until the name of the product changed and for all issues including old and new product name. This was needed because the name change of a product can also lead to a change of different aspects of this product. The table dim_prev_product is a view of the dim_product table and connected to the fact table. This table is used, if the product of one issue is changed to store the previous product value. This is a requirement in the product change metric.

The tables dim_clarification, dim_testfailed, and dim_reviewfailed were added according to the requirement to add those dimensions. Additionally a view for the clarification table was added, this is neccessary since a couple of clarification queries measure, which clarification state an issue left. The value in the dim_prev_clarification is the value of the previous history change of the same issue.

In the fact_table the links to the new created tables were added as foreign key. Additionally the field Parent_Id was added. The Parent_Id is the id of the history event that happened before the current history event. The Parent_Id is only used and set for state change history events. This value is used in the cube is used to create a parent child relationship between the state change history events in the cube definition. Each state history change is connected to the previous history change directly. In the queries this relationship is used for the complex metrics that calculate state changes through more than one following state changes in the issue tracking process.

5.3.3 ETL

The ETL process implemented for the data-warehouse is called process_csv. It is written in java and takes up to seven parameters. It supports the importation of data from csv files.The structure of the csv files is described in the previous input section. The ETL process is included into the installation of the Softnet data-warehouse, but can also be called separately at any time. It supports the sequentially adding of new history events to the fact table and adding of new issues from the issue table. It is not possible to detect changes in the issues or new history events automatically. If issues change it is advised to rerun a complete importation process.

Extraction

In the extraction phase the information is imported from the csv files and are stored into java beans, one bean for each input file. The java beans are stored into lists.

Transformation

The beans that give the information about the mapping of the products and persons do not need any transformation. They are written at the beginning into the corresponding dimensions. The most important beans in the transformation are the issue beans and the history beans. The issue beans contain the information about the issues and the history beans the information about the changes done to the issues. A fact in the fact table contains the information about the issue at one time point and the information about the change that happened.
At the beginning the values of the issue beans that can be changed by an

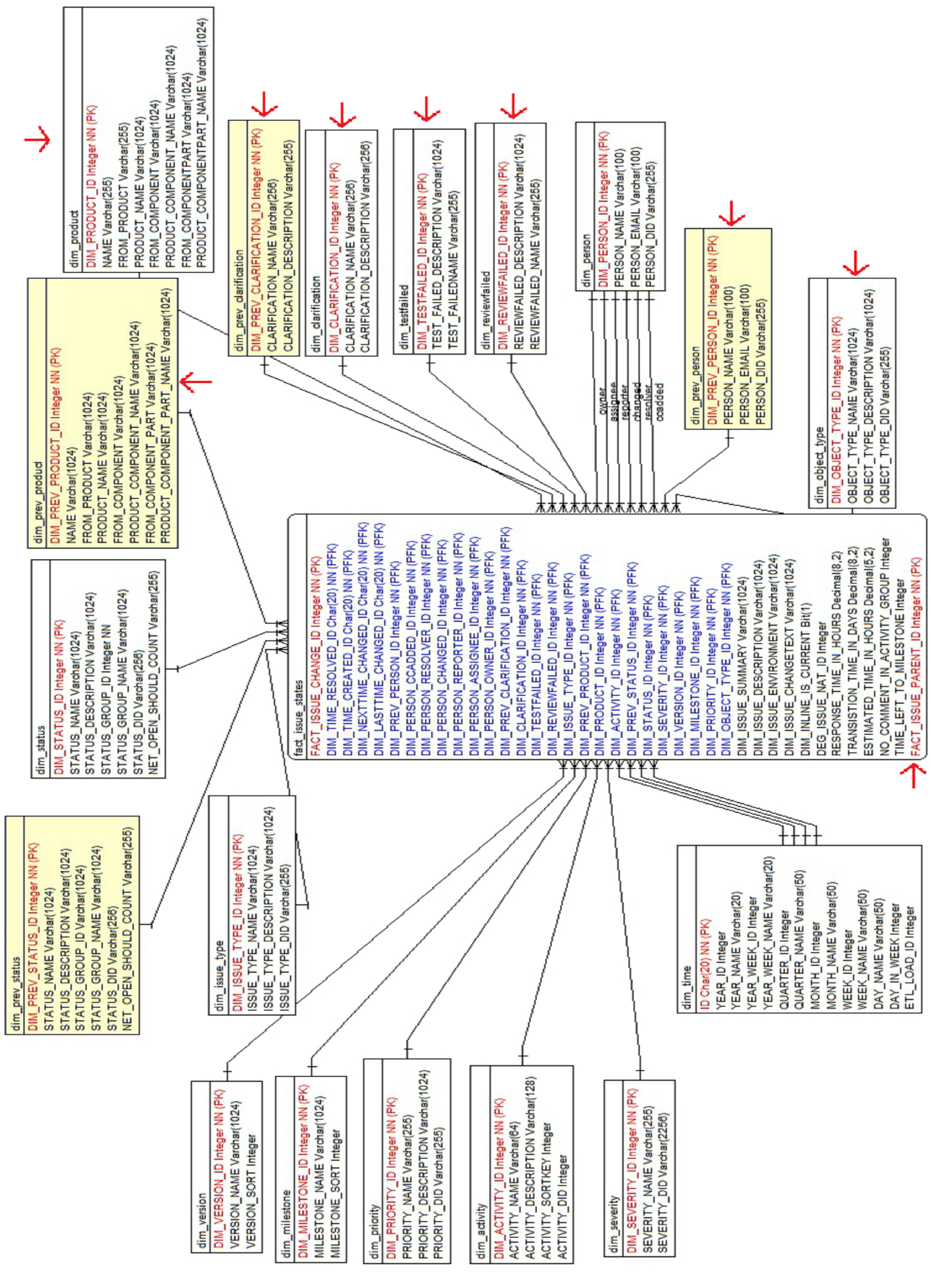

dim_product
DIM_PRODUCT_ID Integer NN (PK)
NAME Varchar(255)
FROM_PRODUCT Varchar(255)
PRODUCT_NAME Varchar(1024)
FROM_COMPONENT_NAME Varchar(1024)
PRODUCT_COMPONENT_NAME Varchar(1024)
FROM_COMPONENTPART Varchar(1024)
PRODUCT_COMPONENTPART_NAME Varchar(1024)
dim_prev_produd
DIM_PREV_PRODUCT_ID Integer NN (PK)
NAME Varchar(1024)
FROM_PRODUCT Varchar(1024)
PRODUCT_NAME Varchar(1024)
FROM_COMPONENT Varchar(1024)
PRODUCT_COMPONENT_NAME Varchar(1024)
FROM_COMPONENT_PART Varchar(1024)
PRODUCT_COMPONENT_PART_NAME Varchar(1024)
dim_prev_clarification
DIM_PREV_CLARIFICATION_ID Integer NN (PK)
CLARIFICATION_NAME Varchar(256)
CLARIFICATION_DESCRIPTION Varchar(255)
dim_clarification
DIM_CLARIFICATION_ID Integer NN (PK)
CLARIFICATION_NAME Varchar(256)
CLARIFICATION_DESCRIPTION Varchar(256)
dim_testfailed
DIM_TESTFAILED_ID Integer NN (PK)
TEST_FAILED_DESCRIPTION Varchar(1024)
TEST_FAILEDNAME Varchar(255)
dim_reviewfailed
DIM_REVIEWFAILED_ID Integer NN (PK)
REVIEWFAILED_DESCRIPTION Varchar(1024)
REVIEWFAILED_NAME Varchar(255)
dim_person
DIM_PERSON_ID Integer NN (PK)
PERSON_NAME Varchar(100)
PERSON_EMAIL Varchar(100)
PERSON_DID Varchar(255)
owner
assignee
reporter
changed
resolver
ccadded
dim_prev_person
DIM_PREV_PERSON_ID Integer NN (PK)
PERSON_NAME Varchar(100)
PERSON_EMAIL Varchar(100)
PERSON_DID Varchar(255)
dim_object_type
DIM_OBJECT_TYPE_ID Integer NN (PK)
OBJECT_TYPE_NAME Varchar(1024)
OBJECT_TYPE_DESCRIPTION Varchar(1024)
OBJECT_TYPE_DID Varchar(255)
fact_issue_states
FACT_ISSUE_CHANGE_ID Integer NN (PK)
DIM_TIME_RESOLVED_ID Char(20) NN (PFK)
DIM_TIME_CREATED_ID Char(20) NN (PFK)
DIM_NEXTTIME_CHANGED_ID Char(20) NN (PFK)
DIM_LASTTIME_CHANGED_ID Char(20) NN (PFK)
DIM_PREV_PERSON_ID Integer NN (PFK)
DIM_PERSON_CCADDED_ID Integer NN (PFK)
DIM_PERSON_RESOLVER_ID Integer NN (PFK)
DIM_PERSON_CHANGED_ID Integer NN (PFK)
DIM_PERSON_REPORTER_ID Integer NN (PFK)
DIM_PERSON_ASSIGNEE_ID Integer NN (PFK)
DIM_PERSON_OWNER_ID Integer NN (PFK)
DIM_PREV_CLARIFICATION_ID Integer NN (PFK)
DIM_CLARIFICATION_ID Integer NN (PFK)
DIM_TESTFAILED_ID Integer NN (PFK)
DIM_REVIEWFAILED_ID Integer NN (PFK)
DIM_ISSUE_TYPE_ID Integer NN (PFK)
DIM_PREV_PRODUCT_ID Integer NN (PFK)
DIM_PRODUCT_ID Integer NN (PFK)
DIM_ACTIVITY_ID Integer NN (PFK)
DIM_PREV_STATUS_ID Integer NN (PFK)
DIM_STATUS_ID Integer NN (PFK)
DIM_SEVERITY_ID Integer NN (PFK)
DIM_VERSION_ID Integer NN (PFK)
DIM_MILESTONE_ID Integer NN (PFK)
DIM_PRIORITY_ID Integer NN (PFK)
DIM_OBJECT_TYPE_ID Integer NN (PFK)
DIM_ISSUE_SUMMARY Varchar(1024)
DIM_ISSUE_DESCRIPTION Varchar(1024)
DIM_ISSUE_ENVIRONMENT Varchar(1024)
DIM_ISSUE_CHANGETEXT Varchar(1024)
DIM_INLINE_IS_CURRENT Bit(1)
DEG_ISSUE_NAT_ID Integer
RESPONSE_TIME_IN_HOURS Decimal(8,2)
TRANSITION_TIME_IN_DAYS Decimal(8,2)
ESTIMATED_TIME_IN_HOURS Decimal(5,2)
NO_COMMENT_IN_ACTIVITY_GROUP Integer
TIME_LEFT_TO_MILESTONE Integer
FACT_ISSUE_PARENT_ID Integer NN (PK)
dim_status
DIM_STATUS_ID Integer NN (PK)
STATUS_NAME Varchar(1024)
STATUS_DESCRIPTION Varchar(1024)
STATUS_GROUP_ID Integer NN
STATUS_GROUP_NAME Varchar(1024)
STATUS_DID Varchar(256)
NET_OPEN_SHOULD_COUNT Varchar(255)
dim_prev_status
DIM_PREV_STATUS_ID Integer NN (PK)
STATUS_NAME Varchar(1024)
STATUS_DESCRIPTION Varchar(1024)
STATUS_GROUP_ID Varchar(1024)
STATUS_GROUP_NAME Varchar(1024)
STATUS_DID Varchar(256)
NET_OPEN_SHOULD_COUNT Varchar(255)
dim_issue_type
DIM_ISSUE_TYPE_ID Integer NN (PK)
ISSUE_TYPE_NAME Varchar(1024)
ISSUE_TYPE_DESCRIPTION Varchar(1024)
ISSUE_TYPE_DID Varchar(255)
dim_time
ID Char(20) NN (PK)
YEAR_ID Integer
YEAR_NAME Varchar(20)
YEAR_WEEK_ID Integer
YEAR_WEEK_NAME Varchar(20)
QUARTER_ID Integer
QUARTER_NAME Varchar(50)
MONTH_ID Integer
MONTH_NAME Varchar(50)
WEEK_ID Integer
WEEK_NAME Varchar(50)
DAY_NAME Varchar(50)
DAY_IN_WEEK Integer
ETL_LOAD_ID Integer
dim_version
DIM_VERSION_ID Integer NN (PK)
VERSION_NAME Varchar(1024)
VERSION_SORT Integer
dim_milestone
DIM_MILESTONE_ID Integer NN (PK)
MILESTONE_NAME Varchar(1024)
MILESTONE_SORT Integer
dim_priority
DIM_PRIORITY_ID Integer NN (PK)
PRIORITY_NAME Varchar(255)
PRIORITY_DESCRIPTION Varchar(1024)
PRIORITY_DID Varchar(255)
dim_activity
DIM_ACTIVITY_ID Integer NN (PK)
ACTIVITY_NAME Varchar(64)
ACTIVITY_DESCRIPTION Varchar(128)
ACTIVITY_SORTKEY Integer
ACTIVITY_DID Integer
dim_severity
DIM_SEVERITY_ID Integer NN (PK)
SEVERITY_NAME Varchar(255)
SEVERITY_DID Varchar(2256)

Figure 5.4: Star schema

history change are set empty. The other values are kept same they have always the same value, the history is not available or not needed, for example the creation time of the issue will stay always same. After that the history beans are sorted according the their creation time from the oldest to the newest. Than the history beans are processed in this order.
For each history bean the issue bean that is linked to this history bean is searched and loaded. Than the type of change also called activity is identified and written into the activity field of the issue. Afterward the value of the field that should be changed in the issue is re-written according to the new value available in the history bean. If a view is available for this activity to store the previous value before the change happened this value is filled in now. At the end the last time changed field of the issue is overwritten. The issue bean is now written into the fact database as fact. The issue bean has the values at the time of the history change and can be changed by the next history change. This is done for all history changes.

Load and sort

The loading is the functionality of writing the information into the database. Information that is not mapped and never changes, like the states, is written into the database in the installation phase. The information that needs to be mapped before the transformation, the facts are loaded during the transformation. Additionally a sorting is done in the loading.

