FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Global Joint Artifacts for Outsourced Software Testing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Markus Gassner
Matrikelnummer 9925801

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Thomas Grechenig
Mitwirkung: Paul Pöltner

Wien, am _____ _____ _____
(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Global Joint Artifacts for Outsourced Software Testing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Markus Gassner

9925801

ausgeführt am

Institut für Rechnergestützte Automation

Forschungsgruppe Industrial Software

der Fakultät für Informatik der Technischen Universität Wien

Betreuung:

Betreuer: Thomas Grechenig

Mitwirkung: Paul Pöltner

Wien, _____

Eidesstattliche Erklärung

Ich erkläre an Eides statt, daß ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfaßt, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am _____ _____

Name

Kurzfassung

Soziale Netzwerke wie Facebook, Xing oder LinkedIn fördern unter anderem die Vernetzung von einander unbekannten Menschen basierend auf geteilten Interessen und Aktivitäten. Es gibt allerdings heutzutage noch keine Plattform, die einen Austausch und Vernetzung hinsichtlich gemeinsamer Aufgabenstellungen ermöglicht.

Die vorliegende Arbeit setzt sich als Ziel, die Anforderungen an eine solche Plattform zu erheben, die die bestehenden Netzwerke erweitert. Dabei wurde der Begriff "Global Joint Artifacts" (GJA) neu eingeführt. In der Software-Entwicklung werden un/fertige Arbeitspakete bzw. Projektteile häufig als "Artefakte" bezeichnet. Global Joint Artifacts sind somit global verteilte, aber miteinander verbundene Artefakte. Neben diesem abstrakten Konzept trägt auch die zu entwerfende Plattform den Namen "Global Joint Artifacts".

Zunächst wurde eine theoretische Recherche durchgeführt, um unterschiedliche Trends, den State of the Art und technologische Möglichkeiten aufzuzeigen. Dabei fokussiert die vorliegende Arbeit den global verteilten Softwaretest, wenn Menschen mit ihren Artefakten, sowie die Artefakte miteinander verbunden werden. GJA soll im Software-Test verwendet werden können, um Arbeitspakete, Bestandteile der Software-Entwicklung, Software-Fehler und grundsätzlich das Wissen im Projekt zu verwalten.

Zur praktischen Anwendung der theoretischen Erkenntnisse wurde ein Prototyp von GJA implementiert, der unter Berücksichtigung der Recherche die erarbeiteten Anforderungen erfüllt. Er soll aufzeigen, dass der Wert ursprünglich alleinstehender Information durch die Verbindungen erhöht wird. Auch Benutzer an verteilten Standorten können so am Wissensaustausch und der Wissensverteilung teilnehmen.

Als Ergebnis der Arbeit wurde der GJA Prototyp mit ausgewählten Systemen der Software-Entwicklung und -Test verglichen, wobei deutlich gezeigt werden konnte, dass GJA deutlich dazu beiträgt, rasch auf ähnliche Daten zuzugreifen und Wissensträger aufzufinden. Neben der Ersparnis von Zeit und Geld werden die alleinstehenden Informationen aufgewertet, Redundanzen vermeidbar und Synergien nutzbar gemacht.

Keywords:
Verteilte agile Entwicklung, Globale Software Entwicklung, Software Test, Outsourcing, Distributed Computing, Knowledge Management, Soziale Netzwerke

Abstract

Social networks like Facebook, Xing or LinkedIn support people previously unknown to each other in getting connected based on their shared interests. But a common platform that would allow them to additionally find each other, get connected and exchange information according to their shared tasks does not exist yet.

The paper strives to define and specify such a platform that extends the current networks and introduces the term "Global Joint Artifacts" (GJA). In software development in/complete work packages and project pieces are often called "artifacts". As such Global Joint Artifacts are globally distributed, but connected and dependant artifacts. Apart from the general concept the platform to be designed is also named "Global Joint Artifacts".

First a theoretical study was conducted to identify the current trends, the state of the art and the technological possibilities. The paper focuses the outsourced software test, where people get connected with their artifacts, and artifacts are interconnected with each other. It should be possible to use GJA to manage the work packages, software development artifacts, defects and generally the knowledge in a project.

A proof of concept prototype was implemented for the practical application of the theoretical findings. Its purpose is to show that the value of originally independent information is increased by connections. So even users at remote locations can participate in knowledge exchange and knowledge distribution.

The result of the paper is the comparison of the GJA prototype to existing software systems, which are used for software development and test. This proved clearly that through its connections GJA allows quick access to similar information and allows to find the carriers of knowledge. Together with saving time and money the information's value is increased, redundancies can be avoided and synergies can be exploited.

Keywords:
Distributed Agile Development, Global Software Development, Software Testing, Outsourcing, Distributed Computing, Knowledge Management, Social Networks

Table of Contents

List of Figures

List of Tables

1 Introduction

This chapter serves as an introduction and discusses the problem, motivation and research goals. The latter are formulated as hypotheses.

The term „Global Joint Artifacts“ is a slogan for interconnected data. Global Joint Artifacts has two meanings. First it stands for the artifacts themselves: Workpackages, defects or any kind of arbitrary knowledge items. In its second meaning it is the social website that allows connecting the artifacts to people and also artifacts with other artifacts. As such it is a system to organize and share knowledge.

Global Joint Artifacts does not try to be just another new social network, but utilizes the existing frameworks that already work well with interconnecting people. It is an add-on to the existing networks, by adding the artifacts to the global system. Its use is open to the public sector, business as well as academic applications.

The concrete application of the concept is for globally distributed software development test teams.

Neither the concept nor the application exist yet. As such this thesis can not only focus on working on the hypotheses, but also has to detail the theoretical background and out of those fundamentals specify the concept and the prototype application Global Joint Artifacts. In future work a thorough formal analysis of the then existing software can be created.

1.1 Problem

Traditional project management and controlling literature often focuses on the planning activities of software development. Many tools exist for example to create GANTT diagrams or work-breakdown-structures. Many books were written on project development methods like the Rational Unified Process or also newer approaches like Scrum. Even global software development is supported by configuration management tools like CVS or Subversion.

But too often the people involved in the development are seen as static entities, like employees. In a global software development scenario you will have people joining and leaving the project regularly. Teams are flexible and experts are assigned/connected and unassigned/disconnected to the project on a needs basis. It is simply not sufficient to only narrow down the distance of teams by modern communication means.[1] Instead it is vital for successful projects to connect the right people, which posses the right knowledge, with their history, experiences and already created work - artifacts.[2]

Through cooperation and collaboration time and money can be saved. People working on similar content should be able to connect, work together, share their findings and collaborate to a common goal.[3]

Further details on the status quo can be found in chapter 2.

1.2 Motivation

For ten years the author has been working in a software development company. The motivation for this paper is fueled from working in that field. An example are “re-inventing the wheel” experiences, where one artifact was created in a department just to realize that the very same artifact already existed in another. Also too often too many parallel systems exist, which perform the same work related tasks.

Good examples are time tracking, project controlling, organization of work packages, customer calls and defects. There the same data and information has to be provided several times in different systems. Often historical values for implementing change requests or fixing defects are taken into account when planning the next increment. If the data basis is incorrect or incomplete the metrics and assumptions will also be flawed.

It’s challenging to identify who would be the perfect match for a certain task in a project. With a lot of historical data, it would become obvious who already worked in a certain field and probably has extensive knowledge for the problems at hand. A normal team usually consists of “normal” people and not just the best of the best – simply because normal people tend to be more common and therefore affordable. As such we should strive to use everyone on the team to his or her best. [3]

1.3 Research Goal

Generating substantially original knowledge in the information society has become a challenge. Sometimes existing knowledge is only reinvented or presented after others who had worked on the same subject. Instead of combining resources, research is often undertaken in parallel. But if one starts in a field it’s easier said than done to grasp the status quo. At that early point in time the big picture is missing and you can only see chunks of the overall situation.

It's even more difficult to figure out who is currently working on the topic. As a result the acceptance rates at conferences of submitted papers are decreasing for many years now.[4]

The challenges of the 21st century can only be overcome by collaboration.[5] People must be enabled to spot similar projects and project artifacts, so they can make the decision to either participate or if they cannot or don't want to, may continue their work in different fields.

This is accomplished by not only connecting people and artifacts, but also similar artifacts with each other. A concrete example for distributed testing would be to connect two setup test cases. If another person has to write a setup test case for another project, the tester can already utilize from re-using parts of the existing document as template, but also knows who was originally involved in specification and execution of these test cases. Also metadata exists, like if it was a manual or automated test case, how long it took the person to write the test case, how long an execution of that test case took and how often it was run.

In a software development project often many systems are used to track requirements, bugs and calls from customers. Usually those systems are autonomous. As said Global Joint Artifacts does not try to replace existing systems, but should be an additional system. It is just a system helping to understand the vast amount of information.

The overall assumption is that the network of connected people and artifacts results in added value, as compared to having this as individual information hidden in several systems, accessible only to those actually looking for specific information.

1.3.1.1 Approach

For testing the following hypotheses live data from a call tracking (Oracle Service Request) and bug tracking (Rational ClearQuest) system were used. These systems are technically independent of each other, yet contain knowledge that is semantically related. Thus the traditional systems only provide a narrow view on a topic, but prevent spotting the big picture. The traditional systems are compared with the assumed added value from the Global Joint Artifacts prototype, which creates a uniform and complete view on a topic. Such a topic could be the complete dependency chain from the initial discussion with a customer at a trade show, over several steps in the product development till a test case that should finally test that feature. The assumed added value and benefits will be evaluated in the hypotheses. The live data was made entirely anonymous. All the people's and project names were changed for confidentiality reasons.

1.3.1.2 Hypothesis 1: Increased benefit when connecting people and artifacts

The connection of artifacts with people helps remote teams to clearly understand where requirements, findings and assumptions derive from. This should let them work self-sufficient from the main team and allow them to better estimate and prioritize.

1.3.1.3 Hypothesis 2: Increased benefit when connecting artifacts and artifacts

Only if the whole picture is visible in a single view (like Global Joint Artifacts) the background, preconditions, and context become evident.

The findings for the hypotheses are discussed in chapter 5 “Results and analysis of the prototype”.

1.4 Structure

The thesis is structured to give a clear and logical red line to follow. First the theoretical fundamentals are discussed, then the practical work on the prototype is discussed and finally the prototype is analyzed.

Chapter 2 covers the basics and definitions of the usage scenario for Global Joint Artifacts, which is the concrete field of outsourced software testing. As such it first covers the very foundation of how knowledge can be managed in Chapter 2.1 Knowledge Management. Chapter 2.2 covers software testing and is followed by chapter 2.3, which dives into the details of software development, software development project management and development process models. The latter chapter discusses industry standards like the Rational Unified Process or Scrum. As the name Global Joint Artifacts implies a discussion on outsourcing and offshoring is needed, which is explained in chapter 2.4 Global Software Development. The fundamentals chapter is concluded by chapter 2.5, which discusses necessary technological aspects for storing such highly interconnected data. Chapter 2 is the backbone of the paper and gives insight into its academic and industry background.

The practical work, the Global Joint Artifacts prototype, is discussed in chapter 3 and 4. While chapter 3 presents the basic definition, requirements and analysis chapter 4 can be read as a design and development documentation for the prototype. Chapter 4.2 discusses the the used methods and technologies in more detail.

The hypotheses set by the thesis are discussed in chapter 5, which is a summary of the results and their analysis of the prototype. In chapter 6 a critical discussion follows, which shows strengths and weaknesses of the results and gives an outlook into the future of this topic. Finally chapter 7 summarizes the thesis.

2 Fundamentals - Relevant topics and state of the art

The Global Joint Artifacts are an approach to organize and share knowledge globally and get connected to experts. The concrete example in this thesis is the use of Global Joint Artifacts for off-shored software testing.

The following subchapters will discuss the state of the art in the fields of knowledge management, software development and outsourcing. As even in small groups large amounts of interconnected data would be created new ways to store and retrieve such data need to be evaluated.

These topics are the fundamental key stones and theoretical basis for this thesis. As such they are explained in more detail to show the environment of the problem.

2.1 Knowledge Management

Data, information and ultimately knowledge are key factors for successful enterprises in today's world. Traditional economics as characterized by Adam Smith and David Ricardo knows the production factors land, labour and capital [6] [7]. In the information age, knowledge has been added to this list as well [8].

This chapter will look closer into how knowledge can be managed in a company, and what strategies help organizations to utilize their people's knowledge to the maximum.

The analysis starts to look at the keystones of knowledge management data, information and knowledge itself. The distinct patterns of knowledge and how they can help with innovation are discussed in chapter 2.1.3. Then the main strategies to manage knowledge in an organization or company are approached. The chapter is then concluded by looking at the learning strategy Connectivism, which explains how people can learn from interconnected knowledge.

2.1.1 Data, Information and Knowledge

Before the discussion of knowledge management can begin, a short look into its foundations of data, information and knowledge is necessary.

The steps from data to information and from information to knowledge are usually seen as adding value to the previous. But no uniform definition exists of those three terms.

The basic definition from the dictionary for data is “factual information (as measurements or statistics) used as a basis for reasoning, discussion, or calculation” and “output by a sensing device or organ that includes both useful and irrelevant or redundant information and must be processed to be meaningful” [9]. Information is “knowledge obtained from investigation, study, or instruction”[10]. Knowledge is “the fact or condition of knowing something with familiarity gained through experience or association” and “acquaintance with or understanding of a science, art, or technique”[11]. As can be seen usually those three terms are defined recursively, using one of the others to describe them.

Willke defines data as numbers, speech/text or pictures, encoded and available in large quantities. Data is only seen as an abundant raw material, without much inherent value. Only when processed to information or knowledge value is added. Data becomes information by getting connected into a relevant context. Relevancy is always seen as specific to a certain system. Knowledge is information that is connected with experience and personal relevancy. Only information with a utility to someone can be seen as knowledge of or to that person or organisation [12].

Ten years earlier Ackoff defined data as raw material that is not necessarily in a usable form. Information is data with meaning, which is connected. Relations are by the individual or organisation. Knowledge is mainly perceived as amassed information, where patterns can be seen and understood. Ackoff introduces a fourth level, called wisdom, which posses the highest degree of connectedness and understanding, even of principles.[13]

So, to try to get to the essence of all the definitions data can be seen as the raw numbers e.g. output by some measuring device, without much value in themselves. Information is the data already applied to a certain context. Knowledge is what some individual or organisation derives from the information.

2.1.2 Introduction to Knowledge Management

Just as with the basic cornerstones data, information and knowledge, also for knowledge management a clear and uniform definition does not exist. Knowledge management is generally about storing and sharing knowledge as well as aiding creating new knowledge or transforming existing knowledge. In their paper Bouthillier and Shearer [14] take a look at private and public institutions and what goals they pursue under the term “knowledge management”. Communication is an important factor, as people from dif-

ferent departments or who are geographically distributed need to get connected to share ideas.

Another aspect is not to directly connect people, but to allow them to tap into a network of stored knowledge and retrieving the inherent information. Means for this are audits to discover knowledge within an organization, expert databases identifying experts in knowledge areas and knowledge databases that store the knowledge in the form of documents. The distribution or retrieval of knowledge is an equally important topic as to storing it in the first place. The knowledge must spread throughout the organization to be used by many individuals. Alerts can help to broadcast updated information. Continuous education tries to help people through training activities. Virtual collaboration enables people to work together ignoring departmental or geographical boundaries. "Knowledge managers are concerned with the dynamic dimension of knowledge, not only with data, information and technology."[14]

2.1.3 Patterns of knowledge[15]

So that an organisation is able to be innovative, its people need to be able to effectively make their knowledge available to others, so those can learn and transform existing knowledge to their areas of interest. Communicating knowledge has to be a continuous process and on all levels. Continuous innovation is seen as the only way to stand out of the crowd of similar products. [15]

Knowledge can be abstracted into two different categories:

"Explicit knowledge is formal and systematic."[16] This kind of knowledge can be learned from books and can be found in e.g. technical requirement documents for a software project or scientific journals. These are rather objective facts, which can be often measured or proved. Contrary tacit knowledge is a very subjective and personal kind of information. It consists of mental models and beliefs. It is difficult to communicate or put down in written form. This kind of knowledge is usually passed on from master to apprentice e.g. just like in a craft or profession. Simplified it can be summarized as "know-how".

It is now interesting to take a look into ways to transform and transmit one kind of knowledge – even between the same categories.

2.1.3.1 Tacit to tacit (socialization)

As mentioned above this kind of knowledge is often given from master to apprentice. Objective is to transform the tacit knowledge of person A to the tacit knowledge of per-

son B. Means to achieve this are observation, imitation, and practice. Above means already suggest that this is both for the master as well as the apprentice a time consuming effort, which can not easily be applied to a larger organisation or company. [15]

2.1.3.2 Explicit to explicit (externalization)

Existing formalized knowledge is taken to either educate one person or to create new knowledge by assembling pieces to a new whole. The organisation's knowledge value is not drastically increased by this process, but the individual can gain new insights into a subject. [15]

2.1.3.3 Tacit to explicit (articulation)

Converting existing tacit knowledge into explicit knowledge is a challenging process. At one point in time it might become necessary to formalize knowledge, to put it down into a book or manual. Usually people who used tacit knowledge over years will be able to derive certain theories or aspects that can be systematically written down. Slogans and metaphors can help to articulate hard to grasp tacit knowledge. They are an attempt to put all the tacit knowledge into an easy to communicate and intuitively understandable term. [15]

2.1.3.4 Explicit to tacit (internalization)

This is probably the most desired aspect if someone learns from formalized knowledge – that this knowledge is not just repeated, but really internalized. The knowledge is not just used, but rooted in the people's way of working.

In the end all the patterns exist in dynamic interaction in what Nonaka calls the "spiral of knowledge" [15]. The initial metaphor starts the process of creating knowledge. The initial idea is then taken into an exploratory phase in which it is evaluated. Based on analogies it is decided what the object under investigation is like and what it is definitely not. In the last step an actual model is created from the knowledge that crystallized. This model then allows communicating the findings to the rest of the company or organisation. [15]

2.1.4 Strategies of managing knowledge[2]

In today's highly competitive markets mainly knowledge and how it is distributed in an organisation is the unique selling proposition.[15]

By the example of the consulting industry Hansen et al. explain that they see two distinct strategies of how knowledge can be made available to the people in a company or an organisation.[2] Their findings are not limited to the consulting firms as the point out, but can also be observed in e.g. the healthcare market. So it is assumed that their so called codification and personalization strategies are universal to all organisations engaged in creating and (re-)using knowledge.

Both strategies share the idea that re-using knowledge will allow people to save time. Instead of re-inventing solutions themselves they can tap into existing knowledge.

As the name implies the codification strategy tries to codify and store knowledge into databases. The main goal is to have once uncovered knowledge available for later access. Here means to store and subsequently retrieve information are important. The focus is to allow people to re-use existing knowledge, without the need to contact the person who originally created the knowledge. Knowledge objects are put into a repository for later re-use by other individuals.

Advantages of this approach are reduced communication cost and as people save time on their actual work, they can take on more work in the same time frame. Taking on more work should then result in growing the business.

A disadvantage as opposed to the personalization strategy is that the process of making the documents anonymous, codifying them so they can be retrieved, and even summarizing information to reports, needs specialists who post-process the knowledge to usable knowledge objects.

The personalization strategy is more a directory of experts. People are encouraged to actually contact the person who was previously already involved in solving a similar problem. The provided means are mainly revolving around connecting people with each other. Proficient colleagues give initial input or serve even as advisers throughout the whole project.[2]

It is equally important that a mind set for this strategy exists within the organization. Telephone calls and eMails need to be answered promptly to make this strategy work. People need to be willing to maybe even travel to aid their colleagues face-to-face. Previous work is also stored in the personalization strategy, but it serves here only as a starting point to connect the appropriate people, and not to actually learn from the documents themselves.

Unlike the codification strategy the advantage of personalization is not so much that more work can be done, but that the consultant can really provide in-depth knowledge.

So the desired advantage is to drastically increase the quality of output, and not so much the efficiency.

What strategy can or should be used depends on the drivers problem, degree of reusability, business objective, the people involved and rewards for contributors: [2]

Codification can be used to solve similar problems over and over in a timely and high-quality manner. Personalization provides creative and highly customized solutions to individual problems. So what strategy should be used highly depends on the problem at hand.

Codification serves a very good purpose if solutions need to be applied or implemented over and over again. You invest once in creating the abstract knowledge objects and then can easily and cheaply in matters of time and cost apply them to several problems. Contrary personalization works best for highly customized solutions, where every client's problem is very individual.

The business objective also influences the strategy to use. For codification to be successful you need a lot of similar projects and need to try to generate large overall revenue, but due to the similar solutions, it will not be possible to charge much from each client. Personalization allows generating high margins, due to highly customized, non-standard solutions. [2]

Codification demands users to implement existing knowledge, so the ideal workforce would be college graduates, who are used to learn and derive knowledge from written documents. Personalization is for inventors and highly communicative people, who learn through mentoring.

In a scenario where codification is used people should be rewarded both for using as well as contributing to the repository of knowledge objects. To make personalization work, people need to be encouraged to directly share their knowledge and if their help is requested, respond in a timely manner.

Taking those drivers and customer requirements into account gives organisations a good starting point to decide on the appropriate knowledge management strategy. But although Hansen et al. come to the conclusion that a company should not try to excel in both strategies, it is still a good idea to use the second strategy as a supplement to the primary. They recommend maintaining an 80:20 ratio between the main and secondary strategy. [2]

2.1.5 Connectivism[17]

Acquiring new knowledge is the process of learning. One recent learning theory that emphasises that learning is an interconnected process is connectivism. Built on top of traditional learning theories like behaviourism, cognitivism and constructivism it explicitly tries to foster distributed learning and collaboration and help to master the challenges of knowledge management activities. “The organization and the individual are both learning organisms.”[17]

Behaviourism and cognitivism see knowledge as an external asset. Cognitivism already sees the tendency that the learner tries to create meaning and, thus, alters the knowledge on the way of learning. Looking back at the tacit to tacit knowledge transfer, it is obvious that e.g. an apprentice in bakery will learn much of his or her knowledge from the master, but throughout the process of learning will also add new and slightly changed knowledge to his set of skills.

Siemens [17] cites Stephenson [18] to emphasise that connecting people to elevate personal knowledge is in today’s world of utmost importance: “Experience has long been considered the best teacher of knowledge. Since we cannot experience everything, other people's experiences, and hence other people, become the surrogate for knowledge. ‘I store my knowledge in my friends’ is an axiom for collecting knowledge through collecting people.”[18]

Only if we share knowledge, we will be able to tap into the knowledge of others. And without the access to other people’s knowledge we are limited, as our own capacity to gather information is limited. “Within social networks, hubs are well-connected people who are able to foster and maintain knowledge flow. Their interdependence results in effective knowledge flow, enabling the personal understanding of the state of activities organizationally.”[17] It is also stressed that through feedback loops between individual and the network, as well as organisations and the network, people are able to utilize the connections to stay up to date on their topics of interest. Connectivism claims that the connections can and will amplify the knowledge creation.

For the management connectivism has the clear implication that only connected and diverse teams can come up with innovation. Such teams are also more promising to deliver faster implementations of new ideas. [17]

Connectivism also addresses the problem that in the modern world there is much to learn, as access to information is rather quick and easy. But we need to be able to see a purpose or meaning in why we should spend time on a certain topic. Other people that are also interested in that topic could be that purpose or meaning. Barabási wrote that “the node with the most links will get more links.”[19]. And pointing out valuable in-

formation is not limited to learning, but can be an important aspect of general knowledge management as well.

Connectivism can be summarized as the theory that people cannot know all he or she will need in their daily life, so they have to get connected to share and combine their knowledge. To know where to find information is also higher valued than to actually know certain things at one point in time.

2.1.6 Conclusion

The challenges given in this chapter can be summarized as follows:

So, knowledge can be shared, it has to be structured
Sharing of ideas and thoughts needs communication
Distributed teams lack natural communction (e.g. in the hallway of the office)
To reach many, information needs to be broadcasted
Technology can help to reduce the amount of communication needed to spread information
A single person cannot know everything, and so has to collaborate with others to achieve their goal

Table 1: Challenges from Knowledge Management

2.2 Software Testing

Similar to most engineering disciplines software development imposes the need to verify, validate and evaluate the end product of the development process. Similar to building an elevator the software test tries to confirm that desired quality criteria such as security are met by attempting to deliberately breaking the system. Testing objectivises information derived from the testers and gives input to the developers. This chapter will give a general overview of testing software and what structures and topics are relevant.

Testing is a diverse field with many different opinions and testing is often performed by people from different fields and backgrounds.[20] This chapter will try to accumulate the information from accepted standards and sources in the industry.

The next chapter 2.3 will then go into more detail on the software process models with a focus on testing. So, this chapter should also be seen as a precondition for fully understanding the next chapter.

2.2.1 IEEE 829 [21]

The standard for software and system test documentation is a common framework for testing. The test tasks, inputs and outputs are defined. It applies to software, hardware, mixed software/hardware systems and their interfaces. The standard applies to all states of the software development life cycle, no matter if the software is being developed or already maintained or even re-used.

The standard emphasizes the key concept of the integrity levels. “An integrity level is an indication of the relative importance of software (or of a software characteristic, component, or test level) to its stakeholders, as established by a set of selected attributes such as complexity, risk assessment, safety level, security level, data integrity, desired performance, reliability, quality, cost, size, and/or other unique characteristics of the software.”[21] So, the highest level would be critical parts of software that if malfunctioning would threaten people’s life, while the lowest level are only negligible features. Depending on integrity level different minimum test tasks are required.

The standard defines several test documents. Those test documents are an example for Global Joint Artifacts, which are interconnected between people, but also between related documents within a project and content-wise related documents across projects.

The “Master Test Plan” connects the multiple levels of a system under test or several projects. It contains the test objectives, purpose and scope of the test effort. It gives an overview on the test with schedule, resources, roles and responsibilities, risk and as-

sumptions, project standards, integrity schema, number of levels of test and overall tasks to be performed.

The prefix “Level” on the following documents means that such a document should be available for the levels of the project. The levels are to be defined by the individual project. For example the integration levels component/unit, component integration, system, and acceptance could be used. But, it could also be operations, installation, maintenance, and regression.

A “Level Test Plan” contains scope, approach, schedule and resources trimmed down to a distinct level of test. It should name the test tasks, like features to be tested, and by whom those tasks should be performed. Why certain features or combinations were not tested should be remarked. Examples for test approaches would be black box or white box test, analysis or inspection. Depending on the selected detail the level test plan contains also a “Test Traceability Matrix”, which links requirements with one or more test cases. To be able to know when a test has finished pass and fail criteria should be specified, and optionally also suspension and resumption criteria. A typical pass criterion would be that no critical or severe anomalies were found in the system under test.

The “Level Test Design” refines the test approach from the level test plan and the features under test. The test cases are described on a high level or as scenarios. The test case design functions as an umbrella for one or several more detailed test case documents.

The “Level Test Case” describes the test cases from the test level design in more detail, containing input parameters and expected output.

The “Level Test Procedure” goes into even more detail describing all necessary preconditions, steps and relations to other test procedures. Depending on the level, but also technical or functional skills of the actual tester the test cases will contain even detailed descriptions of the steps to be performed.

The “Level Test Log” is a record on the test execution. Like all the other documents it can be fully tailored to the project. It should contain at least the version information of the item under test, who ran the test, at what date and time and if the test passed or failed.

An “Anomaly Report” is generally a document that tracks all problems, incidents, defects or errors found during testing that need further investigation. Such reports are usually tracked with a database-driven system such as Rational ClearQuest, Mantis or BugZilla.

While the focus of the “Level Interim Test Status Report” is usually just to summarize the testing activities for the current level, the “Level Test Report” often tries to additionally give further input. It evaluates the test results and gives recommendations on how the project can or should proceed. What details those documents really contain can be tailored to the specific needs of the company or project.

To aggregate the knowledge from all testing activities over all levels the “Master Test Report” is used. It gives an executive-level summary of the testing activities for the increment. It should give a categorized summary of the anomalies and which of those were solved and which are still open in the release. An important point in this document is the assessment of the increment’s quality. The document is closed with conclusions and recommendations for the increment.

So, the Master Plan expresses what should be done, while the Master Test Report gives the answer how this task was put into practice.

Depending on the size of the project, its integrity level and how long the system will be used, the documents can be combined or even eliminated at all. When using agile methods the standard recommends having a general approach test plan and at least keeping the anomaly reports documented.

The IEEE 829 standard was also very influential on the ISTQB certification, as can be seen from many references to the standard, as well as many terms taken from the document.

2.2.2 ISO 9126-1 Product quality -- Part 1: Quality model

Testing revolves around the complex of validating that the desired quality was achieved. The ISO 9126 standard specifies several quality criteria in detail. This allows straightening the terminology within the development and test team and allowing all parties involved to speak the same “language”. Such a taxonomy also helps the discussion between supplier and acquirer. [22]

At the same time the standard greatly helps in making-trade-offs between software product capabilities, identifying requirements, design and testing objectives and validating the completeness of requirements, design and testing.

“Comprehensive specification and evaluation of software product quality is a key factor in ensuring adequate quality.” [22]

The standard specifies a quality model for external and internal quality. It is separated into four parts. Part one is the quality model. Part two are external and part three the internal metrics. Part four are the quality in use metrics.

For this chapter only the first part, which explains the model (Fig. 1) itself, is of interest.