After all events from the history events are loaded the fact issue table contains all history events, this way it has the same number of rows as the number of history events. Now the sorting is started. In the sorting process the facts of each issue are sorted for their created date. The transition time between the history events with a status activity is calculated by subtracting the created dates of the two following history events. After the sorting the fact issue table is updated.

5.3.4 ETL Changes

In the extraction phase the possibility was added to import the data from a Salesforce database (*Salesforce @ONLINE* 2013). A connection and queries were implemented. The tables in the Salesforce database correspond to the csv files, this way no more adoption were necessary.

Also the extraction of the facts from the history table was adapted. The history changes with the following activities were also extracted, transformed and loaded into the data-warehouse: clarification, test failed, review failed, object type, product component part, owner.

The calculation of the parent id, which is necessary for the implementation of the parent child relationship, was included in to the load and sort part of the ETL process. In the sort section the facts are already sorted for each issue, that way it is easy to implement the parent id by writing the id of each state change fact to the following fact in the parent id field.

5.3.5 Mondrian cube definition

The primary key of the dimensions tables in the relational databases is a foreign key in the fact table. The cubes in Mondrian is created with the different dimensions, by a direct link to the database tables. The following table shows the connection of the MySQL database keys and dimension tables in the snowflake schema to the cube definition in Mondrian. Afterward the dimensions are explained. These dimensions are used in the queries, which implement the metrics, they are explained in the following chapter.

Column in the fact table of the database	Dimension table in the database	Dimension in Softnet cube definition
DIM_VERSION_ID	dim_version	Version
DIM_MILESTONE_ID	dim_milestone	Milestone
DIM_PRODUCT_ID	dim_product	Product
DIM_STATUS_ID	dim_status	Status
DIM_PREVSTATUS_ID	dim_status	PrevStatus
DIM_CLARIFICATION_ID	dim_clarification	Clarification
DIM_PREV _CLARIFICATION_ID	dim_clarification	PrevClarification
DIM_PREV_PRODUCT_ID	dim_product	PrevProduct
DIM_PREV_PERSON	dim_person	PrevPerson
DIM_PRIORITY_ID	dim_priority	Priority
DIM_SEVERITY_ID	dim_severity	Severity
DIM_ISSUE_TYPE_ID	dim_issuetype	IssueType
DIM_OBJECT_TYPE_ID	dim_objecttype	ObjectType
DIM_LAST _TIME_CHANGED_ID	dim_date	LastTimeChanged
DIM_LAST_TIME _CHANGED_HOURS_ID	INT	
DIM_TIME_CREATED_ID	dim_date	CreatedDate
DIM_PERSON_ASSIGNEE_ID	dim_person	Assignee
DIM_PERSON_REPORTER_ID	dim_person	Reporter
DIM_PERSON_CHANGED_ID	dim_person	Changer
DIM_PERSON_RESOLVED_ID	dim_person	Resolver
DIM_PERSON_OWNER_ID	dim_person	Owner
DIM_TEST_FAILED_ID	dim_testfailed	TestFailed
DIM_ACTIVITY_ID	dim_activity	Activity
DIM_REVIEW_FAILED_ID	dim_reviewfailed	ReviewFailed
DIM_TRANSITION _TIME_IN_DAYS	dim_date	Used by Measure

Status and PrevStatus dimension

The Status dimensions contains the 12 states of the issue tracking process, which is defined previously. A issue can have one state at the time of a history event. The PrevStatus dimension is used to define, which was the previous state of the issue. This PrevStatus dimension is only used for state change activities, otherwise this value would be the same as the Status dimension and would cause problems in state change queries. Both tables are filled in the installation process and change never.

Product and PrevProduct dimension

The product dimensions defines the product of an issue. It has 3 levels in the hierarchy: the product, the product component and the product component part hierarchy.

Milestone, Severity, Priority, Version, IssueType dimensions

The milestone, severity, priority, version, issue type dimensions describe the corresponding attribute of one issue. All dimensions have only one level in their hierarchy. These dimensions are automatically filled during the ETL process by adding an element to the table if a new element occurs in any of the input files, this way this dimension can increase in the run time of the Softnet Cockpit.

Person and PrevPerson dimension

The person dimensions describe, which persons are responsible for an issue. There are four person dimensions are defined in the Softnet data-warehouse cube: the owner, changer, assignee and reporter dimension. All of them refer to the same person table in the star schema database. There is also one PrevPerson table, which describes the previous person is a person change history event happened. The person dimension that changed and the PrevPerson table is referring to can be identified with the activity dimension.

If no person change happened, this dimension keeps empty. The person table in the database is filled during the ETL process through the person file and has a fixed size.

Clarification and PrevClarification dimension

Clarification is a dimension with two states and an empty row, that are filled in the installation of the Softnet Cockpit. Those two states are sent and clarified; both states do not depend on the other states of the states dimension. PrevClarification is set if a history change with the clarify activity happens and defines the previous clarification state.

Test and Review failed dimension

Those two dimensions are set, if a test or review failed activity happened. Each of the tables can have two values: true or false. They are filled into the table during the installation process and have a fixed size.

Activity dimension

An activity describes what changed during a history event. If a state change happens the activity is named after the state the activity leads into. All other activities are named similar to dimension where the change happened. The activity table is filled during the installation process of the Softnet data-warehouse and has a fixed number of elements.

FactIssue dimension

This dimension is the fact table itself implemented in the cube. This dimension is mainly implemented, to create the parent child relationship between the state change history events. By using the fact table as dimension and implementing a parent child relation in this dimension for the elements in this dimension it is possible to navigate through all state change history events of one issue and use parent child function on those history events.

Time Dimensions

There are three time dimensions defined in the cube definition of the Softnet data-warehouse. They all refer to the same time table in the database. This table is filled during the installation of the application on the server. All dates have to be filled in the database manually, if one date does not exist in this table it has to be added. The date table in the database contains one or more columns for the year, quarter, month, week and day. The diagram shows how those date levels are ordered hierarchically in the cube definition and can be seen as a tree structure. This structure helps to drill through the cube later in the results, every date can be extended to the lowest granularity the date.

CreatedDate

This time dimension refers to the creation time of the issues, this time never changes during the lifetime of the issues. It has all date levels. It is filled in the ETL process during the importation of the issues. The id in the fact table is connected to the date table and contains one day as id.

LastChangedDate

This dimension refers to the last change date of an issue. It is connected to the date table of the database. It has all date levels. It refers to the last history event that occurred for one issue. It changes for an issue every time a history event occurs. The time between two state change dates is the transition time of two history events.

LastChangedDateWeek

This date dimension is the same dimension as LastChangedDate, but only has the root level and the week level. In several metrics company A needs the level week for all weeks of one year and only this level. The drill down option of the time dimension lead to confusion in this use case, because a few weeks were displayed two times. The reason was because if a week starts in one month and ends in the following one it is displayed in both as full week. The only way to solve this problem was to create a separated dimension.

Measure Dimensions

Measure is a special dimension in the cube of the data-warehouse. This dimension is not connected to any table of the data-warehouse. It is connected directly to elements inside the fact table or values which are not connected to any table. Measures have an aggregator like sum, average or count. This aggregator defines a mathematical operation done through the run-time of a query on a amount of defined facts. The measure dimension has only the root level and one level below, where the measures are defined.

IssuesStatesCount

This measure is a sum up of all history events in the cube or calculated cube, if the cube is splinted or drilled down. The maximum number of this measure is the number of facts in the fact table, where each fact represents one history event.

CurrentIssueCount

This measure counts only the last history event happened to an issues which fulfill the requirements of the query. This measure is used in the snapshot queries explained in Chapter 6. The maximum number of this measure is the number of issues in the database.

MinAgeInDays, MaxAgeInDays, AvgAgeInDays

These measures count the minimum, maximum and average age in days for each issue which fulfills the requirements and are counted in the count measure. The age is calculated from the creation of the issue in the issue file to the last change that happened. The average age is calculated, if it can be rolled up automatically by Mondrian, in more complex queries it has to be calculated manually.

MinTransitionTime, MaxTransitionTime and AvgTransitionTime

These measures count the time between one state history event and the next state history event of the same issue. It is only calculated between status history events and is not available for non state activities. It is calculated during the sort process of the ETL process. In some queries the average transition time cannot be rolled up by the Mondrian, in this case the average transition time has to be calculated manually.

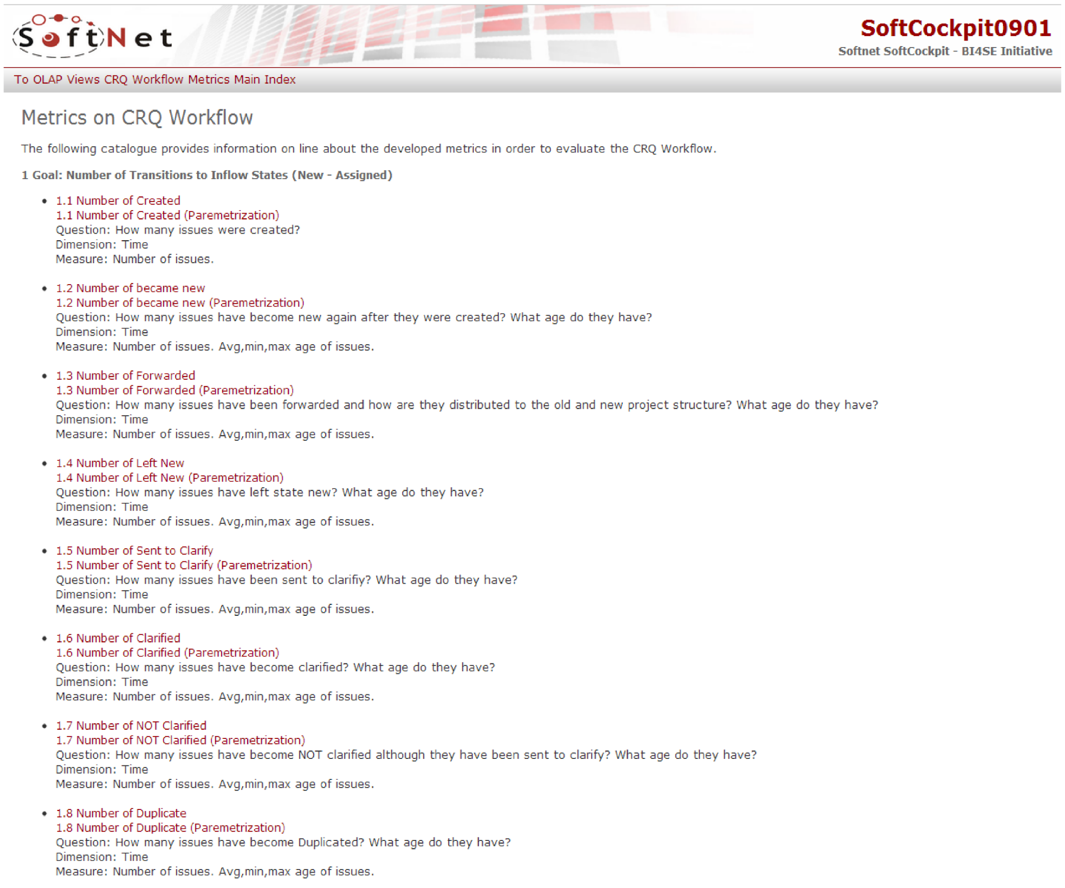
5.3.6 Changes in the cube definition

The dimensions Clarification, TestFailed, ReviewFailed, PrevClarification, PrevPerson, PrevProduct were added to the cube. Also the LastChangedDateWeek time dimension was added. Some minor additions were done by adding e few predefined member and sets, that are needed later in the queries. The parent child relationship between in the FactIssue dimension was added.

5.3.7 JPivot web-pages

The start page of the web portal shows, where links to the different groups of metrics are presented and a sub group where the metrics that base on new requirements, like snapshot or the week time line. The new metrics lead directly to the result page, the group links lead to a page where all metrics of this group are presented as web links. Every metric is available as parametrized version or standard version, more to the parametrization is explained in Chapter 6. By clicking on a metric link the main result page is loaded.
In the center the result page shows the multidimensional result. It is possible to navigate through this result cube by clicking on a plus this element is drilled down, by clicking on a minus this element is rolled up. Slicing and dicing can be done by manipulating the query directly. Above the multidimensional result there is a navigation bar. With the navigation buttons it is possible to get to the MDX query explorer. Per default this explorer shows the last query that was executed. In this field the query can be manipulated and executed. Another option is to create diagrams and change the options of this diagram. More information about the diagram is given in Chapter-chap:Querys. The last buttons give the possibility to create reports. Those reports consist of the multidimensional result, the diagram, if the option is chosen and can be generated in the formats pdf and excel.
Figure 5.5 shows the navigation page to the first eight metrics of group one of the metrics. All metrics are have two links to execute the implemented query and present it on the result screen. One is the parametrized version one the standard version. Additional information is given for both all metrics. Figure 5.6 shows the most important parts of the result page for one query. The title is the name of the metric. Below the title there is the navigation bar. The two big buttons on the top are save and load button, the implemented save and load option has some errors and was not used in this project. The three buttons on the right give the option to create reports and change the reports options. The buttons with the small diagrams give the possibility to create a diagram of the current result of the query and change the diagram options. The button with the letters MDX opens the MDX query explorer. It is activated right now and is presented below the navigation bar. The query can be changed, set back and executed. On the bottom there is

the multidimensional result. On the left axis, there are two dimensions, the IssueType dimension, which is drilled down and the Version dimensions, which is rolled up. Both dimensions have only one level below of the highest level. On the top axis there is one measure IssuesStatesCount, which is combined with the time dimension CreatedDate. The time dimension is drilled down to the year level, this dimension contains more levels below the year level and can be drilled down further.