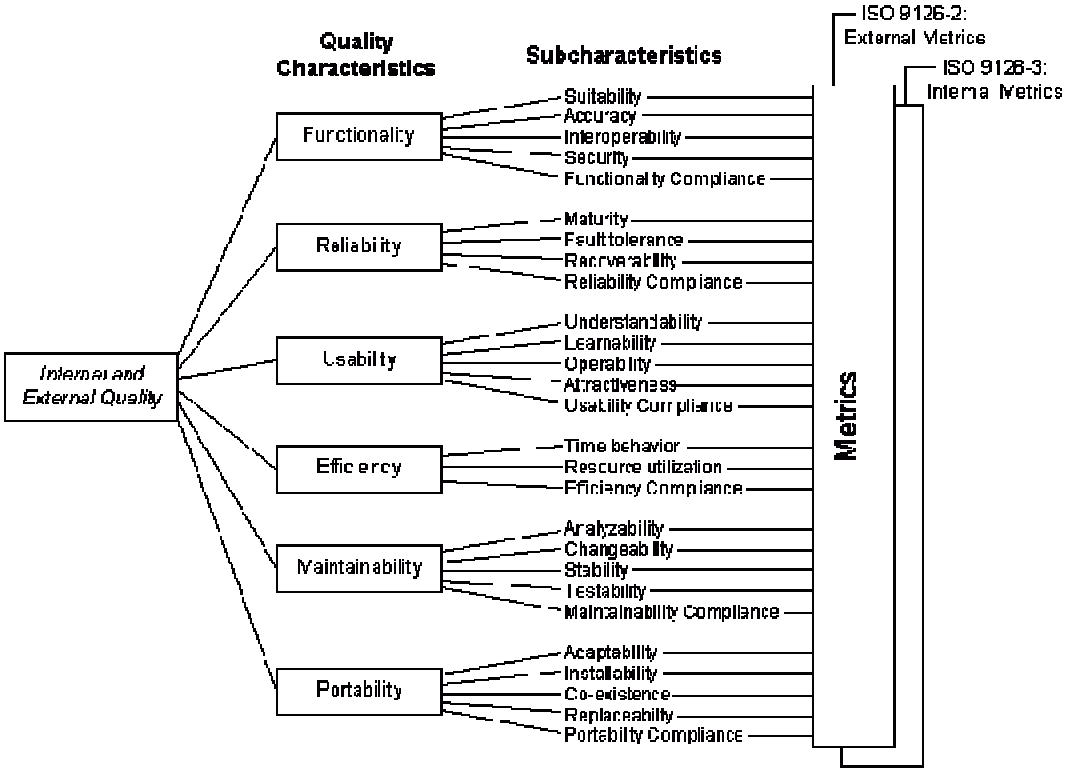

Figure 1: ISO 9126 [23]

The quality factor functionality describes the required capabilities of the system to be developed and under test. It specifies input and output behaviour and the results in the system. Suitability discusses if all requirements are realized in a suitable way. Especially when it comes to software architecture the discussion on suitability often has to revolve around trade-offs. An accurate system returns expected results according to the specification, similar to a calculator that returns 1 + 2 = 3. Interoperability verifies that the system under test is able to work with other systems that were specified. A secure system is protected against accidental and intentional access to data and functionality. Compliance is guaranteed if the system abides to the application specific rules, regulations and laws. This compliance is equivalent for the other quality factors.

Reliability describes a systems ability to keep its performance under specified conditions for a specified amount of time. Testing usually can give a statement on the maturity of the system – how often does the system fail. A fault tolerant system can keep functionality under failure condition. If a failure condition occurred recoverability

measures the cost to recover the system. For example it could measure time and cost of the recovery procedure, time needed to recover affected data, or overall time needed to recover from failure to a fully functional system.

The quality factor usability measures how easy it is to understand, learn and operate the system. It also looks at how appealing the visual appearance of the system is.

Efficiency mainly describes the time behaviour and resource utilization of the system.

A maintainable system can be used over a sustained period of time and even on changing platforms. Maintainability revolves around how difficult it is to analyze, change and repeatedly test the code.

Maintainability and portability are related. Maintainability looks more at the inner quality factors, while portability is focused on outer factors. A portable system can be easily adapted to new frame conditions, is easy to install, can co-exist with other related systems and can be easily replaced.

A concrete example for above is writing a test suite that implements and verifies an SDK. This software probably can make trade-offs in usability as it will be used by testers and not end users. This software probably does not need to be as efficient as end user software. Installability could also be less important as technically experienced engineers install it. As it does not generate revenue itself, it should be easy to develop and maintain, and probably also portable. Foremost test software needs to be accurate so the results are reliable.

2.2.3 Test types, differentiated by software state

Testing is a very diverse field, and there are several dimensions at which one can look for a taxonomy of testing.

One method is to distinguish by the state of the system under test and explore the techniques (Fig. 2) that can be used in that particular state.

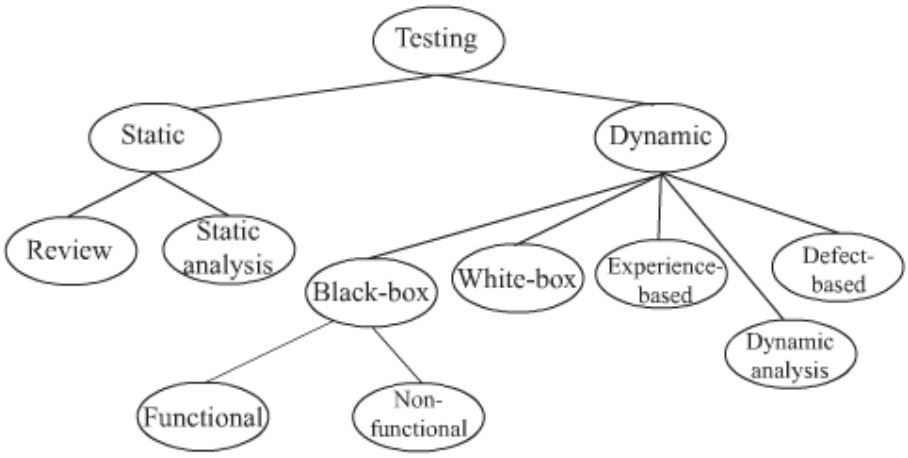

Figure 2: Test Techniques [24]

Dynamic testing looks at the running application, while static tests can be performed on the source code without prior compilation or execution. While the review usually puts the people in the place of finding defects, static analysis is usually performed by tools that assess source-code. Such a tool is for example FxCop, which verifies coding guidelines and naming conventions. Dynamic testing consists of five subcategories, of which black box and white box testing are the most prominent. [24]

In Black box testing the system runs and is actually used for testing purposes. ISTQB then further on differentiates between functional and non-functional testing, as also described in chapter 2.2.2 “ISO 9126-1 Product quality -- Part 1: Quality model” in this document. Black box testing does not know anything about the internals of the software, or does not use that knowledge. “The easy way to distinguish between functional and non functional is that functional tests address what the system does and non functional tests address how the system does what it does.”[24] As such the non-functional parts of black box testing, like usability tests, can also be performed by non-technical testers, who are more focused on the business logic of the software.

White box testing as opposed to this uses the knowledge about the internal workings of the software.

A concrete example is storing information in a database. For black box testing it is important that the values are stored accurately. As opposed to this for the white box testing it could be only interesting with what SQL statements it is stored, using a database profiler while the software stores its results.

A mixture of using black box and white box testing is unit testing. There internal knowledge is used, but still only entire functions are tested, without any interest in how the input is transferred into output. Thus, such tests are often called grey box testing.

The following test techniques usually overlap with black box or white box testing. Defect-based testing takes found anomalies and creates tests, to prevent the re-introduction of already found bugs into the software at a later point in time. If the code is instrumented and allows seeing in detail information on the inner behaviour it can be dynamically analyzed. Experience-based testing tries to make the most of the testers' personal skills and intuition from previous versions of the software or experience with similar applications and technologies.

2.2.4 Types of test, differentiated by test object [25]

Tests can be differentiated by the integration level of the test object.

The unit/component/module test checks individual components independently from other components of the software. In modern programming languages such as Java and C# usually the individual classes are tested by unit tests. In traditional MFC C++ development a module test could for example mean even a complete DLL.

As the components are isolated from the rest of the system it can become necessary to simulate the rest of the system through test drivers and stubs.

Such tests are often performed by the developers of the code themselves. Usually test cases cover certain input/output combinations. In Test Driven Development the test code is even written before the actual code. The weakness of this approach is that developers usually write code to prove that their code works, while the testers' task would be to intentionally break the system. These tests can elevate the accuracy of the system and especially help to prevent boundary errors.

The Integration test takes a glance at sub systems and interconnected components. Purpose of this kind of test is to find errors in interfaces or protocols and when data moves between the components. Monitors could become necessary to be able to log the movement of data and investigate issues, such as timing, load or performance problems.

The system test verifies the system as a whole. This test step verifies that the subsystems of the system work as expected and that the functional and non-functional requirements are met by the software. As such the test environment should try to reproduce the production environment as closely as possible. As the possible combinations for theoretical test cases of the full system are usually a high number, it will be impor-

tant to use some test technique such as risk based testing to test only what is necessary and feasible. Errors that should be found with this kind of test are wrong or missing functionality, lack of performance or usability.

The Acceptance test is a system test from the point of view of the acquirer and other stakeholders. Involving the end customer in this stage assures that the explicit requirements as well as the harder to tackle implicit expectations of the acquirer are really met. The acceptance test is the final step for confirming the acquirer's as well as final users' acceptance. Depending on the amount of customers the acceptance test can also be extended to a field test, often called alpha and beta test phases.

2.2.5 Automated testing

Test automation can either support manual test by automating certain test steps, or be the enabler for things that cannot be tested manually (e.g. creating test tools for APIs, and creating large scale tests). Test automation is often only perceived by management as a means to reduce testing cost, by replacing testers with automated scripts. Apart from this "hard" return on investment factor, there are also a couple of soft issues to think about. [26]

It can also be used to increase testers' morale by relieving them from repetitive tests and giving them interesting manual testing work. Instead of just re-executing already well specified and thus boring test cases testers get the opportunity to also test parts of the software, which are new, changed, or simply less often tested and do not have tests specified yet. New challenging tasks will make testers also more alert to issues in the software. [26]

At the same time an automated test guarantees to execute the very same steps over and over again. In rather complex test scenarios a human will likely never be able to perform exactly the same steps. Regular regression tests guarantee that after changes to the software the development department notices defects immediately, increasing again the confidence in the software's quality.[26]

„An effective test program that incorporates the automation of software testing has a development life cycle of its own. This development effort comes complete with its own strategy and goal planning, test requirement definition, analysis, design, and coding. Like software application development, the test development effort requires careful analysis and design."[27]

Test automation should therefore re-use findings from traditional software engineering in regards to processes and architecture, understand and properly report up to manage-

ment the hard and soft ROIs and keep the balance between a long termed structured development approach and a short termed service oriented approach.

A “good” test tool is one, which is quick, cheap and easy to create, understand, execute, maintain and evolve. Proper communication within the team as well as re-use of common components for configuration, logging, reporting, the GUI, and measurements, will help to fulfil these requirements.

2.2.6 Conclusion

The challenges given in this chapter can be summarized as follows:

Software testing is only part of a larger collaboration
Testers need to know a lot of information, and often very early in the project
Testers need plenty of information to specify and perform useful tests
Complete testing is not possible, so testers need to prioritize
IEEE 829 and ISO 9126 are helpful standards, yet they further increase the amount of documents needed for the project

Table 2: Challenges from Software Testing

2.3 Software Development, Software Development Project Management and Development Process Models

The chapter starts to describe the most fundamental terms involved in Software Development Project Management.

Then a selection of five concrete process models is described, also with respect to how testing is performed therein. A detailed discussion of Testing can be found in chapter 2.2 “Software Testing”.

Any discussion on software development starts with the waterfall model, being the most traditional software development process model. It already contains the most important steps of creating new software. The Rational Unified Process is an internationally well accepted process, and Global Joint Artifacts uses some of its terminology. The V-Modell XT is a requirement for developers working for the German federal authorities, so it is important throughout the entire DACH-region (which means Germany (D), Austria (A), and Switzerland (CH)). Above process models are mainly covered to see what artifacts and roles are defined there and for general completeness of the topic.

Scrum and Software Kanban are more recent development approaches, which stress the importance of agility in the development process. These will be discussed in more detail. Both are being adopted in the industry at a fast pace. Global Joint Artifacts with the concrete usage scenario for outsourced software testing will use Scrum as its example process model.

This chapter is concluded with a short summary and comparison of the process models.

2.3.1 Definitions

As a precondition to be able to describe the different process models in more detail, certain basic definitions need to be discussed:

2.3.1.1 Project

A project is a collection of related temporary activities to complete a specific unique artifact (product, service or some other kind of tangible result) between a defined start and end date. To achieve the objective the project consumes resources – both human and nonhuman (funding, equipment, …). The project ends either if the objective has been met or the project was terminated, because its objectives cannot be met. [28] [29]

2.3.1.2 Project Management

“Project management is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements.”[29] Project management can be separated in the five main process groups related to the phases in the project. These are initiating, planning, executing, and closing, while monitoring and controlling is a continuous responsibility throughout the project life cycle. If a project is completed within time, cost, and fulfils the customer’s requirements and is accepted one can conclude that the project management achieved its goals. [28]

2.3.1.3 Process Model

A process model describes what steps, in which order need to be performed for the organisation and implementation of a project. [30]

2.3.2 Waterfall Model

The waterfall model (Fig. 3) is the classic development process, which emphasises the engineering aspect of software development. Just like building a bridge, the problem is first analyzed, then the software designed, implemented and once everything is ready a final test is performed. [31]

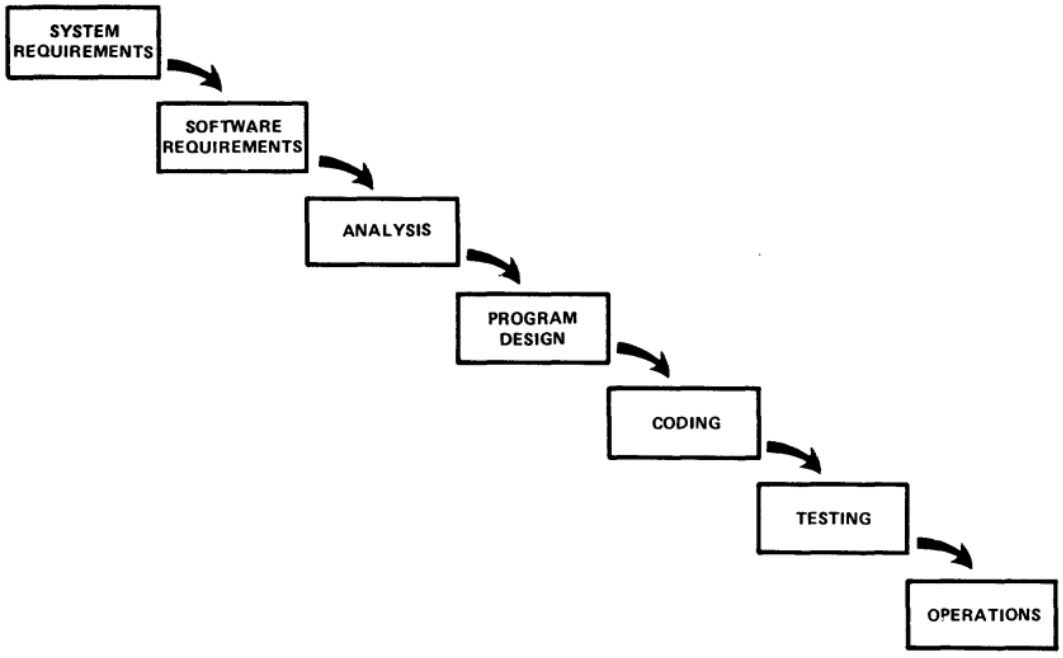

Figure 3: Classic waterfall model from Royce [31]

So, the waterfall process involves all important states of software development: Requirement analysis, design, implementation, and test.

The severe issue with this process is that the whole development is done in a strictly sequential process. Testing is performed in this process only at the very end of the process and only once.

Working with such a process even on small projects reveals critical deficits. It lacks feedback loops and even if they exist, usually only directly between successive steps. The later tests are performed the more expensive it will be to fix found defects[32]. There is also a high risk that the built product does not fit the customer's needs due to very late involvement.

Still, looking at processes like Scrum (see chapter 2.3.5) every sprint can be seen as a small self-contained waterfall process, so it can be asserted that the waterfall process works just fine for very small development items that do not exceed a couple of weeks with a few people involved.

2.3.3 Rational Unified Process

This chapter describes the Rational Unified Process (RUP), which is a software engineering process that is widely used throughout the industry. The RUP is a prescriptive process framework, meaning that many concrete guidelines and rules on how to work and behave in the project are given. It tries to be extensive, so one usually only selects a subset of the available components and yet it also allows to extend the process to the project's individual needs.[32]

The development of the Unified Modelling Language (UML) is tightly bound to the development of the Rational Unified Process (RUP), as the same people were involved: Grady Booch, Ivar Jacobson and James Rumbaugh. "Modeling is important because it helps the development team visualize, specify, construct, and document the structure and behavior of a system's architecture."[32]

2.3.3.1 Best practices

The process is based on six best practices [33] for commercial software development, which are described below.

Iterative and incremental software development. Already in the 1970s Royce [31] describes the problem with the sequential approach of a waterfall-like software development model. Since then software systems' complexity even increased. It is not possible to have a holistic view on the problem and to know all relevant details upfront. Therefore the RUP tries to refine the increments from one iteration to the next. Several itera-

tions of intermediate builds are normal in the Rational Unified Process. Those working increments help mitigating risks, as they allow seeing and experiencing the actual behaviour and if needed it is possible to quickly change requirements, features or the schedule. [33]

Requirements Management. The analysis phase gets special attention as said above complete knowledge of the problem will never exist. Use cases and scenarios are a major tool for requirements elicitation. The RUP provides concrete descriptions how to elicit and later on utilize the requirements. With the end user in mind the requirements should drive the rest of the development cycle and not vice versa. To document constraints and requirements and decisions on them is specifically demanded by the process. [33]

Component-based Architectures. The Rational Unified Process not only demands a flexible architecture that is able to handle changes and is easy to understand, but also tries to describe how to create it. Effective software reuse is motivated through promoting a modularized component architecture. Using component infrastructures like COM or EJB is specifically encouraged. [33]

Modelling software visually. The Rational Unified Process is tightly bound to the Unified Modelling Language. When working in teams it will become necessary to have a means to discuss the architecture. UML allows changing the degree of abstraction according to the current needs. [33]

Verify Software Quality. Poor quality software will also only generate poor acceptance from its users. The RUP stresses the need for proper software verification. Not just the mere functionality has to be tested, but also other quality attributes like reliability and performance. The RUP offers a whole test lifecycle of planning, design, implementation, execution, and evaluation. According to the RUP testing should affect the whole team, and not just a dedicated test team. [33]

Control Changes to Software. Tracking changes is essential, as changes will occur often to refine the functionality from one increment to the other. Using software configuration management can help to control, track and monitor changes in the iterative development, even when several distributed people work on a single source-file. [33]

2.3.3.2 Primary modeling elements

“A process describes who is doing what, how, and when.”[33] The RUP names the following elements:

The “who” are called workers. Workers are not actual people in the organization, but roles that can be either fulfilled by an individual or group of individuals. The roles assigned specify the responsibilities regarding activities and ownership of artifacts. Example for a worker’s role could be designer, someone performing a review or architect. Depending on the context, this will change concrete individuals over the lifetime of a product. [32]

The “how” are called activities. Activities connect workers to artifacts. The activity specifies the unit of work that a specific role needs to accomplish. Every activity has an explicit goal regarding an artifact, like the test lead (role) needs to create a test plan (artifact) for the next increment (artifact). Activities are an approach to distribute work among a team and can be used for planning and tracking. [32]

The “what” are called artifacts. “An artifact is a piece of information that is produced, modified, or used by a process. Artifacts are the tangible products of the project, the things the project produces or uses while working towards the final product. Artifacts are used as input by workers to perform an activity, and are the result or output of such activities.”[33] Examples for artifacts are all kinds of UML models, any document related to the project, source code or the executables created by the software development process. [32]

The “when” are called workflows. Workflows are small processes combining activities into sequences so a concrete artifact with inherent value is created. [32]

The following diagram (Fig. 4) shows above elements with the concrete example of a tester. Worker in this example is the tester role. Activities are “Implement Test” and “Execute Test Suite”. The “Test Suite” and also the “Test Script” within the test suite are artifacts. The whole process of Activities “Implement Test” to “Analyse Test Failure” is a workflow. [32]

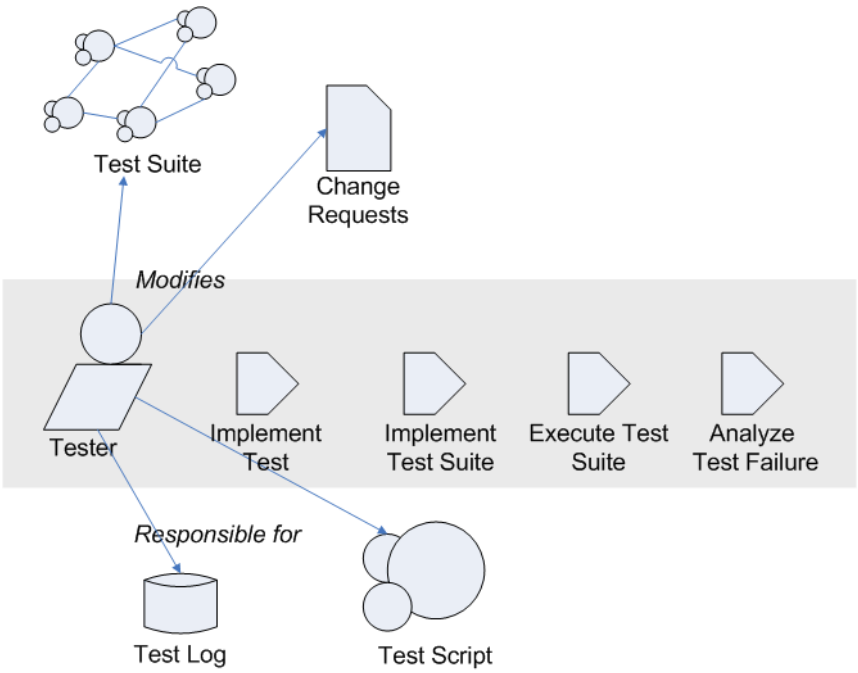

Figure 4: Concrete example for worker, activities, artifacts and workflows [32]

2.3.3.3 Dynamic Structure: Phases

The RUP is divided into the following phases, which all end with a milestone and can each consist of several iterations.

At the Inception phase the project starts. Setting goals for realizing the product during the project, requirement elicitation and creating the initial use cases are key tasks in this phase. A vision document contains the core requirements, key features and main constraints. Additionally the initial business case, risk assessment and project plan are created, and if suitable a prototype. If the milestone cannot be met, the project may be cancelled or re-planned. [32]

During Elaboration the analysis moves into design to create a proper architecture. Objective should be to gain a better understanding of the product to be developed. The end of the phase is concluded with an important milestone as the theoretical engineering becomes the high-cost and thus high-risk activity of actual implementation. Therefore it is essential that the vision and architecture are sufficiently described and stable. The list of risks as well as the business and use cases must be revised to the latest knowledge. [32]

The Construction phase is the manufacturing process in which the actual, deployable application is developed. The quality management triangle consisting of cost, schedule and quality is in the focus of management and controlling activities. Ultimate goal of

this phase is to create an executable software product that can be even handed to end-users. The question that needs to be answered to complete the milestone is if the release can be handed over to end-users or if another iteration will be necessary to create a mature and stable product. Goal of this phase could be to create a software version, which is mature enough for a beta test with a closed community. [32]

During the Transition phase the development of the internally identified workpackages has been finished. But when the software product is exposed to the end user new defects and change requests are likely to emerge. A user manual and end user training are important aspects of this phase. The phase concludes with the product release that concludes the project and hands the software product over to the customer. [32]

The RUP's process structure is organized horizontally along time and vertically along content (Fig. 5).

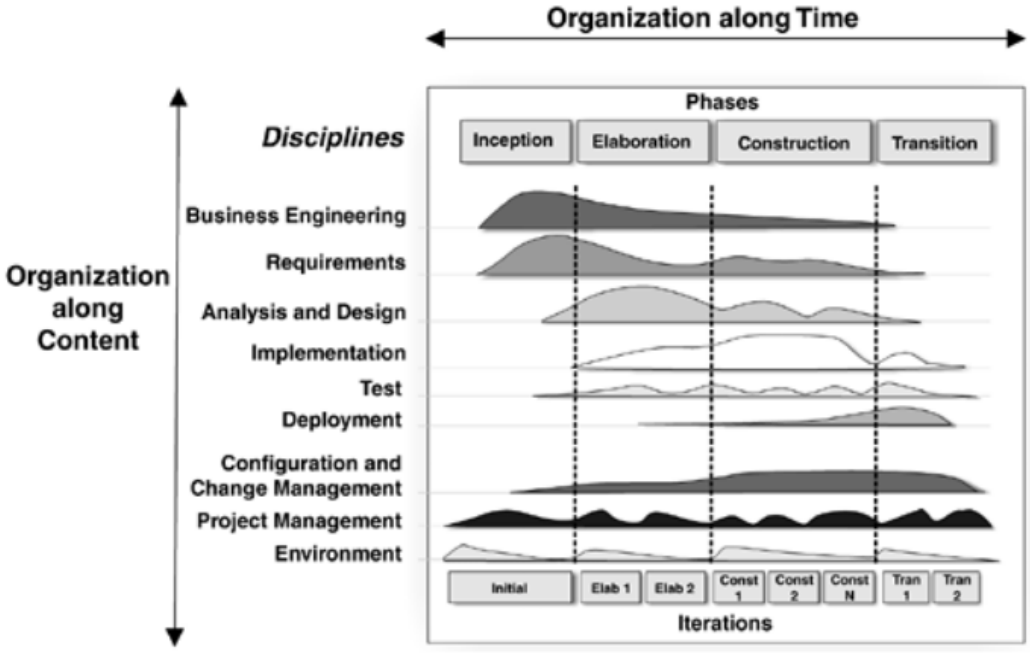

Figure 5: The two dimensions of the Rational Unified Process' structure [32]

2.3.3.4 Testing

Testing in the RUP is a separate so called discipline, which means that it has at least in theory the same importance as implementation or deployment.

Testing starts early in the process and is separated into different points in time, which are unit test, integration test, system test and acceptance test.

The RUP also enumerates the types of test, for example benchmark test, configuration test, or regression test.[34]

The RUP has a testmodel, which explains in more detail the artifacts of the test process. Below diagram (Fig. 6) shows the elements of the testmodel for the concrete example of an ATM system.

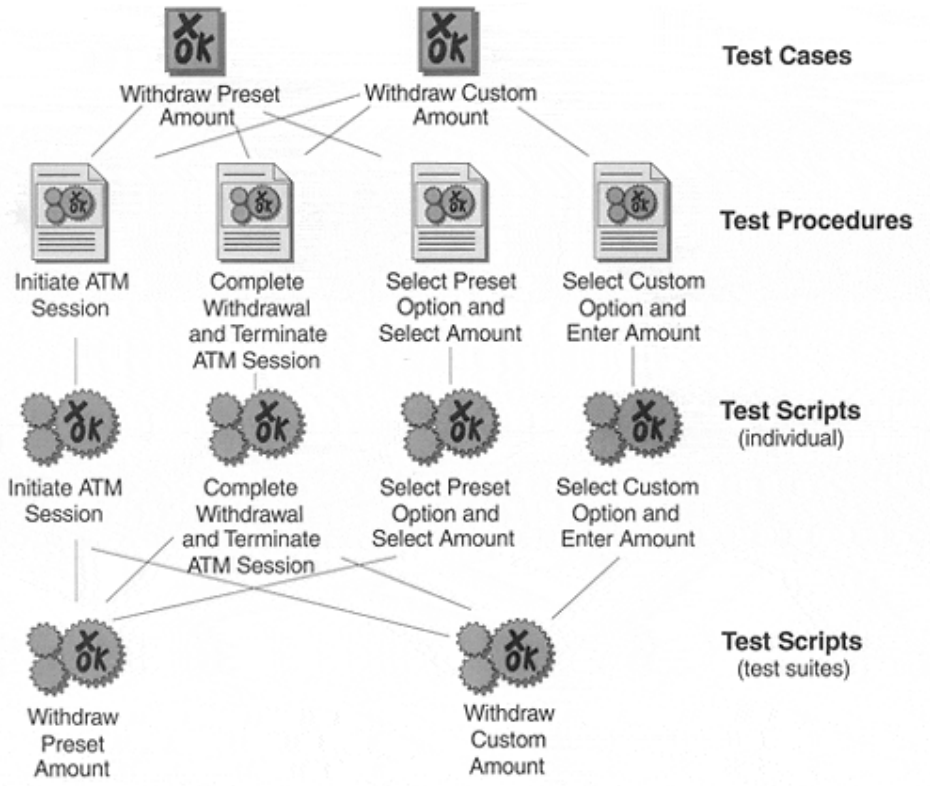

Figure 6: Test cases, test procedures, and test scripts for an ATM system [32]

The worker/role "Test Designer" is responsible to create the test model, with the test procedure as well as the concrete test cases. The test designer also writes the project management related test plan.