SoftNet
SoftCockpit0901
Softnet SoftCockpit - BI4SE Initiative
To OLAP Views CRQ Workflow Metrics Main Index

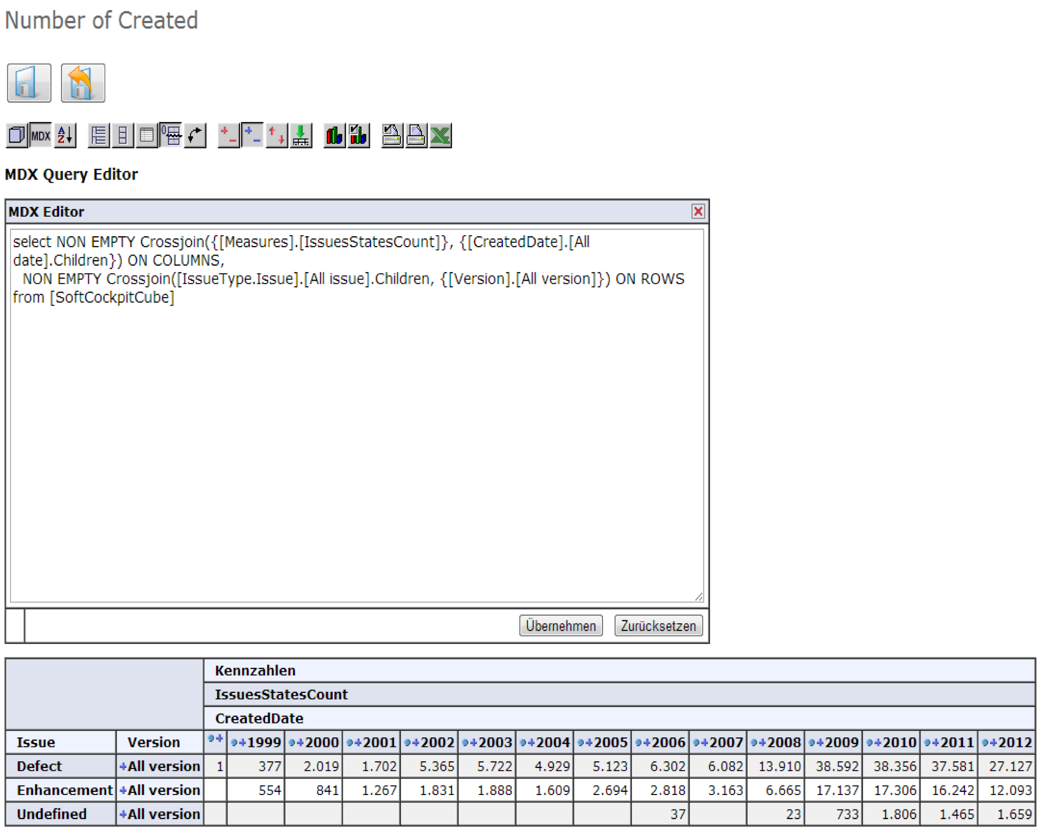
Metrics on CRQ Workflow

The following catalogue provides information on line about the developed metrics in order to evaluate the CRQ Workflow.

1 Goal: Number of Transitions to Inflow States (New - Assigned)

- 1.1 Number of Created
  1.1 Number of Created (Paremetrization)
  Question: How many issues were created?
  Dimension: Time
  Measure: Number of issues.
- 1.2 Number of became new
  1.2 Number of became new (Paremetrization)
  Question: How many issues have become new again after they were created? What age do they have?
  Dimension: Time
  Measure: Number of issues. Avg,min,max age of issues.
- 1.3 Number of Forwarded
  1.3 Number of Forwarded (Paremetrization)
  Question: How many issues have been forwarded and how are they distributed to the old and new project structure? What age do they have?
  Dimension: Time
  Measure: Number of issues. Avg,min,max age of issues.
- 1.4 Number of Left New
  1.4 Number of Left New (Paremetrization)
  Question: How many issues have left state new? What age do they have?
  Dimension: Time
  Measure: Number of issues. Avg,min,max age of issues.
- 1.5 Number of Sent to Clarify
  1.5 Number of Sent to Clarify (Paremetrization)
  Question: How many issues have been sent to clarifiy? What age do they have?
  Dimension: Time
  Measure: Number of issues. Avg,min,max age of issues.
- 1.6 Number of Clarified
  1.6 Number of Clarified (Paremetrization)
  Question: How many issues have become clarified? What age do they have?
  Dimension: Time
  Measure: Number of issues. Avg,min,max age of issues.
- 1.7 Number of NOT Clarified
  1.7 Number of NOT Clarified (Paremetrization)
  Question: How many issues have become NOT clarified although they have been sent to clarify? What age do they have?
  Dimension: Time
  Measure: Number of issues. Avg,min,max age of issues.
- 1.8 Number of Duplicate
  1.8 Number of Duplicate (Paremetrization)
  Question: How many issues have become Duplicated? What age do they have?
  Dimension: Time
  Measure: Number of issues. Avg,min,max age of issues.

Figure 5.5: Navigation page of metric group 1



```
select NON EMPTY Crossjoin({[Measures].[IssuesStatesCount]}, {[CreatedDate].[All
date].Children}) ON COLUMNS,
  NON EMPTY Crossjoin([IssueType.Issue].[All issue].Children, {[Version].[All version]}) ON ROWS
from [SoftCockpitCube]
```

		Kennzahlen														
		IssuesStatesCount														
		CreatedDate														
Issue	Version		1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Defect	+All version	1	377	2.019	1.702	5.365	5.722	4.929	5.123	6.302	6.082	13.910	38.592	38.356	37.581	27.127
Enhancement	+All version		554	841	1.267	1.831	1.888	1.609	2.694	2.818	3.163	6.665	17.137	17.306	16.242	12.093
Undefined	+All version									37		23	733	1.806	1.465	1.659

Figure 5.6: Result page of the Softnet Cockpit

5.3.8 JPivot web-pages Changes

The main changes done to the front-end was to insert the premade queries, which are implemented according to the set of metrics given by our partner company and the parametrization of the queries. More information about the queries and the parametrization of the queries is given in the next chapter.

5.3.9 Installation

The prerequisites of the installation process are the installation package, a Tomcat Apache Server and a MySQL server. During the installation process the database and its tables in star form are set up. The state, activity and date tables are filled in with the predefined values, which never change during run-time. The data-warehouse and the MDX queries are copied to the Apache Tomcat server. Optionally the ETL process can be run. If the ETL process is run and the option of importing the data from Salesforce is activated in the ETL process the installation needs an internet connection.

6 Queries and query templates

Queries are a important part of the Softnet Cockpit, they give the possibility to access the data in the data-warehouse. In Pentaho Mondrian (*Pentaho Community @ONLINE* 2013) the queries are written in MDX, a multidimensional query format introduced by Microsoft (*Microsoft MDX reference page @ONLINE* 2013). In the Softnet Cockpit solution, the front-end, JPivot, loads the MDX queries from the query files, or takes them from a MDX editor in the browser. After that it changes the queries by replacing the parameters, adds drill down options and passes them to the Mondrian engine. Mondrian parses each query, creates structured query language (SQL) queries and executes those SQL queries on the MySQL database, the data-warehouse. The results of the SQL queries are interpreted by Mondrian and used to create multidimensional cubes. JPivot presents those multidimensional cubes graphically and gives the option to navigate through that cube. A navigation operation, like the drill through the hierarchies, creates a new MDX query, which is executed again.

This chapter describes how the queries in the Sofnet Cockpit are implemented generally. After that queries are described, that were implemented according to the requirements given in the previous chapter. In the last section of this chapter, it is described, how to parametrize the queries. The parametrization of the queries was also a requirement.

6.1 General query

This section describes the general construction of a MDX query and the most important parts of a MDX query.

Listing 6.1: Example query

```
With Member/Set .. AS ..
Select .. On Rows
  On Column
From ..
Where ..
```

Listing 6.1 shows a general example. A MDX query consists of 4 main parts: The first part is the *With* part. In this part members of dimensions and sets are defined. A definition can be a simple assign of a value or a complex calculation. It is important to take care of the order of members, if the member depend on each other. Members, which have to be calculated first, have to be ranked first.
The second part, the *Select* part, defines which values should be printed on which axis of the multidimensional cube. The most common way is to use two axes the *Row* and the *Column*. The Microsoft documentation allows more than two axes, but Mondrian does not allow more than two. No matter how many axes are used the statement *Select* is only called once. The statement *On* stands in front of the name of the axis. On each axis several dimensions can be combined, but every dimension can only occur on one axis.
The *From* statement defines the cube, which is used in this query. Every cube definition file can have several cube definitions, but the query can only use one cube.
The *Where* section contains the slicer of the query. The observed cube is reduced to the members and sets defined in this section. It is very important to use this part wisely, to ensure the correct result and reduce the duration of each query. The execution time depends on the amount of data that is observed in one query. A dimension, which occurs in the *Where* section, cannot occur on an axis.

6.2 Queries

This section describes queries, which were implemented according to the metrics given by the partner company. The metrics are splinted in six groups, which are described in Chapter 5. The metrics were created by using the goal question metric method. The queries explained in this section are examples of real queries in the productive system. Usually the queries occur in several different versions, where the dimensions or measure dimensions are exchanged. This way the metrics can be observed according to several different dimensions without the necessity of changing the query in the editor. There are also parametrized versions of the queries in the Softnet Cockpit. They are described in the last section of this chapter.

6.2.1 State queries

State queries examine the behavior of the issues that change their state. The previous state and following state of the state of one issue can be observed at the time of a state change. The state dimension is always part of the query and is joined with all other dimensions. If a query should monitor issues over more than one state change, the more complex parent-child queries are needed. These queries calculate the number of state changes to or from an state, queries that calculate the number of issues in one state are the snapshot queries. The state change queries can be divided into three subgroups:

- Number of changes group:
  These queries show all state change history events, which lead from one state to another state, by using the activity dimension in the *Where* clause. The activity is named after the state the issue changes to.
- Elapsed time in group:
  Elapsed time queries show the time between the history events of the issue. It is the time an issue needs from one state to the next, it is called transition time. The minimum, maximum and average transition time of all history changes in one query are calculated.

- Left state group:
  These queries show all state change history events, that lead to another state and filter for its previous state. They use the *PrevStatus* dimension in the *Where* clause to identify the previous state and present all history events, that lead to this state. These queries are usually combined with either elapsed time or number of query, as additional information.

Listing 6.2: State query 1

```
Select NON EMPTY Crossjoin({[LastChangedDate].[All date].Children}, {[Measures].[
    IssuesStatesCount], [Measures].[MinTransitionTimeInDays], [Measures].[
    MaxTransitionTimeInDays], [Measures].[AvgTransitionTimeInDays]}) ON COLUMNS,
 NON EMPTY Crossjoin([Product].[All product].Children, {[Version].[All version]})
     ON ROWS
From [SoftCockpitCube]
Where [PrevStatus].[All Status].[POSTPONED]
```

Listing 6.3: State query 2

```
Select NON EMPTY Crossjoin({[LastChangedDate].[All date].Children}, {[Measures].[
    IssuesStatesCount], [Measures].[MinTransitionTimeInDays], [Measures].[
    MaxTransitionTimeInDays], [Measures].[AvgTransitionTimeInDays]}) ON COLUMNS,
 NON EMPTY Crossjoin([Product].[All product].Children, {[Version].[All version]})
     ON ROWS
From [SoftCockpitCube]
Where [Activity].[All activity].[POSTPONED]
```

This first query, in Listing 6.2, calculates all history events, which left the state *Postponed* and presents the minimum, maximum and average transition time from *Postponed* to any another state. In the result, on the row the product and the version of the history changes and on the column the transition time and the date of the history changes are shown. The second query, presented in Listing 6.3, measures all history events that lead into the state *Postponed*. In this query it is enough to filter for the activity the history event leads into, because there is an activity for each state history event.

6.2.2 Clarification queries

Clarification is a special dimension and represents a state an issue can have additionally to the other states. All clarification queries filter for the clarification activity in their *Where* clause and are combined with one clarification state: *Sent*, *Clarify* or empty. The subgroups of the state queries can also be applied to this group.

Listing 6.4: Clarification query

```
Select NON EMPTY Crossjoin({[LastChangedDate].[All date].Children}, {[Measures].[
    IssuesStatesCount], [Measures].[MinAgeInDays], [Measures].[MaxAgeInDays], [
    Measures].[AvgAgeInDays]}) ON COLUMNS,
  NON EMPTY Crossjoin([Product].[All product].Children, {[Version].[All version]})
      ON ROWS
From [SoftCockpitCube]
Where ([Activity].[All activities].[txt_Clarification__c], [Clarification].[All
    clarification].[Sent]
```

The query in Listing 6.4 checks for all history events, which lead to the state *Sent* of the clarification dimension. It shows the age of the issues in these events. The average age is calculated separately, because Mondrian cannot roll this measure up in this query. The other values are the same as in the state query before.

6.2.3 Test, Review failed and Re-Open queries

Test and review failed are history events, that can occur in the states *Resolved*, *ToBeReviewed* or *Verified*. They lead into the special *Open* state, *Re-Open*, which is internally treated like a normal *Open*. Both the state change history event and the test or review failed history event can occur independently. There are two queries which identify, if an issue in the state *Open* is a real open or a *Re-Open*, by checking the current activity and the previous state of the history events. Two other queries check if a test or review failed activity occurred in the correct state.

Listing 6.5: Test query 1

```
With member [PrevStatus].[Tested] as 'Aggregate({[PrevStatus].[All Status].[
    RESOLVED], [PrevStatus].[All Status].[VERIFIED]})'
  member [Measures].[AvgAge] as '(([Measures].[SumAgeInDays], [PrevStatus].[Tested
      ], [Activity].[All activities].[opened]) / [Measures].[IssuesStatesCount])'
Select NON EMPTY Crossjoin({[LastChangedDate].[All date].Children}, {[Measures].[
    IssuesStatesCount], [Measures].[MinAgeInDays], [Measures].[MaxAgeInDays], [
    Measures].[AvgAge]}) ON COLUMNS,
  NON EMPTY Crossjoin([Product].[All product].Children, {[Milestone].[All
      milestone]}) ON ROWS
From [SoftCockpitCube]
Where ([PrevStatus].[Tested], [Activity].[All activities].[opened])
```

Listing 6.6: Test query 2

```
Select NON EMPTY Crossjoin({[LastChangedDate].[All date].Children}, {[Measures].[
    IssuesStatesCount], [Measures].[MinAgeInDays], [Measures].[MaxAgeInDays], [
    Measures].[AvgAge]}) ON COLUMNS,
  NON EMPTY Crossjoin([Product].[All product].Children, {[Milestone].[All
      milestone]}) ON ROWS
From [SoftCockpitCube]
Where ([Activity].[All activities].[txt_testfailed])
```

The query in Listing 6.5 shows all *Re-Open* history events. It shows all events, which have been in the states *Verified* or *Resolved* and lead to the state *Open*, by filtering history changes with the open activity in the *Where* clause. The number of issues, and their minimum, maximum and average age are calculated. A state change from state *Resolved* or *Verified* should only happens, if a test fails.
The result of this query should be compared with the result of the query that calculates the number of test failed history events. The number of the test fails and the state changes to the state *Re-Open* have to correspond. This is calculated in the query presented in Listing 6.6.