The test designer is supported by the developer roles of Designer and Implementer to create automated scripts for the test process.

The tester is then only responsible to deliver test results. As roles can be accumulated at a single person of course a senior test engineer would take both roles of test designer and tester. But theoretically it is also possible to have highly skilled test designers that do most of the brainwork and a cheap team of pure testers that only perform tests according to highly specified tests.

Kruchten specifies a strict waterfall-like process for testing. The test is planned, designed, implemented and ultimately executed, but for every iteration. There is need for automation and tool support as tests need to be re-run several times throughout the product development. In essence testing is seen to be a “feedback mechanism for the project, enabling quality to be measured and defects to be identified.” [32]

2.3.3.5 Conclusion

The RUP contains most of the components of a modern development approach. The system is developed in iterative and incremental steps. There are fixed roles that the team members should fill out.

Although the Rational Unified Process can be tailored to the individual needs, it is more suited to larger projects. Small teams of only a handful of participants will easily get lost in the challenging task of selecting those components they really need. “ ‘[..] will we need Configuration audit findings artifacts? Will we need a Change control manager role? Not sure, so we better keep them just in case.’ This may be one of the reasons why RUP implementations often end up quite heavy-weight compared to Agile methods such as Scrum and XP.” [35]

2.3.4 V-Modell XT

The V-Modell was designed to be a guideline for the planning and implementation of software development projects. In the V-Modell software development artifacts are called "products". It itemises the products and gives suggestions, which products should be created throughout a project. Additionally it also describes how the products can be created. By using standardised methods the V-Modell attempts to run even complex projects in a systematic, planned, and traceable manner. Intention is to increase reliability and quality of the process.[36] The V-Modell XT offers concrete templates for the documents that need to be created.

For the German federal administration and the federal authorities the V-Modell XT is the binding process model for the development of IT systems. [37] Therefore the V-Modell XT is an important process model in the central European DACH-region: Germany, Austria and Switzerland.

The V-Modell XT does not only take care of the actual development process, but supports throughout the whole project lifecycle (Fig. 7).

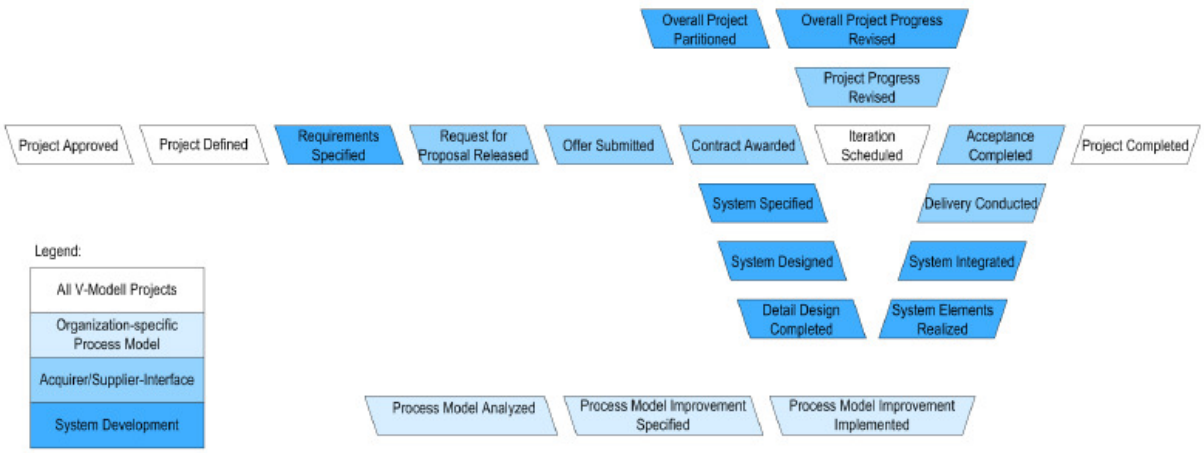

Figure 7: V-Model XT [36]

As depicted by the legend white trapezia are valid for all V-Modell projects. The brightest blue trapezia on the bottom are from the organization-specific process model. The bright blue trapezia within the project are connecting the acquirer and supplier side. Only the dark blue trapezia are the actual development related topics.

The term V-model is derived from the original V-Model's shape which puts to every integration step an according verification and validation step. Today the process model's system development structure is still V-shaped (Fig. 8).

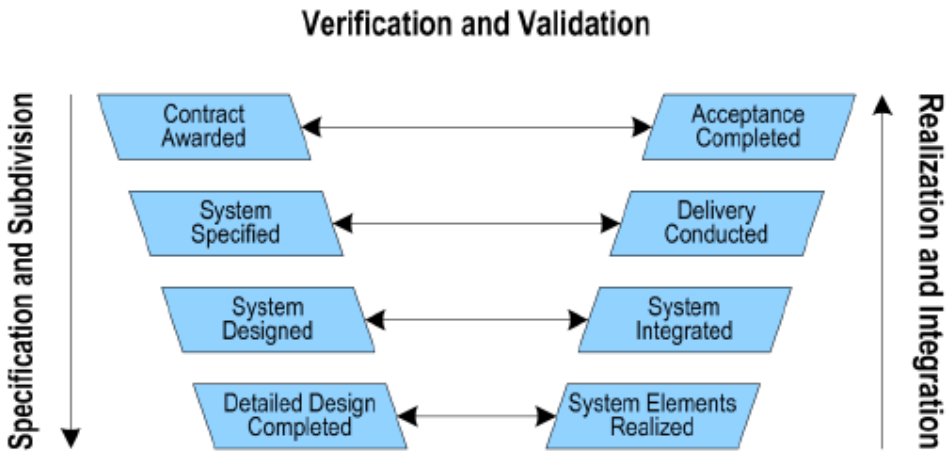

Figure 8: V-Modell XT System Development Structure [36]

Verification ensures that the project's requirements are met by the system. Typical verification methods are tests and peer reviews.

The purpose of validation is to check if the development efforts meet the specified requirements and the implemented functionality fulfils its task in the target environment. Depending on the results it might become necessary to change either the design or the requirements.

2.3.4.1 Tailoring

Tailoring is the activity of adapting the generic V-Modell to an actual project. Like selecting tools from a toolbox first the project type and project type variant are selected. Then the applicable process modules are chosen. Several of the modules are available not just for the supplier, but also for the organization acquiring the project outcome. The supplier will be the party in a contract that is obliged to deliver a system. That system is than either accepted or rejected by the acquirer. So the V-Modell XT specifies activities and responsibilities for both parties in the contract (Fig. 9). [36]

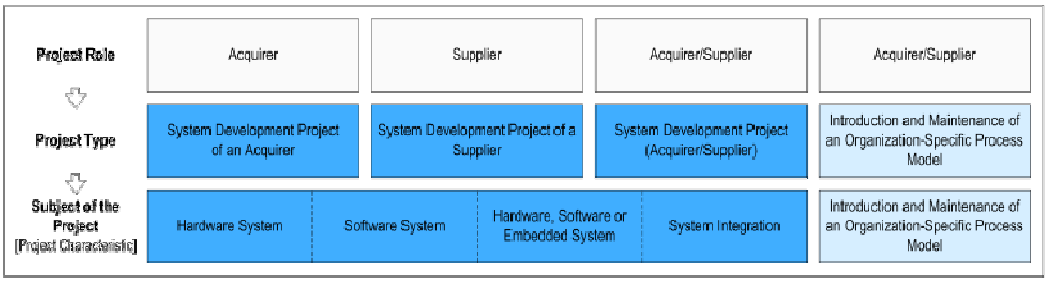

Figure 9: Classification of Projects and Subdivision into Project Types[36]

A concrete example for tailoring would be that there is a project, which will acquire a system from the supplier, who is successful in a tender. The project type will be “System Development Project (Acquirer)” then. A suitable project type variant will then be “Project (Acquirer) with One Supplier”. Due to the project type several process modules like “Project Management”, “Quality Assurance” and “Specification of Requirements” will be mandatory.

2.3.4.2 Process Module

A process module contains roles, products and activities to cover a certain task. They are a self-contained unit that can be changed according to the project’s needs. Its contents, which are roles, activities and products, are discussed in the following subchapters. [36]

2.3.4.3 Roles

The V-Modell XT defines and explains all roles that are needed in the software development process, and apart from roles such as Software Architect, also contain Account Managers, Controllers and even the Users.[36]

Every role is generally described first. Then a list of tasks and responsibilities the role will have to fulfil are given. The role description also gives details on the skills that are required by a person suitable for that role. The role allocation is also specified, giving hints if this role should be from the internal team or an external organization and if this is a role that should be present in any development team. E.g. a project manager can be seen as mandatory.

The role description is interconnected to the products and activity descriptions.

2.3.4.4 Activities

The V-Modell XT describes the actual activities that need to be performed by certain roles to create a certain product.

The activities of the V-Modell are hierarchically structured. The highest level is the disciplines, which group related activities. Those disciplines can be roughly separated into Project, Development and Organization.

Activities use the UML notification for their description.

There is a correlation between the disciplines for activities and for products.

2.3.4.5 Testing

The discipline Evaluation covers “all products and activities required for testing documents, system elements and processes.”[36]

The V-Modell stresses that it is important that the requirements of the system to be built already contain the needs and wishes of the client as well as the future users. It is only possible to run a test against certain evaluation criteria if they are properly described in the requirements. The requirements should contain the purpose, the desired performance characteristics and quality.

The classic V-Modell derives its name from the levels of development which are not only always input for the lower layer, but also have a link to an evaluation document on the same level (Fig. 10).

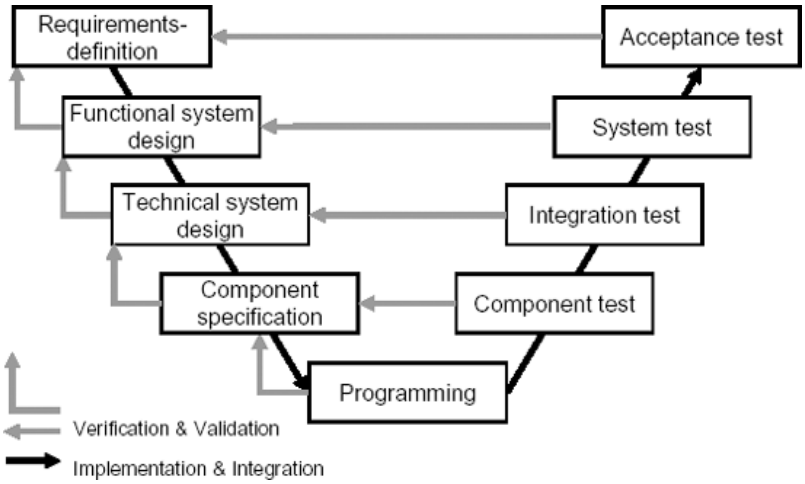

Figure 10: Classic / General V-Modell [38]

It can be observed that the requirements document is not only input for the functional system design, but also has the acceptance test as its validation on the same level.

Just like in the Rational Unified Process (RUP) the V-Modell knows different layers of testing, depending on the scope of test and the point in time the test is performed. The component test is similar to Unit test in RUP. It is also followed by integration test, system test and acceptance test.

As the V-Modell XT is a very descriptive process the tester gets a lot of concrete recommendations on how to test the configuration, documents, processes, usability and even delivery.

2.3.4.6 Products

Similar to the artifacts in the Rational Unified Process (RUP) Work Products are concrete documents that need to be created throughout the development process. They have a specific name and are explained in detail what they should contain. [36]

The work products of the V-Modell are hierarchically structured. The highest level are the discipline groups, which are Project, Development and Organization. Disciplines then group the products into meaningful chunks. [36]

For example the “Planning and Controlling” group of products consists of Project Progress Decision, Project Manual, QA Manual, Project Management Infrastructure, Estimation, Risk List, Project Plan, Work Order, and Life Cycle Cost Calculation (Fig. 11). [36]

Figure 11: Planning and Controlling discipline[36]

Products can be flagged as Initial and External product.

Note the “I” on the “Project Plan” product. The initial products are those which shall always be developed only once during a project.

The “E” on the “Project Progress Decision” product means that it is an external product – a product that comes from outside of the V-Modell XT project as input to the project. Although they come from the outside, the expected contents are defined in the V-Modell.

The Initial and External flag are not exclusive, meaning, that a product can be both or none, and just because a product is *not* initial does not imply it is external.

All products are described in the context of their discipline. The products are interconnected between their process module, responsible and participating roles and the activity that creates the product. To motivate the creation of the respective document a detailed explanation on the product’s purpose is given.

Products are usually generated by other products. The project plan for example generates many other products. Two or more products may depend on each other. This dependency can either be within the same process module, but also between products of different process modules.

The clear descriptions of the products are both an advantage, but also disadvantage of the V-Modell. The clear advantage is that in a team experienced with V-Modell XT its members will always know where to find the information they are looking for. All information is to be collected in specific documents and the terminology is always uniform so there are no uncertainties. The creating and sharing knowledge should be tremendously eased due to the structured approach of the V-Modell XT.

But the amount of documents is also a disadvantage. Looking at the agile approaches the amount of documents that need to be created is huge, and creating documents rather than the actual system under development is usually nothing that raises motivation in a team.

Working on the different documents can create problems with keeping the information current. Always representing the actual state of the project is a challenging task, especially for smaller teams. Bartelt et al. point out that a model based approach can mitigate this problem, as CASE Tools help keeping a consistent model with the actual code.[39]

Although the document centric approach of the V-Modell XT might be better suited for larger teams it has been also successfully applied to small teams.[40]

2.3.5 Scrum

Scrum also specifies roles, artifacts and clear rules, yet unlike the previously discussed RUP and V-Modell XT there is only a small number of each. Scrum does not try to extensively describe all possibilities, because its creators simply do not believe that you can think of all possibilities in advance. What has to be done in a certain situation needs to be determined by the people involved in the process according to their experience.

Schwaber uses the analogy of building a house when comparing Scrum to other process models. While the more conventional methods try to create a thorough plan upfront and build the whole house in one go, agile methods like Scrum build one room after the other. The plumbing and electrical are created for one room and if needed extended to the further rooms.[41]

Strictly seen Scrum is more a management process rather than a software engineering process. It can be used in conjunction with other engineering methodologies, which describe the actual software development tasks in more detail. [42]

2.3.5.1 Roles

Scrum defines only three distinct roles, who share the different management responsibilities within the project in a clearly separated way. [41]

As many agile processes Scrum puts the people in the centre of its thinking. The “Team” has many rights that are conventionally in the hands of a single project manager. The team is self-organized in Scrum, managing their own work. This is believed to be more efficient due to the high complexity of projects. The team estimates the efforts

for concrete tasks. The efforts (=cost) are a direct input to the product owner's prioritization. It's the team's responsibility to decide what work items from the so called Product Backlog will create functionality of a useful increment in the coming iteration. The team's decisions need to follow the Product Owner's priorities for the tasks. The team then commits to create a functional iteration during the next Sprint. The Product Backlog as well as the whole Scrum process will be discussed in the coming chapters in more detail. To be efficient Scrum teams are usually 7 +/- 2 persons.[43]

The "ScrumMaster" is similar to the project manager in other kinds of process models. Yet his rights and responsibilities are modified and somewhat reduced by the team deciding more on their own than usual. The ScrumMaster's responsibility is foremost to manage the Scrum process and not like in a traditional project to define and manage the tasks and individuals on the team. He or she must know the Scrum practices, meetings, artifacts, and terminology. But even more important he or she knows how to actually run a Scrum project. To learn the theoretical cornerstones of Scrum is a rather trivial task, yet knowing when a project is going well, or when it would need corrections needs experience. The ScrumMaster's task is to mitigate any problems and obstacles the team is facing and also to foster continuous improvement. [41]

The "Product Owner" is similar to the traditional role of the product manager, who focuses on the return on investment (ROI) of the project. The product owner should deliver the vision in a manner that maximizes their ROI. The responsibility is to administer the Product Backlog, with all the customer's requirements. The customer's requirements are usually defined in so called User Stories. Prioritization has to match the highest value to the customer's business in correlation with the cost. And if the business conditions change the requirements in the Product Backlog need to be adapted or extended. The Product Owner is also in charge of caring for the initial and continuous project budget. [41]

The "Scrum Team" stands for all those parties directly committed on creating the software, which are Team, ScrumMaster and Product Owner. These three roles are also referred to as "pigs". As opposed to this are the "chickens", which are people, who are only involved, but not committed, i.e. managers. [41]

2.3.5.2 Process and Artifacts

Scrum is an iterative process which outputs a functional increment in the end. The iterations are called "Sprint" in Scrum terminology and are recommended to last 30 days. Depending on team size and other factors this can vary between two to six weeks. Still every team should try to keep the sprint duration stable for their process, no matter how long it actually takes. [41]

Apart from the fixed milestone every month there is a second cycle that inspects the project's progress daily. Those 24-hour loops are called "Daily Scrum".

Below diagram shows the cornerstones of the Scrum process (Fig. 12).

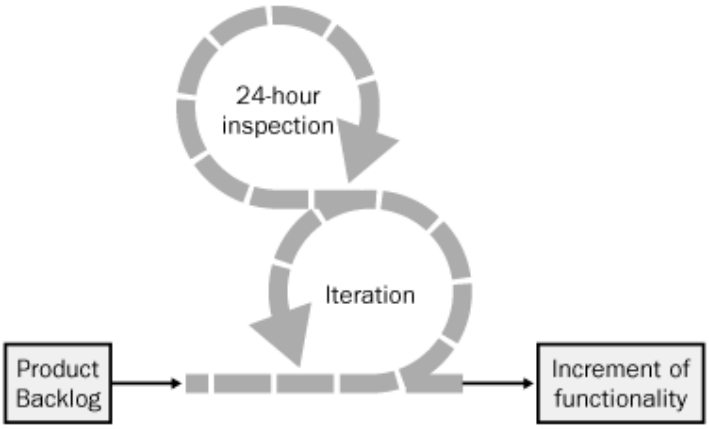

Figure 12: Scrum Skeleton [41]

The daily scrum should be a short (about 15 minutes) meeting that is held every working day at the same time. Just like with the sprint the time should be kept to the same amount of time throughout the project – "timeboxed". To keep the meeting short and productive it is often recommended to hold it as a stand-up meeting.

Every team member is expected to be able to answer three distinct questions:[41]

- What have you done since yesterday?
- What are you planning to do today?
- Do you have any problems preventing you from accomplishing your goal?

Goal of the meeting is to share as much information as possible throughout the whole team – everyone should know everything. Although a lightweight process "Scrum addresses the [.. CMM] KPAs for level 2 and 3 similarly, empirically and with a minimum of documentation and overhead."[41]

All requirements are collected and prioritized in the "Product Backlog". The efforts and thus costs of the individual features are estimated by the team. With this input the product owner can prioritize the individual items. The product backlog is not a static document. Its contents will be added, removed or adapted according to the market demands and as the environment evolves to keep the product competitive and useful.

Upfront to the next iteration the "Sprint Planning Meeting" is held. In the first part of this meeting the product owner explains the backlog entries to the team. Output of this

meeting is the “Selected Product Backlog”. Additionally a goal for the sprint is agreed. This will help to have appropriate criteria for acceptance of the increment. It is important to note that the product owner may only explain and set priorities. Unlike a project manager the ScrumMaster has no other role than moderating. The team directly talks to the product owner. [41]

The second part of the planning meeting is self-organized by the team. The team selects those requirements feasible for the next iteration and splits them into appropriate tasks. They themselves select the appropriate items from the backlog and commit to fulfil them during the iteration to create an increment useful to the customer(s). It is desired that members get roughly the same amount of work assigned. [41]

From this the list of actual tasks, called “Sprint Backlog” is derived. For the sprint backlog the items on the selected backlog should be split down into chunks that a developer or pair of developers can implement within 4 to 16 hours.[41] The sprint backlog is a means to make the development process transparent and as such is visible to all members of the project. The sprint backlog’s items also have a distinct person working on the task assigned. [41]

One important artifact of creating visibility in the Scrum project is the “Burndown Chart” (Fig. 13). It shows the amount of work done and remaining across a time axis. All project members can see the project’s progress at one glimpse. Also it allows grasping if the reality matches the original plans and schedules. [41]

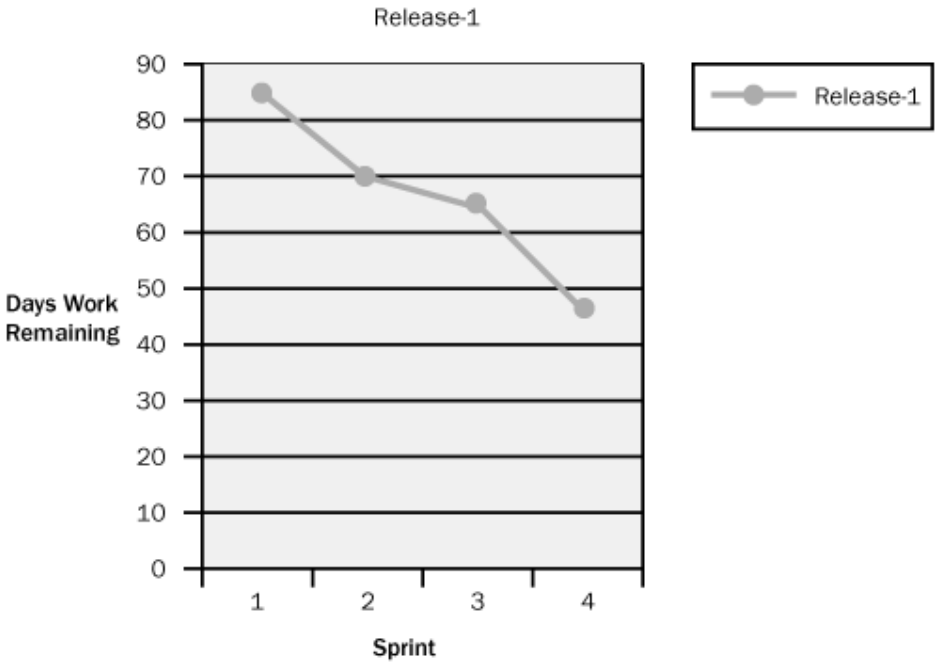

Figure 13: Burndown Chart (example)[41]

In the example above the remaining efforts are measured in days and given on the vertical y-axis. The sprints in which the efforts were used up are given on the horizontal x-axis. All four iterations are for the commercial Release 1.

At the end of every sprint a “Sprint Review Meeting” is held. Again there are concrete recommendations in Schwaber’s book, such as on how long the meeting should last. It is timeboxed to four hours and preparation should not take more than one hour. [41] The purpose of the meeting is that the team presents the increment to the product owner.

At this point it is important that the team as well as the product owner and optional stakeholders have the same understanding of the term “done”. While someone might assume done means, implemented, unit and acceptance tested, a developer might also assert it just means that the code compiles without errors. Open communication is important throughout the process, but even more in this meeting.

Of course functionality and artifacts, which are not done may not be presented as such. Especially stakeholders should not be lured in believing that unfinished aspects of the software might already work. Also stakeholders should not require understanding system development. Functionality should be presented on a server close to the production environment, such as a server in a test house.

Finally the “Sprint Retrospective Meeting” is a meeting held between the team and the ScrumMaster, with the optional involvement of the product owner. Its purpose is to look back at the previous sprint phase. “Together, the Sprint planning meeting, the Daily Scrum, the Sprint review, and the Sprint retrospective constitute the empirical inspection and adaptation practices of Scrum.”[41]

It is recommended that this meeting starts by everyone having to answer the following two questions:[41]

- What went well during the last Sprint?
- What could be improved in the next Sprint?

Those answers are written down and prioritized in what order the team wants to discuss them. Again the ScrumMaster’s role is not to give answers to issues, but just to facilitate this meeting. Output should be concrete things that need to change in the next sprint to make the process not only more efficient, but also enjoyable for the project members.

The requirements elicitation for Scrum projects is usually performed in the form of “User Stories”. [44] A user story tries to grasp the customer’s requirements in a very functional language. User stories have to represent the cutstomer’s perspective so tech-

nical requirements are less important and often do not make a useful user story. The future user of the system should be able to value the user story's content. A user story should include the description of the requirement that can be used for planning. Details of the story can derive from the conversations about the story. To determine when a story is complete tests that convey and document details are necessary. [44]

Often user stories are collected on handwritten cards that can be put on a wall or whiteboard so all members of the Scrum team can see them. Visibility is an important aspect of agile processes in which everyone needs to possess a lot of knowledge. [44]

The detail of degree for a user story should be measured on how long a developer would need to code the feature. One developer or a pair should be able to realize the feature between half a day and at most two weeks. It is perfectly acceptable to initially have only rough user stories, which are later split and detailed into useful chunks. User stories as all documentation should only be sufficiently reasonable and realistic – not everything needs to be written down in full detail. Certain details are also better explained as annotations from a direct conversation between developer and customer. User stories are living documents. They need to be adapted and extended according to conversations with the customers and users. Working with user stories can only work if the customer agrees to this kind of cooperation. A customer who wants to just give the initial project kick-off and then disappears will not be of much help. [44]

Main advantage of user stories over conventional requirements or use case documents is that the verbal nature of the communication between customer and team are emphasized. At the same time the user stories are comprehensible by both the technical team as well as the functional people on the customer side. [44]

2.3.5.3 Testing

Scrum, just like the other agile processes, is not very descriptive and does not specifically define how tests should be performed. Scrum is often accompanied by more concrete software development processes, such as Extreme Programming (XP) or Test Driven Development (TDD). [42] So, no Scrum specific testing practices need to be followed. Testing is seen as in integral part of the process, which the slogan of "sashimi" also tries to convey. Every sprint must create an increment that is, similar to sashimi, a fully operational slice of the product. A potentially shippable increment presented at the sprint review meeting must contain all steps (analysis, design, coding, testing and documentation) of the development process.

Scrum emphasizes that early tests will save substantial effort. “Scrum tests early and often whether the system being developed will deliver value and exactly what that value will look like.”[41]

Refactoring, the restructuring and rewriting of code, is an important aspect of agile processes. Refactoring depends on proper test coverage to prevent defects in the fixed area as well as side effects to other areas in the code. In other process models the testers can take the requirements and design documents, such as use cases, and create test cases from them. Requirements can be volatile in a Scrum project and often only apply for one sprint. As the agile manifesto specifies documentation is not as important as having running applications, detailed written descriptions of the design might not be available. In this scenario testers should try to “identify and prioritize key quality risk areas; i.e., they can follow an analytical risk-based test strategy.”[24]

2.3.5.4 Related Work

Several of Scrum’s principles can already be found in the knowledge management article by Takeuchi and Nonaka “The New New Product Development Game”. The journal article already features the analogy of Rugy to developing a new product. It also emphasizes the importance of understanding that modern markets are highly competitive and demand speed and flexibility. [45]

As a result it propagates the following paradigms. An iterative process is favoured over a strictly linear and sequential development method. Self-organizing autonomous teams work at their own dynamic order, and work towards their own set goals. With that in mind management only gives rather vague goals, and lets the team feel responsible. At the same time management controls only through defined checkpoints, but avoids rigid monitoring. Suppliers are involved into the process. [45]

Finally the Agile Manifesto that is the core of Scrum and most other agile software development principles is quoted below:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more." [46]

2.3.6 Software Kanban

Software Kanban is an agile development process that was influenced by the Toyota Production System (TPS). It adapts the general principles and not so much actual concepts for developing software. As a lean production system, it is based on five core principles:[47] Elimination of waste (muda), just in time (kanban), continuous improvement (kaizen), mistake-proofing (poka yoke) and worker/employee involvement.

Apart from mixing agile software development with lean production, Software Kanban also contains ideas from Lean Thinking, Theory of Constraints, and Product Development Flow.[48] [49]

Software Kanban is often seen as a lightweight approach to developing software. Unlike Rational Unified Process or the V-Modell XT the rules that have to be followed and the demanded artifacts are very small. What counts is to focus on the actual work and not so much on procedures around the topic.

With Software Kanban it is often possible to change the current development into an agile direction, without the harsh changes that for example Scrum demands. Software Kanban can be rolled out slowly over an existing development process. While Scrum can only be rolled out in a revolutionary approach, Software Kanban also makes sense if used evolutionary. And often the small steps can help larger groups of people getting acquainted with the unknown, thus raising the acceptance. [48]

What makes it possible to introduce Software Kanban evolutionary is the fact that not the final result, but the current situation is always in the focus. Software Kanban does not specify new roles or artifacts. Instead it tries to give best practices to shorten turnaround time, increase quality and improve reliability of delivery.

People can keep on working as they were used to, just with small modifications. And those small changes could even be easily undone if Software Kanban is really not the right model for the organization.[48]

Lean production tries to mitigate the usual human resistance to any change by pacifying the stakeholders with introducing only small changes.

Software Kanban shares with the other agile development approaches the desire to keep a continuous working pace, without the need for overtime or working on the weekends.[50]

Software Kanban tries to break down the big project into the small chunks of features. Every feature is then seen as an individual unit that needs to go through certain process steps, which can come from the currently used process. Usual examples are analysis, design, development, test and delivery. But Kanban itself does not describe what steps are needed and how the should be called.

2.3.6.1 Kanban Board with Tickets

A so called Kanban board, typically a larger whiteboard, visualizes the current state of the project for all team members. It is the pivot point for the whole project.

Figure 14: "Kanban"-Cover Comic

Above comic (Fig. 14) on the cover of David J. Anderson's book "Kanban"[51] shows such a board.