6.2.4 Re-assignment or product change queries

Re-Assign and *ProductComponentPart* changes are history events, which occur independently from any other history event. The queries check how many of those changes happened and when they did happen.

Listing 6.7: Product component query

```
With set [NonEmpty] as 'Except({[Product].[All product].Children}, {[Product].[All
     product].[].[].[]})'
  member [Product].[NonEmpty] as 'Aggregate([NonEmpty])'
  set [NonEmptyPrev] as 'Except({[PrevProduct].[All product].Children}, {[
      PrevProduct].[All product].[].[].[]})'
  member [PrevProduct].[NonEmpty] as 'Aggregate([NonEmptyPrev])'
  member [Measures].[Avg] as '(([Measures].[SumAgeInDays], [Activity].[All
      activities].[pkl_ComponentPart__c], [PrevProduct].[NonEmpty]) / [Measures].[
      IssuesStatesCount])'
Select NON EMPTY Crossjoin({[LastChangedDate].[All date].[2011], [LastChangedDate
    ].[All date].[2012]}, {[Measures].[IssuesStatesCount], [Measures].[
    MinAgeInDays], [Measures].[MaxAgeInDays], [Measures].[Avg]}) ON COLUMNS,
  NON EMPTY Crossjoin([Product].[All product].Children, {[Severity].[All severity
      ]}) ON ROWS
From [SoftCockpitCube]
Where ([Activity].[All activities].[pkl_ComponentPart__c], [PrevProduct].[NonEmpty
    ])
```

Listing 6.7 shows a query that calculates all history events, which caused the change of a product component part of the product dimension. The duration of the execution of these queries for all years is very long, therefor the data-set in the time dimension is restricted to the years 2011 and 2012. The Mondrian engine cannot roll up the average age in this query, due to the complexity of the calculation between different members. The average has to be calculated manually.

6.2.5 Snapshot queries

Snapshot queries show the number of issues that are in one given time-point in a certain state. It would also be possible to show the number of issues in other non status activities. All other queries calculate the changes into or from one state, or other history events, with a non state activity. A snapshot query calculates the number of issues in one state, by subtracting all history events, that document when a issue enters a specific state, by all history events that document, if a issue enters this state. This means that for the calculation of one snapshot day all previous days in the history of the database have to be investigated. The execution time for the calculation raises for each additional snapshot day more than in other queries, since for every additional snapshot day all other previous days have to be included. It was decided to investigate issues only for one day in each week. The execution time for 52 days is much better as 356 days in one year.
It is necessary to create the snapshot days that include all previous days manually in the cube definition. As a example for the year 2012 a data set with 52 snapshot days was implemented. This set is named *Dates* and is a predefined set in the cube definition file. It contains for each last day of each week an element as member of the *LastChangedDate* time dimension. Each member contains all days from the beginning of the period to this day.

Listing 6.8: Snapshot query

```
With member [Measures].[Count] as '([Measures].[IssuesStatesCount],[Activity].[All
      activities].[assigned])-([Measures].[IssuesStatesCount],[PrevStatus].[All
    Status].[assigned])'
Select NON EMPTY ([Measures].[Count]) ON COLUMNS,
  NON EMPTY [Dates] ON ROWS
From [SoftCockpitCube]
```

Listing 6.8 presents a snapshot query. It calculates for 52 snapshot days in the year 2012 the number of issues in the state *Assigned*. The snapshot days are stored in the predefined *Dates* set and are presented on the *Rows* axis. On the *Columns* axis the snapshot measure is printed. This measure is defined as a member in this query. The measure takes all history changes under the condition that their activity leads to the state *Assigned* and subtracts this number by all history changes that have the previous state *Assigned*.

6.2.6 Parent-Child queries

Parent-Child queries monitor issues through more than history state change. It is easy to query for one state change or past state change in the history, by checking their activities in the history. Monitoring of state changes over more than one state, with a mandatory start and end point is not trivial, because it is the only kind of query, that has to consider the dependecies between history changes. An issue can be affected by any amount of history events and can be in any amount of states between the history event to the start state and to the end state. To solve this problem the parent-child hierarchy was introduced in Chapter 5. It is used to connect the history changes of one issue, which depend on each other. In the database the facts between state changes of a same issue were connected, by giving one fact the issue fact of the previous state change fact as parent id. In the cube definition these ids are used to create a parent-child relationship. Usually this relationship is used to create the hierarchy inside of dimensions, for example in the product dimension *Product*, *ProductPart* and *ProductComponentPart*. In the parent-child relationship of the issue tracking model, it is used to connect the state changes between the states of the issue tracking model. The queries can use the functions, which are intended to navigate through parent-child relations between the dimensions in our process oriented model. The navigation through the state facts in the fact table itself is possible by defining the fact table as dimension. Additionally the attribute activity was set inside this dimension, this way it is possible to access the activity of a current state change two times, inside this query and through the activity dimension in one query. This is necessary to define a start and an end activity.

Listing 6.9: Parent child query

```
With member [Measures].[Activity] as 'IIf(([FactIssue].CurrentMember.Level.Name =
    "FactIssueID"), [FactIssue].CurrentMember.Properties("Activity"), NULL)'
  set [Counter] as 'Filter(Descendants([FactIssue].CurrentMember, 0.0,
      SELF_AND_AFTER), ((((([Measures].[Activity], [LastChangedDate].[All date
      ].[2010]) = 10.0) OR (([Measures].[Activity], [LastChangedDate].[All date
      ].[2010]) = 40.0)) OR (([Measures].[Activity], [LastChangedDate].[All date
      ].[2011]) = 10.0)) OR (([Measures].[Activity], [LastChangedDate].[All date
      ].[2011]) = 40.0)))'
  member [Status].[Counter] as 'Aggregate([Counter])'
  member [Measures].[AvgTransition] as '(([Measures].[SumTransitionTime], [
      Activity].[All activities].[verified], [Status].[Counter]) / [Measures].[
      IssuesStatesCount])'
Select NON EMPTY Crossjoin({[LastChangedDate].[All date].[2010], [LastChangedDate
    ].[All date].[2011]}, {[Measures].[IssuesStatesCount], [Measures].[
    MinTransitionTimeInDays], [Measures].[MaxTransitionTimeInDays], [Measures].[
    AvgTransition]}) ON COLUMNS,
  NON EMPTY Crossjoin([Product].[All product].Children, {[Owner].[All person]}) ON
      ROWS
From [SoftCockpitCube]
Where ([Activity].[All activities].[verified], [Status].[Counter])
```

Listing 6.9 shows a complex parent child query. It checks for all issues which start in the state New and stops in the state *Verified*. The query does not check for circles in the process. If an issue is several times in the *Verified* state, only the last change to *Verified* is observed as last state.
At the beginning a member is defined, for the attribute activity in the *Facts* dimension and stored in the measures dimension. The measure dimension is usually intended for calculations, but is easy to handle and used as storage dimension. The attribute is the same as in the dimension activity. Since a dimension can only be used once, the definition of this member as attribute of the *Fact* dimension is used to filter a second time, at the beginning and at the end of the parent-child chain. The set *Counter* is defined, this set contains all descendants of the current activities starting from the activity new or created, by using the parent-child function *Descendants*, with a time restriction. All members are facts from the fact dimension and represent state changes. Only state changes have the parent-child relationship implemented. The numbers 10.0 and 40.0 are the ids of the *New* and *Created* activity in the activity dimension of the database. The only way to access the attribute in this dimension, is by using its value. In the installation the states are added manually and the attribute has always the same value. If the insertion of the states in the installation would change, this query has to be adopted.
The member represents all state change facts and their descendants, where

the activities *New* and *Created* lead into. This includes the *New* state, but also changes to later states like *Assigned* or *Verified*. In the *Where* clause the result is filtered for the activity *Verified*, this is done to filter for the final state. The final result contains now all facts, which lead to the *Verified* state and have a fact in their parent-child relation, with a state *New*. The *Select* defines, which way the result is displayed. It is important to know that this query takes a lot of time, if many facts are included in the query the time restrictions, like the number of months that are investigated, have to be as strict as possible.

6.3 Parametrization of queries

In the Softnet Cockpit MDX queries can be executed by typing the queries in the MDX query editor and execute them, or by executing predefined queries, that are saved in query files. The predefined queries are executed by calling the web-page that includes the JPivot interface to the Mondrian engine. Parameter in the HTML link define, which predefined query should be executed. It is also possible to parametrize elements inside of the queries. This gives the Softnet Cockpit more flexibility. For example if a user wants to execute one query for one special hierarchy of one dimension he can call the parametrized query with this hierarchy and does not have to call the general query and drill down to the expected hierarchy. The first part of this section describes how to access one query, the second part how the parametrized elements inside of this query are defined.

6.3.1 Parameter in a HTML link

The HTML links to the web-enabled dashboard exists of several parameter they are explained in this section.
An example of a link to a full parametrized query:

```
serverlocation/testpage.jsp? query=0\_1\_SnapshotStatusParamObject\&param1=object
    \&paramProduct=[Version].[All\%20Version]\&paramStatus=[Status].[All\%20Status
    ]\&paramObjectType=CRQ\&paramDate1=[CreatedDate].[Day].[2008-11-11]\&
    paramDate2=[CreatedDate].[Day].[2009-11-11]\&paramDate3=[CreatedDate].[Day
    ].[2010-11-11]\&paramDate4=[CreatedDate].[Day].[2011-11-11]\& paramDate5=[
    CreatedDate].[Day].[2012-11-11]
```

Parameter query

The Softnet Cockpit contains many different query files to structure the different queries. The value of this parameter is the name of the query file, where the query is stored without the .jsp suffix. Each HTML link has to have this parameter. Files with the ending Param in their name contain full parametrized queries, other files contain no parametrization inside the queries.
Examples are: query= 1_3_NumberOfForwarded or
query= 1_4_NumberOfLeftNewParam

Parameter param1

The *param1* parameter is used if more than one query is stored in one file, to define the query that should be executed. Queries can not be parametrized between different dimensions like *Product*, *Status* or *Milestone*. This requires one query for each dimension or combination of dimensions. All queries, that display the same result for different dimensions, are stored in the same file with different names for these dimensions. If the file has queries without name or same name the last of those queries is executed. If the parameter has no corresponding query name in the file the last query in the file will be executed.
Examples are: param1=Product, param1=Milestone, param1=issueType

Parameter param(Hierarchy)

The value of this parameter is a dimension, with all hierarchies in this dimension defined. Only the queries that have parametrized dimensions, defined inside, have these parameters. They have one parameter for each parametrized dimensions. Every query can have several parametrized values inside its statement. The goal of this parametrization is to access one specific hierarchy level or element in one hierarchy level of one dimension in the query. The user does not have to drill through all levels to one specific element and can share or save the link to this level or element.

The value of this parameter has to be the exact value of the last level as a string value, or the whole hierarchy. If a level has more than one child or a level other than the last one should be shown, the exact level hierarchy, including sharp brackets for each level, has to be added. A generic example would be: [TopLevel].[n MiddleLevel].[Value] , [TopLevel].[MiddleLevel] or just the value without brackets. For example the level date has to have always the whole level path, due to its three possibilities in the middle level (day, month and year). The value element, which is always the last element, is one specific element or the *All Value*. In case of the *All Value*, all children of the top level will be displayed. The *All Value* consists of the string *All* one empty space and the name of the hierarchy. In case of a person hierarchy it is always *All person*. The specification of the *All Value*, the hierarchies and the levels are in the definition of the cube, the *SoftcockpitCube.xml* file. In some new queries the parametrized element inside the query contains additionally the children function; *.Children*. In this case only elements of the level below the given one are printed. In the query this is used to eliminate the additional click to the hierarchy and produce a better diagram, without the sum of this level at the beginning.
There exist a few problems. If there are more dimensions and views which point to the same table, it is possible that the use of the value only causes a switch to the wrong view. It is advisable to use always the full path like [TopLevel].[n MiddleLevel].[Value]. Those dimensions are the hierarchies which have also a previous table like *Status* or *Product*, or the *Person* hierarchies, which occur as *Owner*, *Changer* and *Reporter*. Examples for different possibilities of this parameter are: paramIssueType=Defect, paramSeverity=D, paramOwner=[Owner].[All Person], paramDate=[CreatedDate].[day].[2012-10-29], [CreatedDate].[week].[2012-W51], [CreatedDate].[month].[2012-Q4-M10], [CreatedDate].[quarter].[2012-Q4], [CreatedDate].[year].[2012], paramStatus=[Status].[All Status].[assigned]

Parameter reallyload

The parameter *reallyload* can have the values 0 or 1. This parameter is optional. It can load a pre-saved query. The parameter query has to be set correctly. If the button save in the program is pressed, the values of a query will be stored in a special XML save file on the server. Only one save-file can exist for each query file. The query will be loaded if such an XML file exists and the query is called with the *reallyload* parameter set to the value 1.

Parameter of the diagram

The attributes of the diagram can be set in four parameters. All parameter are optionally. If one or more of the parameters are not set the the value of this parameter is the default value defined in the attributes file of the diagram.
The parameter *diagram* can have the values true or false and defines if the chart is visible or invisible.
The parameters *diagramWidth*, *diagramHeight* define the width and the height of the chart as integer values.
The parameter *diagramType* defines the type of the chart, like lines or bars in 3D. The value of *diagramType* is an number between 1 and 16. The mapping from this value to the concrete chart is defined in *chartpropertiesform.xml*. They occur also in the same order in the options menu of the dashboard in the chart options. If necessary all other parameters of the chart can also be parametrized.