First swim lanes are drawn that separate the whiteboard into columns, which represent the different process phases. Each of the features, user stories, or generally artifacts will be tracked as a ticket, which is usually visualized by a sticky note. The ticket is then put on the Kanban board according to its state in the process. The further it moves to the right the ticket adds more value, thus the board also shows the value added chain of the project. [51]

2.3.6.2 Elimination of Waste and Flow

The requirements should always be realized with the highest value in the shortest amount of time possible. What does not work for this, or even against this goal, is seen

as waste. Waste needs to be mitigated. Typical examples for waste are queues and unfinished features, as both do not add value to the customer. They increase the turnaround time and increase communication and documentation efforts. [51]

The amount of tickets currently work in progress (WiP) is the fundamental benchmark of Kanban. Each step may only work on a limited number of tickets at the same time, as said above queues and half-baked features are useless to the customer. Software Kanban is a pull-system, meaning that only a team member, who finished a task, may take a new work package. This method of working also allows to easily spot bottlenecks, as it is obvious where tickets are piling up, while on other stations there are idle capacities. [51]

To mitigate such bottlenecks a root cause analysis can help find solutions. The higher the ticket's priority the more urgent it is to resolve the bottleneck. Such solutions can be buffers, or more people, shifting work to other areas, automating tasks or removing technical or other problems will have to be removed. If one bottleneck is loosened probably at another end a new bottleneck will exist, as now at the previous bottleneck more tickets can pass through. But there the same means can be used. This ongoing iterative process of continuous improvement is known as Kaizen, which is a fundamental aspect of Kanban. [51]

Although Kanban does not demand specific methods for the continuous improvement a daily standup meeting is seen as part of any Kanban implementation. The meeting should be kept short and discuss the project status referring to the tickets.

The WiP-limit is not specified by Kanban, but specific to the project team. An initial value could be `developers + 1`, so if there are three developers, four active work packages would be allowed. But also this limit should be part of the continuous process and re-evaluated regularly. [51]

2.3.6.3 Value Trumps Flow, Flow Trumps Waste Elimination

As said, Kanban does not try to give too many strict or binding rules. Above motto of lean thinking emphasises this agile approach. Although eliminating waste and guaranteeing a constant flow is important, still the value of the tickets is the primary objective. So, it is for example perfectly acceptable to break the WiP-limit, if a ticket needs to be finished earlier.

Tickets for features that will generate a high profit will be prioritized over normal tickets. That would be the case in any development process. But with the Kanban Board this can be modelled and visualized in a very intuitive and obvious manner.

High priority tickets will also be defects from the field, which yield high risks or concrete costs.

2.3.6.4 Testing

"In a strict interpretation of Lean there is no such thing as a tester." [42] The whole concept of lean is to prevent waste. Defects are waste. The objective of lean processes is not to identify waste, but to mitigate and ultimately eliminate waste through continuous improvement. So the vision is to be able to even prevent that defects are generated in the lean process.

Lean methodologies want to prevent defects altogether. Levinson recommends that testers should be involved as early as possible and start their work by reviewing the specification through peer reviews flanked by objective test cases. Those test cases could then even be signed off by the customer to increase their practicality. Code reviews will be useful to prevent actual programming mistakes. Having shifted most of the testers time to early steps in the development the tester's task during the construction of the software would really be of inspectors. If still a defect is found a triage process can help to prevent such an error to ever occur again.[42]

Although Software Kanban does not come with strict guidelines and rules, it is highly advised to stick to the rules the project team set for themselves. Only due to proper re-evaluation change e.g. WiP-limits is acceptable. Only this will guarantee a reproducible and trustable process.[48]

2.3.7 Conclusion

Rational Unified Process and V-Modell XT are widely used software development process models. They both share their rather heavy project management footprint and are usually seen to be suited, if larger teams need to create a system, like in the aerospace industry.

Although both RUP and V-Modell exhibit iterative development with feedback loops they still claim that it is possible to specify a large amount of the project details upfront. An example is that in the V-Modell XT the project plan is flagged as Initial product, meaning this artifact should be created once, and only once, in the project. Scrum and Software Kanban are much more centred on the assumption that each iteration has to be a useful product, and only then new requirements are considered. Agile methods accept that market conditions will change over the period of development and do not demand that the requirements should stay static, according to the plan.

As the name already implies the Rational Unified Process is a commercially available and supported process model. Opposed to this the V-Modell XT's documents and tools are available free of charge. Both process models make frequent use of the Unified Modelling Language.

In the traditional methods the architecture is defined once at the beginning, while in the agile methods a proper architectural design is stressed, but it may change due to changing requirements. In traditional methods the programmers are expected to be conservative and willing to follow the plan. Agile developers need to be team-oriented, communicative, flexible and pro-active.

In the agile world the customer needs to be open and ready for active participation in the project, while in traditional systems the role is of a passive nature. Refactoring is an integral part of agile processes, while code changes are generally more expensive in the classical models. The agile methodologies put the working software above everything else, including abiding to processes or creating documentation.[52]

All methodologies share some common elements that are base elements for the Global Joint Artifacts. Each methodology creates documents, source code and executable code that will be named "Artifacts". "Activities" are the small processes that generate those Artifacts. Without "People" no process can work. They are the human resources of the methodology. "Roles" are tightly bound to people, but can either be taken by a single individual or a group of people. Roles can also be accumulated at a single person.

The challenges given in this chapter can be summarized as follows:

Software testing is only part of a larger collaboration
Testers need to know a lot of information, and often very early in the project
Artifacts are created during the software development process and need to be organized
RUP & V-Modell XT create an abundance of artifacts
Scrum and Kanban depend on communication

Table 3: Challenges from Development Process Models

2.4 Global Software Development

Distance is relative in a world of global economy. Events occurring in China directly influence the markets in Europe, which later the same day influence the New York Stock Exchange.

This chapter discusses outsourcing software development and software test to foreign countries - offshoring. Please note that this chapter does not discuss IT infrastructure outsourcing. Eastern European countries as well as the so called BRIC countries (Brasil, Russia, India and China) have the comparative advantage of a large quantity of relatively well educated IT professionals, and yet still relatively low wages. This imbalance of cost and technical competence makes offshoring an interesting option. Another reason for outsourcing is to work with experts who are not physically located at the main team's site. Further assumed benefits can be leveraging time-zone effectiveness, modularization of work, innovation and shared best practices or the closer proximity to the market and customer for the remote team.[53]

Still distributed collaboration is a difficult topic, which requires cooperation in diverse fields to be of sustainable value. It is not just a business nor a technological decision, but also requires social skills and understanding. Carmel organized these requirements into six centripetal forces: collaborative technology, team building, leadership, product architecture and task allocation, software development methodology, and telecommunications infrastructure.[54]

In colloquial language contracting, outsourcing and offshoring are often used interchangeably. Carmel gives a taxonomy of structural arrangements for software development (Fig. 15).

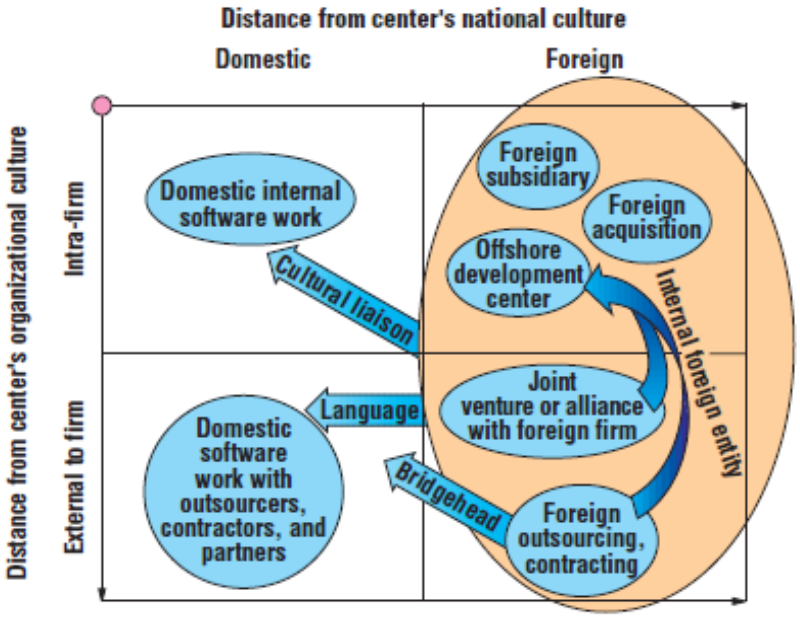

Figure 15: Taxonomy of structural arrangements for software development [1]

The diagram differentiates between spatial and organizational distance.

2.4.1 Temporal distance

Large gaps in the time zones of the distributed teams mean that at least one of the teams needs to change or extend their usual office hours, if they want to be able to synchronously communicate with their colleagues. Motivation will likely drop if people continuously have to work overtime, at night, or from home.

An alternative is to rely on asynchronous communication channels, like e-mail, which are indirect and not prompt. This can increase occurrences when the development and/or test department are blocked and let people feel “being behind”.[55]

Temporal distance can also be a benefit, when used in a follow-the-sun approach. There two teams are separated by a large gap in their time zones – like India and the US. After team 1 finishes their shift they give their work over to team 2, which then continues that work. In theory this can increase the performance of the two teams, but in practice needs a clear separation of concerns and/or a lot of communication to be fruitful.

Testing and fixing bugs is an example that can work quite well. E.g. a remote tester finds a defect and enters it in a bug tracking database. During the night the defect is fixed by a developer in another time zone. Once the working day starts again for the tester, the next build that claims to have fixed the issue can already be tested.

2.4.2 Spatial Distance

Spatial distance does not automatically mean temporal distance. For example outsourcing from Central Europe to South Africa is far in distance, but still in the same time zone.

The modern means of communication are helpful, yet should not be overrated. Obvious at first, but critical in the long run: the local team shares all resources. Even if in different departments people meet at work, on their way to work or during lunch. The local team always has a vast information advantage over a remote team. At the same time for the managers it is more difficult to control and coordinate distributed teams. Reduced communication normally also decreases the coordination and control effectiveness.[1]

A tool addressing the issues of distance has to mitigate the negative effects on communication, coordination and control. Thus, it will support the organizational units to contribute their input properly to the overall objective. It will give managers a way to control the process, by making goals, milestones, quality levels and the like visible. Clear, unambiguous communication of those items between sender and receiver will create common understanding. [55]

2.4.3 Cultural Distance

Any two persons interacting will soon realize that although there might be many similarities their opposite is also different person. When companies interact it is similar. A multinational organization like Siemens develops software different than a garage startup of five people. Even if they utilize the same underlying technology there likely will be organizational culture differences in respect to the methodologies used, how they are applied to the processes and the way project management is practiced. Such effects can be observed even when companies in the same region try to cooperate. [55]

On top of this offshoring introduces the differences in national cultures of the individuals working together. The spoken language is the first and foremost separator. Companies that value proficient English usually outsource to Ireland, India, Singapore or the Philippines. Larger organizations might even invest in language courses in the foreign countries. In addition to the language skills local norms, values, political or socioeconomic views might vary.[1]

2.4.4 Scrum and outsourcing

"Agile methods stress continuous face-to-face communication, whereas communication has been reported as the biggest problem of global software development." [56] The important question is if Scrum has positive effects on outsourced contract work or puts additional pressure on the internal and external teams.

The agile methods see change in the business environment and the requirements as normal. Agile methods usually try to mitigate the problem of traditional software engineering, which often struggles with specifying everything in detail in advance. Instead agile methods try to specify smaller chunks of the original problem and build the software in iterations. Focus is on direct communication and not so much on detailed documentation. Scrum stresses the importance of communication within the team and also with the customer. Making communication an explicit necessity also helps in distributed development and test. [56]

Both Paasivara [56] and Schwaber [41] describe that the direct channel of face-to-face communication can be substituted with instant messaging, IRC, or teleconferencing. Those channels allow quick answers with immediate and synchronous feedback, just like a direct meeting. The emotional component of communication can be strengthened by holding the daily Scrum meeting using videoconferencing. This not only improves motivation, but also helps building trust. At the same time open communication can help mitigate the socio-cultural gap between the teams.

Traditionally when working with distributed teams the whole project would have been specified in advance, split into modules, and every team would only work on their module. Only in the end the independent modules would have been combined in a big bang approach. Scrum makes it explicit that integrating smaller iterations of the software continuously is more efficient in the long run.

2.4.5 Sub/contractor and acquirer relation in Scrum

When the need arises to outsource testing a professional services contractor or subcontractor needs to be selected. Usually the selection can be condensed to a triangle of three criteria: A low price, an early delivery date and reputation. [41]

For the acquirer a fixed-date, fixed-price contract is usually the most appealing option, as it allows clear calculation of time and money. Using Scrum creates friction in an environment of bidding for tenders or responding to requests for proposals. Scrum strives to be the art of delivering the possible in increments. This does not meet the usual acquirer's expectation of getting detailed plans long time in advance and the product in a big bang approach.

As contractor the only option is to emphasize the positive effects of Scrum and trying to understand the acquirer's requirements to a point were an informed offer can be made.

2.4.6 Test types to outsource

Once a contractor is chosen the next important question is what can be outsourced. Theoretically anything can be outsourced – it only depends on the set objectives and expectations. Typically cost reduction is the objective and for this goal it is best to start simple and outsource structured tasks, which do not need the high degree of communication as new product development would demand. [57]

Tests can be outsourced easily that verify relatively stable components. For components that are currently under heavy development the immediate feedback to and from the developer will be beneficial. The remote tester on the other hand will not be able to sit together with the developer to give and get direct input. Also for a relatively stable component it is at least internally clear what should be tested, so the task description to the outsourcing company will also be clearer. [57]

Also tasks that are required, but anyways cannot be performed in house are good candidates for simple outsourcing. Especially for smaller developer companies this means to outsource anything that requires a large quantity of resources. Compatibility testing needs a test lab with many different software and hardware configurations. Localization testing requires testers, which are also proficient in the supported application languages.[57]

Technical tests can be outsourced more easily than functional tests. The remote testers are usually skilled engineers, but cannot be proficient in any functional field. For the remote tester it is more practical to check if a database entry exists, than to check why the calculation in the business logic results in a specific value.

The farther the team is away from their leader the stricter the rules and guidelines given are usually. Still, this is against the principles of Scrum, which believes in self-organized teams that pick up responsibility for their work.

Usually several topics are covered in written form between the company outsourcing testing and the contractor clarifying internal and external responsibilities. Depending on the trust between the parties the tasks at hand are either given in the form of a rough framework or in detailed steps. Someone in the internal team must be responsible to define those tasks, support the external team and then approve the results.

Schedule can either be given strictly top down, with flexible goals or defined by the external team. Risk management is performed to prevent risks from becoming a problem or mitigate their effects in the event of a risk turning into a problem.