6.3.2 Parametrization inside the queries and the code

This section explains, how the parameter are parsed and handled inside of the code in the jsp files or inside of the query files.

Page for executing queries

The file *Testpage.jsp* contains the main functionality of the dashboard and the links to the JPivot and the Mondrian libraries. This file loads and executes the query files. The Mondrian tool resolves and executes the queries JPivot is the interface to Mondrian and displays the results as sheets and diagrams. The default diagram values are interpreted and set to their attribute variables. If the *reallyload* parameter is set, also the XML file including the stored values of a query is loaded.

Query files

All query files are located in the folder *apache-tomcatwebappsSoftCockpit-MondrianWEB-INFqueries*. These files contain the queries. Each file can contain one or more queries. Each query is included in the jpivot tag `<jp:mondrianQuery id="" queryName="" jdbcDriver="" jdbcUrl="" catalogUri="">`

ID attribute

The ID attribute of the Mondrian query has to have the same name as the one defined in the testpage.jsp and in all other queries. If several testpages are used each page can handle one id and can only execute queries with exactly this id. More testpages can be used as a workaround to handle more than one query at a given time.

QueryName attribute

The attribute *queryName* should be unique for all queries in one single file. If the file contains only one query, this attribute is optional and not necessary. If the *queryName* attribute is missing or more than one query has the same name, always the last query in the file will be executed.

JdbcDriver, JdbcUrl attributes

The attributes *JdbcDriver* and *JdbcUrl* point to the data-warehouse database.

CatalogUri attribute

The attribute *catalogUri* contains the link to the definition of the cube.

Selection of one query in a query file

The tag `<jp:chooseQuery id="" queryName="">` defines which query in the file will be executed, if there are more than one queries. In this case *queryName* takes the param1 parameter of the HTML-link. This way the HTML-link is connected to the queries and the correct query will be executed.

QueryName attribute:

The *queryName* attribute defines the query, which will be executed and corresponds to the *queryName* attribute in the `<jp:mondrianQuery>` query parts.

Id attribute:

The id attribute has to have the same name as the one in the *testpage.jsp* file.

Defining a parameter for a query

The `<jp:setParam query="" httpParam="" mdxParam="">` tag defines the a parameter for the use inside of a query. It has to be opened before the `<jp:mondrianQuery>` tag and closed after `< jp:mondrianQuery>` tag with `<jp:setParam>`.

Query attribute

The query attribute has to have as value the *queryName* of the corresponding query.

HTTPParam attribute

The *HTTPParam* attribute has to have the same value than the corresponding one in the HTML link.

MDXParam attribute

The *mdxParam* attribute is the name of the parameter that will be used inside of the MDX query. In the query the *Parameter* function calls the parameter. This function can only occur once for each parameter and query, multiple calls for one predefined parameter will lead to an error. Defining more than one parameter for one level or hierarchy is possible. Each parameter has to be defined outside and used inside of the query.

Parameter function

The function to call the parameter inside of the query is *Parameter(paramMDX, Level, DefaultValue, description)*. Jpivot replaces this function with the correct statement before executing the whole query with Mondrian.

ParamMdx attribute

The attribute *ParamMdx* is the name of the parameter in the set statement defined outside of the query.

Level attribute

The *level* attribute is one specific level in the cube definition. An example would be [CreatedDate].[Year], it is the *Year* level of the *CreatedDate* time dimension.

DefaultValue attribute

The *DefaultValue* should be set if there is no value or a wrong value. Nevertheless it is not good to trust the default value, because the query can lead to an error if wrong or missing arguments are entered.

Description attribute

Description is just a description of the parameter and is not used later anymore and can not be accessed.

The following chapter presents a case study done with the results of the queries. The results are analysed and compared with the model of the issue tracking process.

7 Case study about the issue management process in the Softnet Cockpit

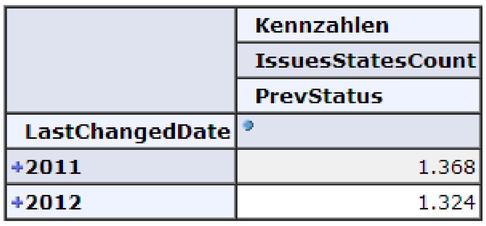
This chapter is a case study about the issue management process of the partner company of this master thesis. The issue management process is analyzed according to the process model described in Chapter 5. The Software Cockpit including the queries and the imported data is used to analyze the states in the process model. The number of issues and the behavior of the issues between the states is analyzed, by using the snapshot queries and state queries, described in Chapter 6. The run-time of the issues and their average time in the states is not analyzed and can be part of further case studies. The analysis of the process model is the start point of a continuous monitoring and further improvement of the issue tracking process. The data was also presented to the partner company and interpreted by the partner company.

7.1 Analysis of the data in the issue process of the partner company

In the following section several queries are run on the Softnet Cockpit with the changes and the new implemented queries. The goal was to evaluate the queries and compare the process model to the real behavior of the history events. The number of issues in the states and the history changes are analyzed. Due to the huge amount of data only one specific product as example was investigated in the years 2011 and 2012. The investigated repository contains around 45000 issues and has between 10000 and 18000 issues in each product between the years 1999 until 2012. The chosen product has around 18000 issues. The number of history events of all activities in this project is around 250000. The number of history changes for the chosen product is around 90000. The analysis is only done for the number of issues in a state or number of history changes. The duration an issue stays in one state or several states is not investigated in this case study.

Creation

At the beginning the number of issues that were created for the chosen product are investigated. Figure 7.1 shows the number of history events that created an issue in the years 2011 and 2012. In the year 2011 1368 new issues were created. This number falls to 1324 in the year 2012. The *Created* activity is not a state change activity. There exists also a history event for issues that move to the state *New* from an empty state as state change history event, but only few issues use this event. Therefor new created issues are put automatically in *New* and the state change history events to the state *New* are only used to identify issues that re-enter the state *New*. An issue can only be created once, this way the number of history events with the *Created* activity is also the number of created issues.
Figure 7.2 shows the number of existing issues in the state *New* in the year 2012. This is calculated by using the snapshot query. The upper graph shows all issues in the state *New*, the lower graph shows only the number of issues in the state *New*, that do not have a milestone assigned. The number of around 12000 issues in the state *New* is very high in relation to the number of overall 45000 issues that exist and the number of 1324 issues that were created in the year 2012. The number issues without an assigned milestone is much lower around 3000, but still more than twice the number of created issues in this year. Both graphs behave same and raise slightly with a drop down in the middle of the year. This shows that over the year the number of issues that are created and that leave this state is equal. The reason for the high number of issues in this state that have an assigned milestone, which intended in the issue process, is given by the partner company in Section 7.2.



	Kennzahlen
	IssuesStatesCount
	PrevStatus
LastChangedDate	
+2011	1.368
+2012	1.324

Figure 7.1: Number of events with the *Created* activity issue in 2011 and 2012

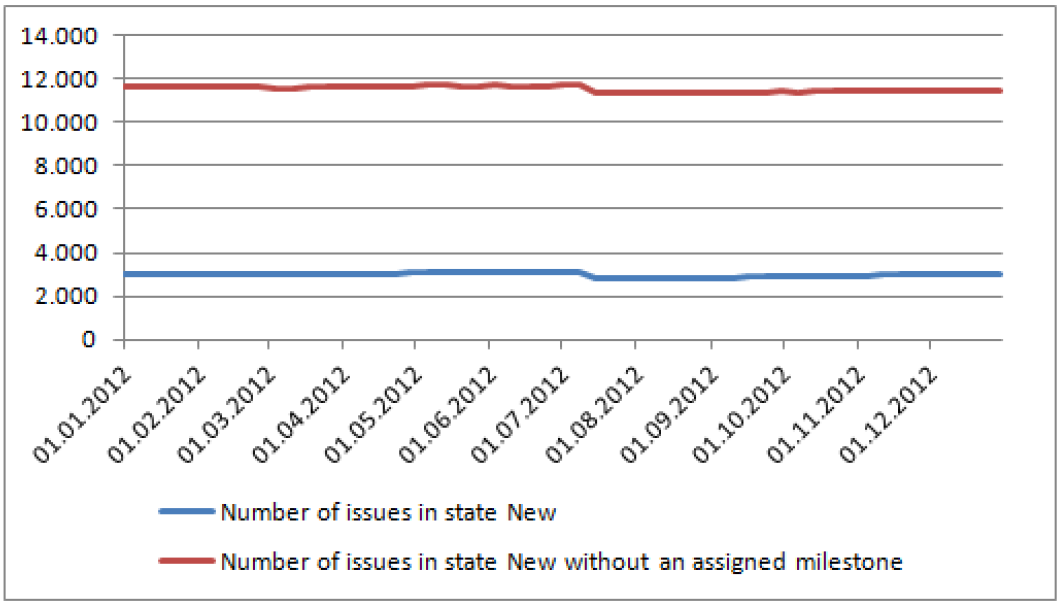


Figure 7.2: Number of issues in the state *New* in the year 2012

Checking and Checked

The number of issues in the state *Checking* is very low at the beginning of the year 2012, the number is at 77 and increases until the middle of 2012 and decreases to 33 at the end of 2012. The number of issues in the state *Checked* is at 23 at the beginning of 2012 and increases steady to 53 at the end of 2012. The number of issues in both states and the number of history events with an activity of one of those states is very low, those states are only used rarely.

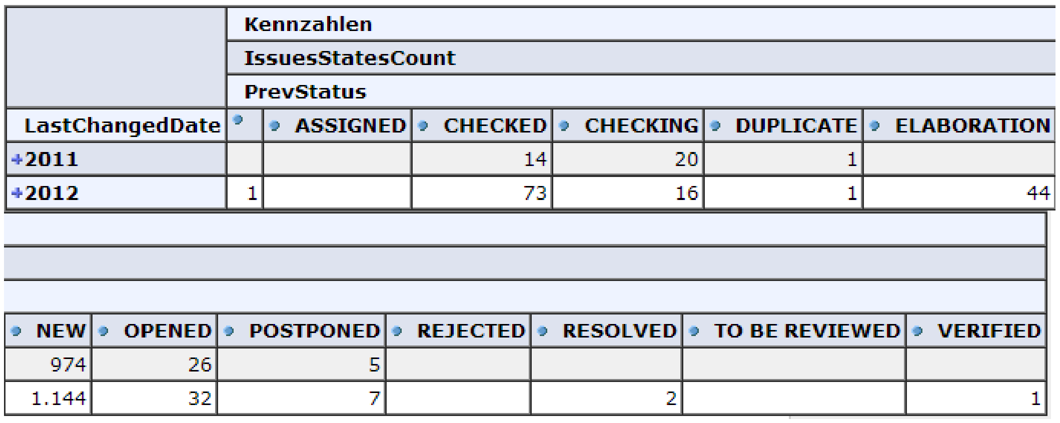
Postponed

From the beginning to the middle of 2012 the number of issues in the state *Postponed* is constant around 100, than it jumps down to 60 stays at this number until the end of 2012. The number of history events that lead to and from this state is very low and only high in the middle of 2012, when the number of issues in this state jumps from 100 to 60 issues.

Assigned

In the state *Assigned* the issue is already assigned to a developer and attributes of the issue like the milestone, product, or severity are already set. The issue is ready to get implemented. Figure 7.3 shows the history events, which lead to the state *Assigned*, splinted to their previous state. Nearly all issues enter the state *Assigned* from the state *New*, a few issues have been in the state *Checked* or *Checking* before. A couple of issues are moved back from *Opened* to *Assigned*. This behavior is not explicitly modeled in the process, but could state a re-assignment or that an issue is moved into the backlog of a developer due to lack of time. Remarkable is also, that in the year 2012 44 issues were moved from *Elaboration* to *Assigned*, but none in 2011.
By comparing the numbers of created issues with the number of history changes that lead from a earlier state to *Assigned* it can be identified, that more than 90 percent of the issues, which have been created in 2012, have become assigned. In 2011 75 percent have become assigned after their creation.
The diagram of the snapshot query in Figure 7.4 shows, that the number of issues in the state *Assigned* stays constant or is even decreasing over the year. This means that issues are not only constantly created in the year 2012, but also assigned to a project and developer. The number of issues in this state is quite high between 300 and 400 compared to the overall created issues in the year 2012.

The number of history events with an *Re-Assignment* activity is shown in Figure 7.5. If a *Re-Assignment* activity happens, in most cases it happens in the *Assigned* state, sometimes in *Checked*, *New* or *Opened*, but never in the final states. The number of *Re-Assigned* events in the state *Opened* is nearly



	Kennzahlen												
	IssuesStatesCount												
	PrevStatus												
LastChangedDate		ASSIGNED	CHECKED	CHECKING	DUPLICATE	ELABORATION	NEW	OPENED	POSTPONED	REJECTED	RESOLVED	TO BE REVIEWED	VERIFIED
+2011			14	20	1		974	26	5				
+2012	1		73	16	1	44	1.144	32	7		2		1

Figure 7.3: Number of state events to the state *Assigned* in 2011 and 2012, splinted to their previous state

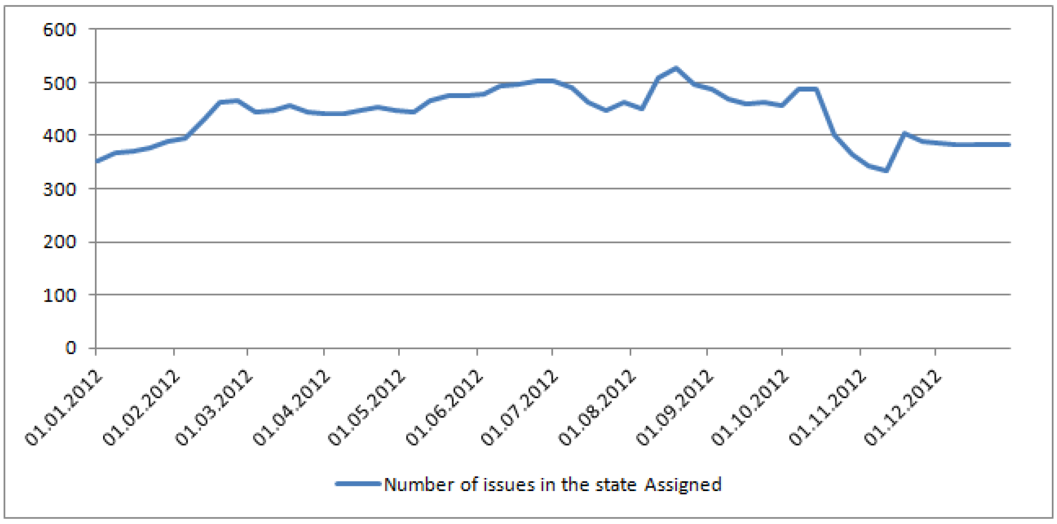
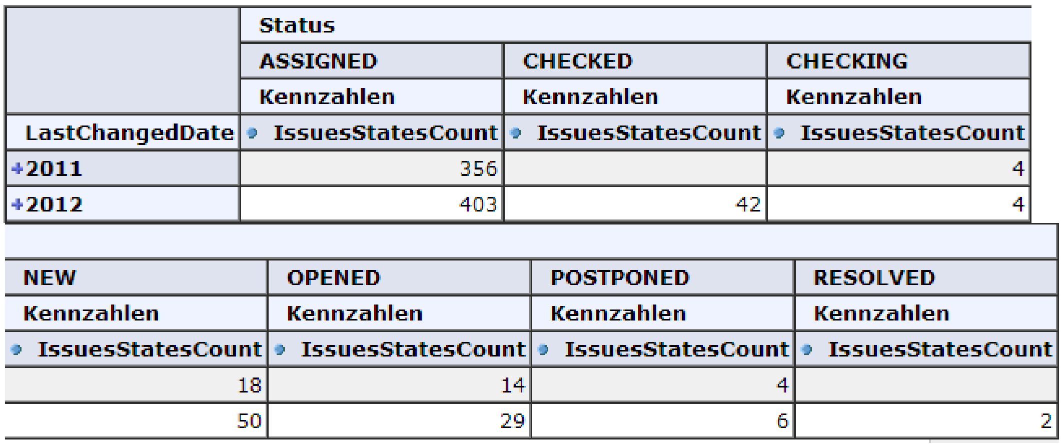


Figure 7.4: Number of issues in the state *Assigned* over the year 2012

equal to the number of state changes from *Opened* to *Assigned*. It can be assumed that those are the same issues and that this behavior is intended, since the new developer has to analyze and open the issue by himself. The issues with a *Re-Assignment* activity in the state *New* have probably already been in *Assigned*, but have been moved back to *New*.



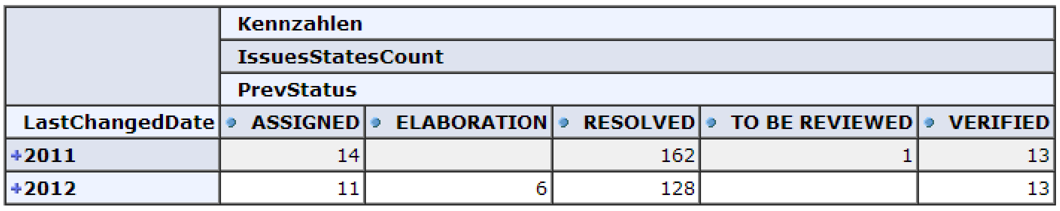
	Status						
	ASSIGNED	CHECKED	CHECKING	NEW	OPENED	POSTPONED	RESOLVED
	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen
LastChangedDate	● IssuesStatesCount	● IssuesStatesCount	● IssuesStatesCount	● IssuesStatesCount	● IssuesStatesCount	● IssuesStatesCount	● IssuesStatesCount
+2011	356		4	18	14	4	
+2012	403	42	4	50	29	6	2

Figure 7.5: Number of events with a *Re-Assignment* activity in 2011 and 2012, and the state where the *Re-Assignment* happened

Opened

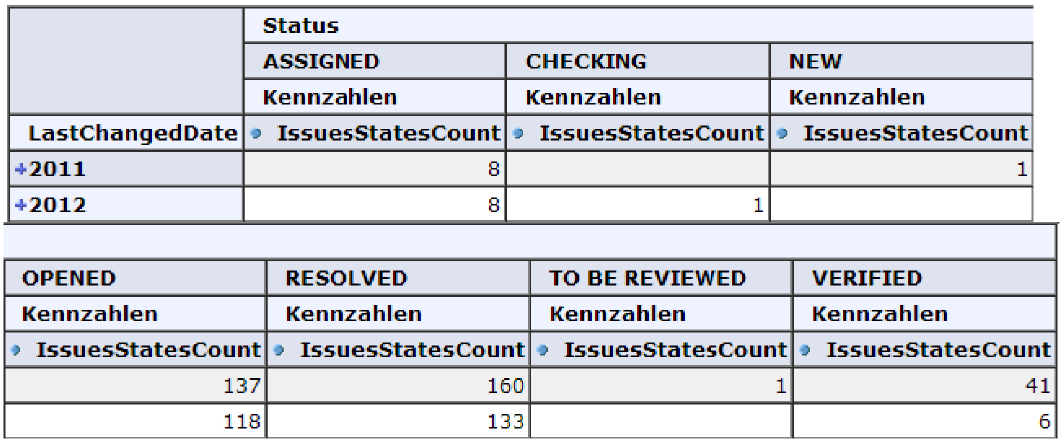
Figure 7.6 shows the history events, that lead to the state *Opened*. All events with previous state *Assigned* are regular *Opened*. Only very few issues use this activity. The major number of issues use the state *Opened* only as *Re-Opened*. A few issues became *Re-Opened* after they were already in the state *Verified*.

The numbers in Figure 7.7 show the result of a query that calculates the number of history events with the test-failed activity. Events with the test-failed activity happen independently from state change events, but should occur together with a state change event form *Resolved* to *Opened*. The number of history changes from the states *Resolved* to *Opened* is nearly equal to the number of failed tests, some test-fails occur in the state *Opened*. This means the event with the activity *Test-failed* together *Re-Opened* state transition is used as intended in the process model. The state *Opened* for the purpose of handling issues that need more development time is only used rare.



	Kennzahlen				
	IssuesStatesCount				
	PrevStatus				
LastChangedDate	ASSIGNED	ELABORATION	RESOLVED	TO BE REVIEWED	VERIFIED
2011	14		162	1	13
2012	11	6	128		13

Figure 7.6: Number of state events to the state *Opened* in 2011 and 2012, splinted to their previous state



	Status						
	ASSIGNED	CHECKING	NEW	OPENED	RESOLVED	TO BE REVIEWED	VERIFIED
	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen	Kennzahlen
LastChangedDate	IssuesStatesCount	IssuesStatesCount	IssuesStatesCount	IssuesStatesCount	IssuesStatesCount	IssuesStatesCount	IssuesStatesCount
2011	8		1	137	160	1	41
2012	8	1		118	133		6

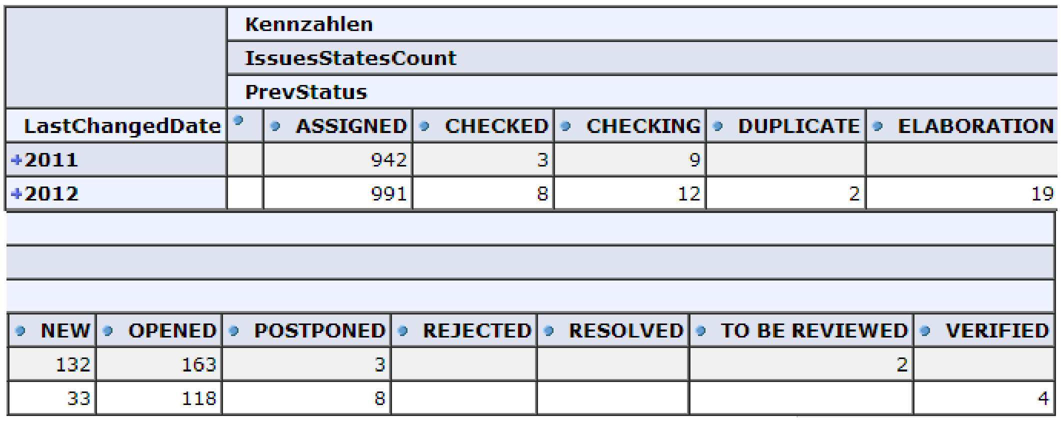
Figure 7.7: Number of events with a *Test-failed* activity in 2011 and 2012, and the state where the test fail happened

Resolved

An issue in the state *Resolved* is already implemented or solved and ready to get tested. Figure 7.8 shows the number of history events into the state *Resolved* splinted to their previous state. Most state changes happened from the state *Assigned* to *Resolved*, a few occurred from *New* or *Opened* to *Resolved* and nearly no changes from other states. Changes from *New* to *Resolved* indicate issues with a very low or no development time, or issues which have already been solved, but need to be tested. The sum of all changes to *Resolved* in 2012 was 1195, which is around 100 issues lower as the number of created issues in this year. The sum of test fails in this state divided through this number shows that nearly 11 percent of the events that lead to

Resolved had a test-failed activity.

The graph in Figure 7.9 shows the number of issues in the state *Resolved* over the year 2012. At the beginning of the year the number of issues in *Resolved* is very high. More than 2000 issues in this state are more than 200 percent of the number of issues created in this year. This seems to be not intended in the process model, since *Resolved* is not a final state. At the middle of 2012 the number of issues in the state *Resolved* decreased tremendously in one week. After that the graph is nearly constant at a level around 350 issues.



	Kennzahlen												
	IssuesStatesCount												
	PrevStatus												
LastChangedDate		ASSIGNED	CHECKED	CHECKING	DUPLICATE	ELABORATION	NEW	OPENED	POSTPONED	REJECTED	RESOLVED	TO BE REVIEWED	VERIFIED
+2011		942	3	9			132	163	3			2	
+2012		991	8	12	2	19	33	118	8				4

Figure 7.8: Number of state events to the state *Resolved* in 2011 and 2012, splinted to their previous state

Duplicate, Rejected and Elaboration

The states *Duplicate*, *Rejected* and *Elaboration* are final states, that are used, if an issue is not going to be resolved. The number of issues in these states is quite low compared to the number of issues in the state *Verified*. The final state *Elaboration* is not used in the year 2012 and there is also only one issue in this state, from a previous year. The number of issues in the state *Rejected* is at 159 at the beginning of 2012, it raises constantly and has a jump in the middle of 2012 from 240 to 340 issues. Afterward the number of issues in this state raise constantly again. The number of issues in the state *Duplicate* raises constant from 215 to 255 during the year 2012.

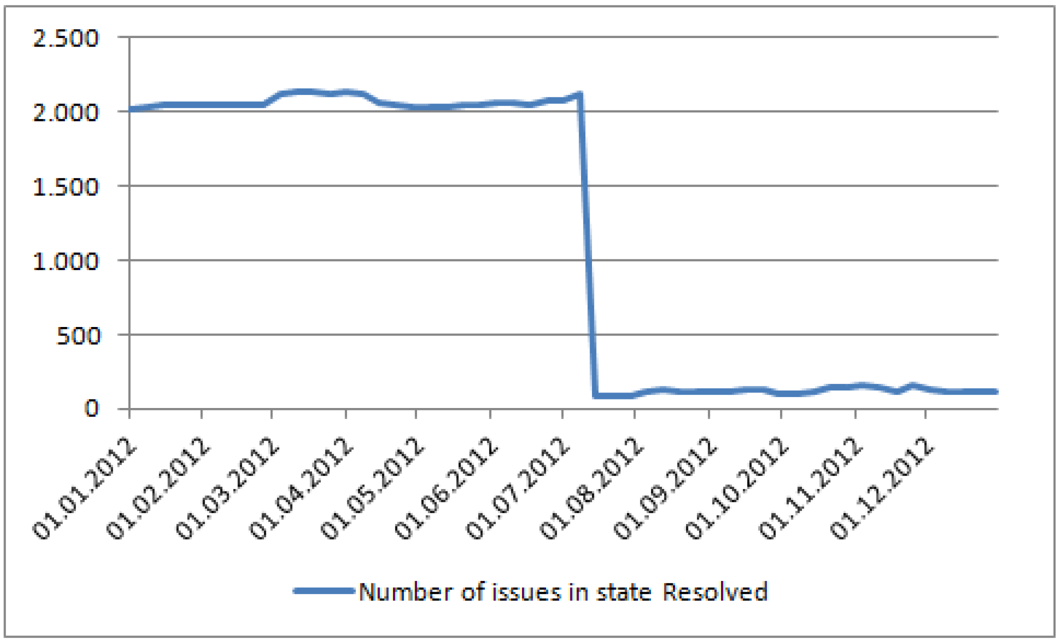
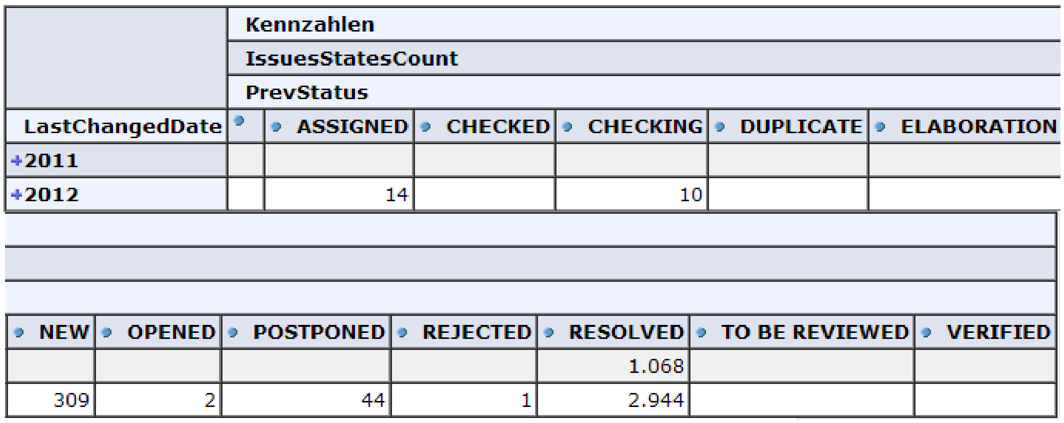


Figure 7.9: Number of issues in the state *Resolved* over the year 2012

Verified

Verified is a final state that indicates that a issue is resolved and tested. A issue should never leave this state again. Figure 7.11 shows number of history events to the status *Verified* splinted to their previous states. Changes from any other state than *Resolved* to *Verified* are issues that have already been solved in the past, like through another issue, or did not use the *Resolved* state correctly. The number of issues that are in the state *Verified* and did not use the *Resolved* state is low, in the year 2012 there were 2944 state changes from *Resolved* to *Verified*, this means 88 percent of the state changes to *Verified* are from the state *Resolved*. Most other state changes are from *New* to *Verified*.
The snapshot graph in Figure 7.10 shows the number of issues in the state *Verified* in the year 2012. The graph raises constantly with one exception where the graph jumps up from from 6000 issues to over 8000 issues in this state. This jump is the opposite behavior from the state *Resolved*, where the graph jumps down.



	Kennzahlen												
	IssuesStatesCount												
	PrevStatus												
LastChangedDate		ASSIGNED	CHECKED	CHECKING	DUPLICATE	ELABORATION	NEW	OPENED	POSTPONED	REJECTED	RESOLVED	TO BE REVIEWED	VERIFIED
+2011											1.068		
+2012		14		10			309	2	44	1	2.944		

Figure 7.10: Number of state events to the state *Verified* in 2011 and 2012, splinted to their previous state

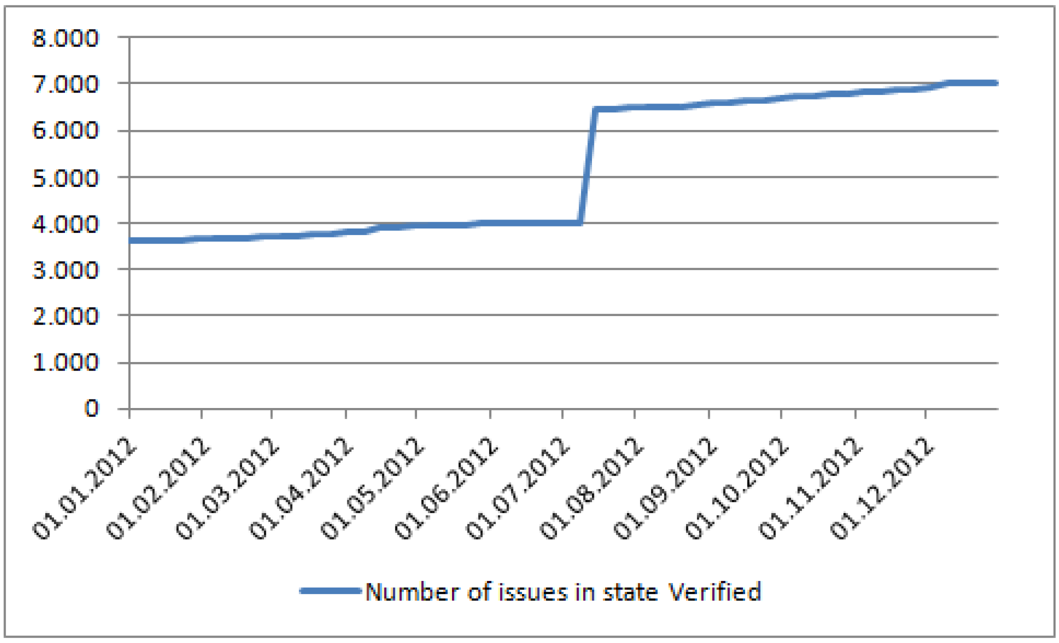


Figure 7.11: Number of issues in the state *Verified* over the year 2012

7.2 Statement of the partner company

The results of the case study were presented to the partner company, to verify the results of the queries. They said that the issue process is still under development and will be improved. At the creation of an issue it is put into the state *New* and should not have a defined milestone. Issues get the information about a milestone in the assignment phase. Most issues, which have a milestone, are rather old and have maybe been reassigned and put to the state *New* later. It is also possible that these states are duplicates, or are obsolete due to another reason. Those issues should be analyzed and put into a final state. Since this process needs much effort and has no priority this process is done very slowly. The lower graph in the diagram 7.1 represents the issues in *New* with an undefined milestone, they are the current issues and should get assigned. Nevertheless since in the year 2012 about 1300 issues were created and 3000 issues without an assigned milestone are in this state this number should decrease and not increase. But a very slow increase is still acceptable.
They explained, that the difference between the states *Assigned* and *Opened* is, that the management wants to check which tasks are right now under development and which tasks are only in the backlog and assigned to a specific developer. A developer can have many assigned issues, but should only have an issue in the state Opened, if it is right now under development. But unfortunately the developer do not accept this state yet and do not use it in the implementation of an issue. They said that, the assignment and the implementation of the issues in this part of the issue management process should be improved. They also try to reduce the number of issues in the state *Assigned*, since quotation of created issues to assigned issues should be better over one year. In the *Assigned* state most *Re-Assignment* activities should happen.
They said that the huge jump in the snapshot graphs in the *Resolved* and *Verify* states is not on purpose. This huge number of state changes in one week occurred due to a cleanup of the database in July 2012. Such cleanups should only occur rarely, but can happen. A solution should be found how to avoid the big influence of this cleanup on the Softnet Cockpit output. The partner company proposed, that a possible solution could be a new separated final state or a flag. A MDX query solution would be to use

the *CreatedDate* dimension, as time dimension in the snapshot query and set the beginning date to a younger date, this way very old issues could be eliminated from the issue. This query would not show a real snapshot anymore.

7.3 Conclusion

The case study proves that in most cases the issue process model is implemented correctly in the partner company. The assignment of the issues stays constant over the year and the assignments correspond to the number of new created issues. Also the number of resolved and verified issues conforms to those numbers and proves that the issue tracking model is used correctly. The step of sending issues back to the state *Opened*, if tests or reviews fail works very well. In its normal purpose to contain issues that need more time to get implemented, the *Opened* state is used too seldom. Also the states *Checked* and *Checking* are rarely used. Solutions have to be found that those states are better accepted by the developers, or the process should be adopted. The standard process in the Bugzilla issue tracker for example does not have an *Opened* state and only an *Assigned* state (*Bugzilla process @ONLINE* 2013).
In the year 2012 a correction of the data in the states *Resolved* and *Verified* happened. This had a big influence on the numbers and queries in the Softnet Cockpit. A solution should be found, how to mark the changes and ignore them in the queries. Nevertheless such changes should not occur often and show that the process needs to be improved. Issues should never stay too long in one of the earlier states, which rises the probability that such issues stay in those states too long.
There is a huge backlog of old issues in the state *New*. These old issues should be cleaned up like already done if possible.
Also the number of reassignments is quite high compared to the number of assigns only. The assignment of issues to developer takes much time and the number of issues should be reduced.

8 Related Work

This chapter provides information about projects, tools and scientific papers, which are related to this master project. It is splinted into a section, in which process monitoring approaches are investigated and a section in which similar cockpit projects are compared with the Softnet Cockpit, that is used in this master thesis.

8.1 Process Monitoring

A couple of work is already done about the monitoring of software processes. (Feiler and Humphrey, 1993) describes a general approach about the monitoring of software processes. A separated observer is used, that monitors the enhance state of the observed process. This agent can be implemented by the observed process or by a separate process. The goal is to get the gathered data about the process and its performance. Furthermore the data is used in a process measurement database for the process planning, analysis, control and adjustment. It is also possible to use a system of several component agents, which are connected to several condition monitor agents that observe a process component system, that consists of several processes (Bunch et al., 2004). The goal of the project in this master thesis is also to monitor a process in the software development. The differences are, that the Softnet Cockpit uses only the history data of a software process and evaluates this data in a OLAP cockpit and does not directly implemented agents in the software process.

(Zur Muehlen and Rosemann, 2000) explains process monitoring as a part of the process change management. The prior steps in the process change management are prioritization of relevant processes, process modeling, process analysis, process optimization, process implementation and process realization. The result of the process monitoring can trigger that those steps have to be redone. This way the process change management can be seen as circle. There are three sights on the process monitoring: the process view, the resource view and the object view. In the process view all key performance indicators, which are related to the business process are presented. Examples can be the average, maximum and minimum process time, the average, maximum and minimum costs of the execution of one process, and the quality of the process expressed in the number of failures or loops as an indicator for necessary rework. If a report of this view is produced, it can contain one specific instance of a process model or an aggregation of several process instances. The resource view concentrates on the costs of a process. A very efficient process can also be not successful if the costs of using it are too high. The data for the resource monitoring can be from the asset management accounting. Costs which can not be connected to one process have to be splinted partly to one process. The object view displays the attributes of all objects that are connected to the process. It explains the difference between an object and a process, that a processes can be defined as the logical sequence of functions necessary to process a business relevant object (Zur Muehlen and Rosemann, 2000). This project is more business oriented than the work, done in this master thesis. But the parts of the process monitoring explained in this paper can be connected to this master thesis. The metrics and the KPIs are part of the process view, the costs of processes were not investigated in this project, the object view can be found in the dimensional presentation of the process in the Softnet Cockpit.

(Van der Aalst, 2011) describes a way of investigating processes, instead of only monitoring a process his main goal is the mining of processes. He shows how to collect the data to mine a software process. The main data are event logs. An event can have any kind of attributes like timestamps, the transactional information and the resource usage but is often only named after the activity. An event can be from any kind of resources like a code visioning system, an email client or a issue tracking system. (Poncin, Serebrenik, and Brand, 2011) try to apply the approach of mining business processes generally to mining for processes in software repositories. They describe the problem of extracting the logs from different sources, especially they investigate the source code repository CVS and the bug-tracker Bugzilla. By using those event logs produced by the tool FRASR. They create with the tools ProM Fuzzy Miner ticket trees from the open source project GCC and compare this model with the standard Bugzilla life cycle model. This master thesis had also the focus on evaluating a given process with event logs. The event logs as history events and the model as issue tracking graph was given by the investigated partner company. Instead of mining a process the process was monitored in this master thesis.

8.2 Projects

Several projects have been done in the field of Software Cockpits.

One of the firsts ideas to collect, every information about a software project and present them to the user was done by (Münch and Heidrich, 2004). The researches done in this field resulted in the start of the Soft-Pit project (Heidrich, Münch, and Wickenkamp, 2006). The goal of this project was the build a software project management control center. It should be similar to a cockpit in a airplane. This way they called it a Software Cockpit. This tool should be used in scientific projects and in small companies. The project was founded and implemented by several scientific and commercial partner. Since there was a focus on project data, this cockpit focuses on process data, but also product and resource data can be investigated. In the evaluation of the data process metrics and the most important process KPIs were implemented in this cockpit. This project is very similar to the Softnet Cockpit that was used in this master thesis, both cockpits monitor business processes in the field of software projects.

The two projects Q-Bench (*QBench @ONLINE* 2013) and sd&m (Bennicke et al., 2007) are also two public projects, with the goal to implement an integrated Software Cockpit. Both projects concentrate on the software application and more on product metrics. The goal of sd&m is to develop a cockpit that has several adapters, that can be connected to different projects, to store the data of several different projects in one data-warehouse and analyze it with integrated rules and metrics. Several quality models are used in this project to identify the metrics and rules. Q-Bench uses product metrics, heuristics and quality models to ensure the quality of applications in object oriented software projects. Both projects use also metrics and data from several sources in the software development, but focus more on the product instead of the processes, which are related to this project.

Business Process Cockpit (BPC) is a tool that supports real-time monitoring, analysis, management, and optimization of business processes. It is a big project with several partner, but the main partner is HP. This cockpit includes a ProcessDatawarehouse (PWD) that stores the data about a process and updates the data online. Queries, that are designed specially to observe and control processes can be executed. It gives also the possibility to influence the process and send events to the process to manage running systems. It can create reports by using commercial OLAP reporting tools like the CrystalReport tool-set. It is also able to communicate with process mining tools to analyze undocumented processes and improve the processes (Sayal et al., 2002). This project is very similar to the Softnet Cockpit, it focuses in processes and uses a special process oriented data-warehouse. Differences are that this project uses commercial software, the Softnet Cockpit uses only open source parts. The queries in the Softnet Cockpit are also more flexible and can be parametrized.

9 Conclusion and Future Work

This chapter presents a short summary of the chapters in this master thesis in the context of the goals. After that possible future work in the field of the Software Cockpit are presented. The last section provides a small conclusion.

9.1 Summary

This master thesis is written in the context of the implementation of the quality measurement of an issue tracking process in a automotive software development company, by using a software measurement and monitoring tool.

At the beginning the application life-cycle management is introduced. Software companies use ALM to control the work-flow of their products through all stages, from the first idea until the end of the product. Issue tracking is one important process in the ALM. It handles the enhancements and bugs that are connected to a software application. It is important that ALM processes, like this issue process runs steady through the whole lifetime of the product. Quality is an important aspect of this process. One important part of the quality engineering is the measurement of the quality. Quality models are to define abstract quality attributes. Metrics are one part of the software quality models. They are rules, that describe how to measure a software process or application. Software monitoring tools implement those metrics and are used for a continuous quality monitoring. A short market overview was given about the most important software monitoring tools.

In the practical part a Software Cockpit, a software application and process monitoring tool, was presented. It bases on the OLAP technology and uses open source tools. This part of the thesis includes also the changes, that were done to this Software Cockpit, according to the requirements and the data of a partner company. Those changes were done to the front-end as well as to the back-end and the database. Most changes that were done to the architecture were necessary, to implement the queries, that give the possibility to access the data in the Software Cockpit. The queries were implemented regarding to a set of metrics given by the partner company. The queries were grouped and explained. Example queries of each group were given, to allow a fast use and adoption of the queries. It was also described, how to parametrize the queries.

At the end a case study was done, which presented the behavior of the issues in the issue process of the partner company. The case study was done by using the data of the partner company in the Software Cockpit and using the process model. Beside of a few problems in the process the issues followed the process model correctly. The case study can be a good start point, for a continuous monitoring of this process in the future.

9.2 Future Work

This section describes future work that could be done to the Softnet Cockpit.

There are several metrics in the Sofnet Cockpit that correlate, but are called and calculated separately and have to be compared to each other. An example would be the test and review failed metrics correlate to the state change metric from the state *Resolved* to the state *Opened*. Another example would be the re-assignment metric of issues and the state change metric from the states *Opened* or *Resolved* to the state *Assigned*. Metrics could be developed, that use both aspects and compares them with each other to get a better overview over the process. Also problems between those correlated history events could be figured out earlier.

Queries should be implemented that can separate old issues and current issues. As showed in the case study in Chapter 7, there exists a problem with old data and old issues in the issue tracker and data data of clean-ups of the database. This influences the results of the queries a lot. Especially in the run-time measurement, old issues influence on the average calculated run-time a lot. The queries should be able to identify old issues and separate between old issues and new issues, without the need of cleaning the whole database.

In the metrics upper and lower boarders should be identified and included into the queries or the data-warehouse. The case study showed the behavior of the issues in the issue process and the partner company made a statement on how the behavior should be. A future work could be to implement upper and lower boarders into the metrics and the data-warehouse, according to the information of the case study and the statement. This way it could be easier identified, if the behavior of the issues in the process flow and in the states is correct, or if problems occurred.

The case study showed, that there is probably a high number of old duplicate issues in the state *New*. A future work could be to identify duplicates by using the information in the Software Cockpit and enhance the the Software Cockpit by attributes of the issues that makes it easier to identify duplicate issues. The information of the attributes in this Software Cockpit could be easily combined with technicians like the natural language processing (Runeson, Alexandersson, and Nyholm, 2007), to identify duplicates fast and avoid old duplicate issues in the current issue tracking work-flow.

9.3 Conclusion

In the complex field of quality engineering in the software development, this thesis has given one possibility to measure a software process. The project as part of this master thesis showed a way to monitor a issue tracking process. There are only few papers and projects in the field of process monitoring with process metrics in combination with a OLAP oriented Software Cockpit. This made it difficult to find a solution to implement the more complex queries, like the parent-child queries and parametrization of those queries. At the end it was possible to find a solution, and present this solution in this master thesis. Nearly all requirements, especially the necessary changes in the architecture and the queries, given by the partner company could be implemented in the Softnet Cockpit.

There are still some disadvantages, that were discovered in the run-time of this project. The user interface is rather old and does not offer new techniques like drag and drop or well designed reports. Also the OLAP engine takes much time to execute complicated queries and therefor the queries must be restricted to less dimensions, or the time over which a query is calculated has to be reduced. Another interface and a faster OLAP engine would improve the experience of this Softnet Cockpit much.

Nevertheless the final version of the Softnet Cockpit is huge help for the project management in the software development, in their goal to monitor this software process and improve its quality. Especially since quality in such processes is a very important factor, it is important to measure all attributes of a process online. This can offer the OLAP based Softnet Cockpit.

In the future this tool will be a good advantage, for this company, to monitor their issue process and especially the quality of their process.

References

Albrecht, Allan J. and John E Gaffney Jr (1983). "Software function, source lines of code, and development effort prediction: a software science validation." In: *Software Engineering, IEEE Transactions on* 6, pp. 639–648 (cit. on pp. 25, 46).

Basili, Victor R, Jens Heidrich, et al. (2007). "Bridging the Gap between Business Strategy and Software Development." In: *ICIS*, p. 25 (cit. on p. 27).

Basili, Victor R and Richard W Selby (1984). "Data collection and analysis in software research and management." In: *Proceedings of the American Statistical Association and Biomeasure Society* (cit. on p. 30).

Basili, Victor et al. (2007). "GQMˆ+ Strategies–Aligning Business Strategies with Software Measurement." In: *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*. IEEE, pp. 488–490 (cit. on p. 27).

Beer, Wolfgang (2009). "Gerhard Weiss, Gustav Pomberger Wolfgang Beer, Georg Buchgeher, Bernhard Dorninger, Josef Pichler, Herbert Prahofer, Rudolf Ramler, Fritz Stallinger, Rainer Weinreich." In: *Hagenberg Research*, p. 157 (cit. on p. 46).

Bennicke, Marcel et al. (2007). "Das sd&m Software Cockpit: Architektur und Erfahrungen." In: *GI Jahrestagung (2)*, pp. 254–260 (cit. on p. 106).

Bunch, Larry et al. (2004). "Software agents for process monitoring and notification." In: *Proceedings of the 2004 ACM symposium on Applied computing*. ACM, pp. 94–100 (cit. on p. 104).

Caldiera, Victor R Basili1 Gianluigi and H Dieter Rombach (1994). "The goal question metric approach." In: *Encyclopedia of software engineering* 2, pp. 528–532 (cit. on pp. 24, 29–31).

Chappell, David et al. (2008). "What is Application Lifecycle Management?" In: *White Paper, December* (cit. on pp. 10, 12, 13).

Chaudhuri, Surajit and Umeshwar Dayal (1997). "An overview of data warehousing and OLAP technology." In: *ACM Sigmod record* 26.1, pp. 65–74 (cit. on p. 5).

Committee, Software & Systems Engineering Standards et al. (1998). "IEEE Std 1061-1998—IEEE standard for a software quality metrics methodology." In: *IEEE Computer Society, Tech. Rep* (cit. on p. 25).

Datta, Anindya and Helen Thomas (1999). "The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses." In: *Decision Support Systems* 27.3, pp. 289–301 (cit. on pp. 50, 51).

Deissenboeck, Florian et al. (2009). "Software quality models: Purposes, usage scenarios and requirements." In: *Software Quality, 2009. WOSQ'09. ICSE Workshop on*. IEEE, pp. 9–14 (cit. on pp. 28, 29).

Doran, George T (1981). "There's a SMART way to write management's goals and objectives." In: *Management Review* 70.11, pp. 35–36 (cit. on p. 32).

Feiler, Peter H and Watts S Humphrey (1993). "Software process development and enactment: Concepts and definitions." In: *Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*. IEEE, pp. 28–40 (cit. on p. 104).

Goth, Greg (2009). "Agile tool market growing with the philosophy." In: *Software, IEEE* 26.2, pp. 88–91 (cit. on p. 16).

Heidrich, Jens, Jürgen Münch, and Axel Wickenkamp (2006). "Zielorientierte Nutzung von Projektleitständen." In: *GI Jahrestagung (2)*, pp. 87–94 (cit. on p. 106).

"IEEE Standard Classification for Software Anomalies" (2010). In: *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, pp. C1–15. DOI: 10.1109/IEEESTD.2010.5399061 (cit. on p. 17).

Koziolek, Heiko (2008). "Goal, question, metric." In: *Dependability metrics*. Springer, pp. 39–42 (cit. on p. 32).

Loper, Stefanie and Gabriele Schmidt (2005). "Softwareprozessmetriken und agile Methoden." In: (cit. on p. 27).

McCabe, Thomas J. (1976). "A complexity measure." In: *Software Engineering, IEEE Transactions on* 4, pp. 308–320 (cit. on p. 26).

McCall, Jim A, Paul K Richards, and Gene F Walters (1977). *Factors in software quality*. General Electric, National Technical Information Service. (cit. on pp. 32, 33).

McDonald, Kevin et al. (2002). *Mastering the SAP business information warehouse*. Wiley (cit. on p. 37).

Münch, Jürgen and Jens Heidrich (2004). "Software project control centers: concepts and approaches." In: *Journal of Systems and Software* 70.1, pp. 3–19 (cit. on p. 106).

Mutafelija, Boris and Harvey Stromberg (2003). *Systematic process improvement using ISO 9001: 2000 and CMMI*. Artech House on Demand (cit. on p. 4).

Negash, Solomon (2004). "Business intelligence." In: *Communications of the Association for Information Systems* 13.1, pp. 177–195 (cit. on p. 4).

Parmenter, David (2010). *Key performance indicators (KPI): developing, implementing, and using winning KPIs*. Wiley (cit. on p. 27).

Pedersen, Torben Bach and Christian S Jensen (2001). "Multidimensional database technology." In: *Computer* 34.12, pp. 40–46 (cit. on p. 52).

Poncin, Wouter, Alexander Serebrenik, and Mark van den Brand (2011). "Process mining software repositories." In: *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE, pp. 5–14 (cit. on p. 105).

Rossberg, Joachim (2009). *Application Lifecycle Management*. English. Apress. ISBN: 978-1-4302-1080-1. DOI: 10.1007/978-1-4302-1079-5_2. URL: http://dx.doi.org/10.1007/978-1-4302-1079-5_2 (cit. on p. 11).

Royce, Winston W (1970). "Managing the development of large software systems." In: *proceedings of IEEE WESCON*. Vol. 26. 8. Los Angeles (cit. on pp. 14, 15).

Runeson, Per, Magnus Alexandersson, and Oskar Nyholm (2007). "Detection of duplicate defect reports using natural language processing." In: *Software Engineering, 2007. ICSE 2007. 29th International Conference on*. IEEE, pp. 499–510 (cit. on p. 110).

Sayal, Mehmet et al. (2002). "Business process cockpit." In: *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, pp. 880–883 (cit. on p. 107).

Singh, Gurdev, Dilbag Singh, and Vikram Singh (2011). "A Study of Software metrics." In: *International Journal of Computational Engineering and Management* 11, pp. 2230–7893 (cit. on p. 25).

Sneed, Harry M, Richard Seidl, and Manfred Baumgartner (2010). *Software in Zahlen: Die Vermessung von Applikationen*. Hanser (cit. on pp. 24–26).

Stücka, Renate (2013). “Passt überall.” In: *iX Developer* 3/2013, pp. 78–80 (cit. on p. 22).

Van der Aalst, Wil MP (2011). *Process mining*. Springerverlag Berlin Heidelberg (cit. on p. 105).

Vassiliadis, Panos, Alkis Simitsis, and Spiros Skiadopoulos (2002). “Conceptual modeling for ETL processes.” In: *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*. ACM, pp. 14–21 (cit. on p. 52).

Wagner, Stefan et al. (2010). “Softwarequalitätsmodelle–Praxisempfehlungen und Forschungsagenda.” In: *Informatik-Spektrum* 33.1, pp. 37–44 (cit. on pp. 25, 29).

Zur Muehlen, Michael and Michael Rosemann (2000). “Workflow-based process monitoring and controlling-technical and organizational issues.” In: *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*. IEEE, 10–pp (cit. on pp. 104, 105).

Online References

agilemanifesto @ONLINE (Mar. 2013). URL: http://agilemanifesto.org/principles.html (cit. on p. 15).

Atlassian ALM @ONLINE (June 2013). URL: http://www.atlassian.com/software (cit. on p. 20).

Bugzilla process @ONLINE (Apr. 2013). URL: http://www.bugzilla.org/docs/2.16/html/how.html (cit. on p. 103).

Business Computing World @ONLINE (May 2013). URL: http://www.businesscomputingworld.co.uk/top-10-software-failures-of-2011/ (cit. on p. 3).

Business Intelligence @ONLINE (May 2013). URL: http://www.gartner.com/it-glossary/business-intelligence-bi/ (cit. on p. 36).

Collab ALM @ONLINE (June 2013). URL: http://www.collab.net/ (cit. on p. 20).

Coverity @ONLINE (July 2013). URL: http://www.coverity.com/products/index.html (cit. on p. 44).

Gartner @ONLINE (June 2013). URL: http://www.gartner.com/technology/reprints.do?id=1-1ASCXON&ct=120606&st=sb (cit. on p. 19).

HP ALM @ONLINE (June 2013). URL: http://www8.hp.com/us/en/software-solutions/software.html?compURI=1174315#.UcB4hflQaYU (cit. on p. 21).

http://innovationcenter.deteconusa.com/ @ONLINE (Apr. 2013). URL: http://innovationcenter.deteconusa.com/article/next-generation-mobile-application-management-strategies-for-leveraging-mobile-applications-within-the-enterprise/ (cit. on p. 11).

IBM ALM @ONLINE (June 2013). URL: http://www-03.ibm.com/software/products/us/en/category/SW860 (cit. on p. 20).

ISO 9000 @ONLINE (May 2013). URL: http://www.iso.org/iso/iso_9000 (cit. on p. 4).

Jaspersoft BI @ONLINE (July 2013). URL: https://www.jaspersoft.com/de/products-de (cit. on p. 40).

JPivot @ONLINE (July 2013). URL: http://jpivot.sourceforge.net/ (cit. on p. 48).
Microsoft ALM tools @ONLINE (Apr. 2013). URL: http://www.microsoft.com/visualstudio/eng#alm (cit. on p. 20).
Microsoft MDX reference page @ONLINE (Apr. 2013). URL: http://msdn.microsoft.com/en-us/library/aa216767(v=sql.80).aspx (cit. on p. 75).
MySQL @ONLINE (Apr. 2013). URL: http://www.mysql.com/ (cit. on p. 38).
OSLC @ONLINE (June 2013). URL: http://open-services.net/resources/tutorials/integrating-products-with-oslc/overview-of-oslc/ (cit. on p. 22).
Parasoft @ONLINE (July 2013). URL: http://www.parasoft.com/jsp/products.jsp?itemId=13 (cit. on p. 43).
Pentaho BI @ONLINE (July 2013). URL: http://www.pentaho.com/explore/products/ (cit. on p. 39).
Pentaho Community @ONLINE (Apr. 2013). URL: http://community.pentaho.com/ (cit. on pp. 38, 39, 75).
QBench @ONLINE (June 2013). URL: www.qbench.de (cit. on p. 106).
Rally ALM @ONLINE (June 2013). URL: http://www.rallydev.com/about/what-is-rally (cit. on p. 20).
Saiku Analytics @ONLINE (July 2013). URL: http://analytical-labs.com/ (cit. on p. 39).
Salesforce @ONLINE (July 2013). URL: http://www.salesforce.com/de/ (cit. on p. 64).
SAP BI @ONLINE (July 2013). URL: http://help.sap.com/bobi (cit. on p. 37).
Sonarqube @ONLINE (July 2013). URL: http://www.sonarqube.org/downloads/ (cit. on p. 42).
Stages @ONLINE (July 2013). URL: http://stages.methodpark.de/ (cit. on p. 44).