The deliverables given to the contractor have to be defined as well as additional documentation or even non-standard peripherals that are required to test the software. At the same time the deliverables from the contractor, when they are delivered, their amount and the means how they are delivered need to be fixed. For example it should be defined how bugs have to be submitted. Same goes with the communication. Meetings have to be defined and who is responsible to hold them and what information should be shared. At the same time a confidentiality or non-disclosure agreement will be necessary to protect intellectual property.

Finally as with every project there should be goals defined, and how those expectations are evaluated.[57]

2.4.7 Conclusion

Global Joint Artifacts helps in spreading information (goals, deadlines, specifications, test cases, …) on the system under test between the internal and external team. At the same time it supports to give out tasks, keeps track on the current work and controls the overall process. It can be used as an asynchronous communication tool, giving self-service feedback on the artifacts' progress.

The challenges given in this chapter can be summarized as follows:

Sharing of ideas and thoughts needs communication
Distributed teams lack natural communction (e.g. in the hallway of the office)
To reach many, information needs to be broadcasted
Technology can help to reduce the amount of communication needed to spread information
A single person cannot know everything, and so has to collaborate with others to achieve their goal
Temporal distance
Spatial distance
Cultural distance

Table 4: Challenges from Global Software Development

2.5 Non-relational data storages

This chapter discusses the challenges that are created by large quantities of connected data and investigates alternatives to the traditionally used relational databases.

It is assumed that the Global Joint Artifacts system must be able to handle self-organizing users. “[..] self-organization is the spontaneous formation of well organized structures, patterns, or behaviours, from random initial conditions. The systems [..] possess a large number of elements or variables, and thus very large state spaces.”[58]

Actual performance data is presented in the chapter 5 “Results and analysis of the prototype” during smoke tests of the Global Joint Artifacts system.

For many years relational database management systems (RDBMS) were the predominant means to store data. The term NoSQL summarizes several different technologies to store data, which all share the common attribute that they are not RDBMS.

These alternative technologies allow improvements in reliability, scalability and performance in distributed environments, yet sometimes even lack the traditional functionality of RDBMS like the traditional atomic, consistent, isolated, and durable (ACID) guarantees. Atomicity means that a transaction is only executed as a whole or not at all. If any part of a transaction fails, all previous steps are also re-rolled, so the database stays unharmed. Consistency means that transactions will keep the database in a consistent state, usually through the use of primary and foreign keys. Isolation means that if a transaction modifies data, it cannot be modified by another transaction at the same time. Durability means that the result of a previous transaction will be incorporated in the next modification of the data.

2.5.1 NoSQL technologies

The relational persistence is based on tables, columns and rows. Details on a selection of alternative NoSQL data persistence technologies follows below:

2.5.1.1 Key-Value Stores:

As the name implies entities are stored in the form of key/value-pairs. This allows storing arbitrary data that is indexed by a key. That key is the unique identifier and the value the data to be persisted.

For a large collection of web pages the row key could be the URL, as it is a unique key. Individual attributes of the web pages then would be stored in column key/value-pairs.

Implementations like BigTable and Cassandra group the individual columns in so called column families. Column families logically and physically group the columns. Column families are used for access control, disk and memory accounting and compression.

The actual key to get the value is then comprised of the row key, the column family and the column key. (Fig. 16) [60] [61]

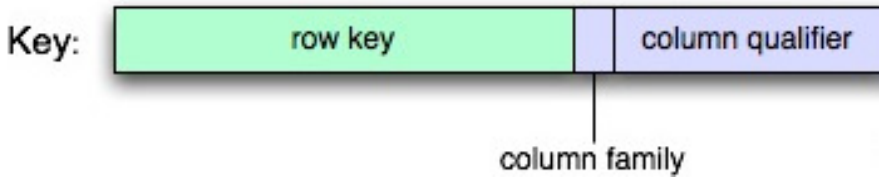

Figure 16: Schematic view on a key [59]

2.5.1.2 Document-oriented Databases

Document databases are usually schemaless. So it is possible to store name and address for one person, but first name, last name, birth date, place of birth, and address for another. In a classic schema-oriented database all those fields would have to be specified in the schema in advance. Then maybe only a small set of the data would use it. Alternatively it would not have been possible to save this additional data at all. Thus document-oriented DBs help to avoid only sparsely populated tables when working with semi-structured data. [62] [63] [64]

In relational databases data is usually stored over several tables and needs to be joined back for viewing. Joins are resource intensive operations. Document-oriented databases usually store the whole information as a single document reducing the need for joins. Usually they are dynamically-typed so the schema can evolve easier than in RDBMS. The relational model for persisting data differs vastly from the object oriented model used in many programming languages. Often object-relational (O/R) mappers are used to ease the integration of the persistence layer into the software. Documents, seen as objects, map more nicely into the data types of programming languages. [62] [63] [64]

2.5.1.3 Graph Databases/Repositories

One type of the non-relational data stores, which is exceptionally well suited for social networking applications are Graph repositories. Here information is not stored in the

relational tables of an RDBMS, but in its more natural structure of interconnected nodes, a graph.

Apart from storing data in their nodes and edges graph DBs provide the benefit of built-in graph processing operations. Usually they provide several methods to traverse a graph from a given node with configurable constraints to fine tune the operation.

The nodes and edges in a graph are more similar to the object oriented data model and as such programming against graph DBs needs less conversion between paradigms. Similarly it is possible to visualize hierarchical structures and networks with out of the box frameworks.

Due to their nature of being able to process large graphs and support data analysis graph DBs are exceptionally useful in the fields of bio-engineering and bioinformatics, semantic web, business and other intelligence, fraud-detection and any form of network research. [65] [66]

More details on a concrete graph repository, neo4j, can be found in the according chapter 4.5.6 “Neo4j”.

2.5.2 Data storage features

This chapter discusses the prominent features of data storages attributed with “NoSQL”. The different technologies do not necessarily offer all, but rather depending on their main focus more or less of these features. At the same time some of the features are now also available on RDBMS like MS SQL Server 2008 R2 or MySQL.

2.5.2.1 Partitioning

For scalability and reliability it is desirable to split large sets of data into smaller chunks - partitions. Cassandra has the ability to scale incrementally by dynamically partitioning the data across nodes in their cluster. [61]

2.5.2.2 Replication

Data is stored in multiple copies to improve fault-tolerancy. If one node fails the data is immediately available from another node. Replication can also improve access speed for the users if data is cached on different locations. In neo4j replication is currently only supported in a master-slave workaround of the backup functionality. The slave can either help increase redundancy by waiting for the master to fail and then taking over its work, or as a read-only slave that improves read performance.[67]

2.5.2.3 Sharding

Sharding usually means a combination of partitioning and replication to benefit from those methods for modularity, scalability and redundancy. MongoDB scales horizontally through sharding. The partitioning into shards works usually analogously to creating a primary key in an RDBMS. One or more fields make up the key to which the partitioning is performed. [68]

2.5.2.4 Semi-structured data

In relational databases there is a tight relation between the schema and the data. The schema enforces constraints how the data will look like. As such it defines what columns exist in any row and what data is accepted in each of the fields. Semi-structured data is either only loosely coupled to a schema or even self-describing. [69] So instead of a pre-defined rigid schema data can adapt more easily to changes in the environment.

2.5.3 Conclusion & application

Selecting a concrete product for storing Global Joint Artifacts' data out of the abundance of RDBMS and NoSQL solutions is a challenging task. Many approaches, technologies and concrete products compete for attention. For the prototype a conventional RDBMS would suffice. But using a graph DB seems to be the most "natural" approach to storing highly interconnected data that is often, but not necessarily, in a hierarchical relation to each other. Depending on their usage and purpose artifacts also store a flexible set of properties, which is already a clear shortcoming of rigidly structured RDBMS. More information on neo4j, which is used as the graph repository can be found in chapter 4.5.6 Neo4j.

Structure of interconnected data is a graph, not a flat table
Tracking all components in a project, or even several projects leads to a high amount of nodes in the graph
Connecting all artifacts with their owners, submitters and observers is multiplying the amount of connections tremendously

Table 5: Challenges of high volume interconnected data

3 Global Joint Artifacts Requirements

This chapter discusses the scenario and describes a solution to the challenges of the project. First it details the concept of a Global Artifact as a work item. Then a system to manage such Global Joint Artifacts in the scenario of outsourced software test is described.

3.1 Concept of Global Joint Artifacts

Global Joint Artifacts is a general concept, newly derived based on the five foundational chapters. (see chapter 2) A Global Joint Artifact is a tangible object that is interlinked with people or other artifacts. Artifacts can either be linked serial/horizontal or parallel/vertical. People connected to artifacts will receive status updates from the artifacts, analogously to their contact's status updates.

Usually a Global Joint Artifact is a work in progress item or a completed piece of work. Often they are created by globally distributed teams in a cooperative effort or are part of such a venture. Examples range from physically existing documents such as a project plan, use case description, test case or user story to items such as the project itself or its increments. A Global Artifact cannot be intangible such as customer satisfaction. Yet a Global Artifact could be a project to raise customer satisfaction with all the necessary steps and artifacts.

Artifacts are linked serially (or also known as horizontally) if they are connected for the items in the same project hive, like project->project plan->test plan->test case design->test case->test case step->test protocol.

Artifacts are linked in parallel (or also known as vertically) if they are connected for similar items, like test case design project G->test case design project B. Artifacts should not be linked just because they are of the same kind (e.g. all test case designs) as this clutters the graph unnecessarily, but only if they really resemble a similar topic. An example for useful parallel linking would be if both test case designs are for the same area, like for a setup program. It will be possible to derive knowledge and re-use findings easily from such a semantically related document.

In their case study on the impact of Scrum on overtime and customer satisfaction Mann and Maurer identified "two problems associated with the rapidly changing requirements. First, there was a very poor picture of what work was going on internally and what work was planned for future completion. Internally, there was no place that a developer could go and see an up-to-date comprehensive picture of what needed to be

completed and when it was expected in relation to other pieces of functionality." [50] "The second problem associated with the rapidly changing requirements was there was no control put in as to what the developers were working on. If a stakeholder suggested something in the weekly meeting it was very likely the developer would add what was requested to what they were working on without consulting anyone."[50] They also state that the customers had their problems with the Scrum process in understanding at what tasks the developers were working. The Scrum planning sessions were not suitable to customers in the beginning and they needed the project manager inbetween the sprints. (see chapter 2.3.6 Software Kanban).

Global Joint Artifacts directly solves these issues by making tasks and their relations to people and other tasks obvious. People connected to artifacts get notifications on their status updates. For example if a test lead creates an artifact representing a specific test run and assigns it to a tester, the test lead will get updates, such as if the test run completed.

The following chapters discuss the system that allows managing people and artifacts.

3.2 Requirements analysis and object-oriented analysis

This chapter discusses the requirements identified by the fundamental research as well as the practical evaluation of similar and peer products of the Global Joint Artifacts application. It splits the system into the functional components.

3.2.1 Stakeholder Identification

Stakeholders in the concrete example of using Global Joint Artifacts for outsourced software test in a Scrum environment are the involved parties, which are the ScrumMaster, product owner, local team, remote individuals or a team of a subcontractor, acquirer, end users and suppliers.

3.2.2 Use Cases

The use cases show how to use the system and its actors.

Some selected use cases are modelled in more detail. The activity diagrams show the workflows, the sequence diagrams the messages sent to and through and the state machine diagrams how the system behaves in its different states.

3.2.2.1 Node operations

Every node, both persons or artifacts, has a certain set of operations attached to them (Fig.17).

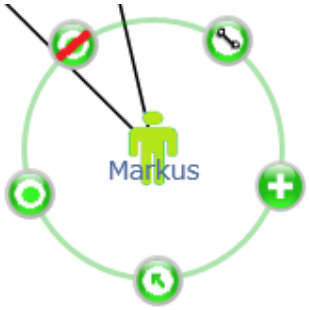

Figure 17: Node actions

Node actions will be normally hidden. They are only visible after a mouse-over and then circle around the current node. A left-click is not needed.

The following describes the individual node operations.

3.2.2.2 Node operations: Expand / collapse node

The „+“ symbol suggests that the node can be expanded and will reveal more attached nodes. Respectively the “-“ symbol will allow to collapse the current node, hiding all child nodes down that part of the graph.

3.2.2.3 Node operations: Focus node

A node must be focused to view the node’s information, change or delete parts of that information, add new properties to the artifact and enter efforts to the node for the current user (Fig. 18).

Figure 18: Focused node properties

It is only possible to enter efforts for artifacts, and only for artifacts, which are owned by the currently logged in user.

The "id" property cannot be changed as it is a fixed system value. Also such a value would not be visible at all in a commercial application, as it servers no purpose to know the neo4j-internal ID. Tracked efforts are accumulated from all connected contributors and automatically summed up. Owner, submitter and observers are dynamically retrieved from the edges, and not directly stored on the node.

3.2.2.4 Time tracking

Companies as well as individuals often track their time in custom applications.[50] It is assumed that people won't use a new system for tracking old items, but rather for things which are under construction. It is assumed that time tracking will be an important part, as it is always necessary to see how much time was spent on a certain artifact and also how much will be billed (Fig. 19).

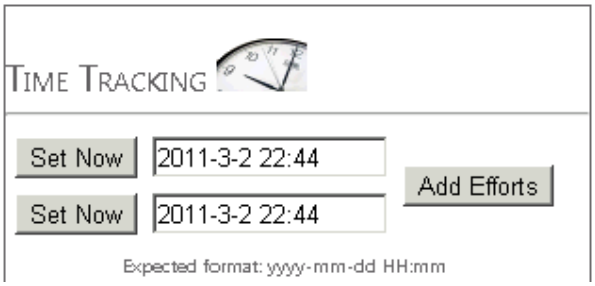

Figure 19: Time tracking pane in Global Joint Artifacts

Start and end time can be retrieved by JavaScript, but can also be modified normally by keyboard.

It is only possible to enter efforts for focused artifacts, and only for artifacts, which are owned by the currently logged in user.

3.2.2.5 Node operations: Add node

Adds a new node to the graph, which will be linked to the currently focused node. The new node is then focused.

3.2.2.6 Node operations: Delete node

Deletes the currently focused node, and all its links to other nodes. This also means that all efforts related to this artifact will be lost.

3.2.2.7 Node operations: Link node

Links the currently focused node to another node.

3.2.3 Object-oriented Analysis

Here the objects are identified and similar objects combined to classes. The classes are further described with their properties/attributes and abilities/methods. Then connections and dependencies between the classes are identified. In this step the model is the

only important aspect, while technical details from controller or view are discussed later in chapter 4.4 “Detailed Design”.

For the static aspects class and object diagrams are used. For the dynamic aspects state machine diagrams and sequence diagrams are applied.

To identify classes nouns are used. Nouns are also used for the classes’ attributes. Methods are named with verbs.

Following the bare classes are described that were identified (Fig. 20).

As mentioned earlier the system Global Artifact should be able to manage artifacts. So “Artifact” is a base class that will contain arbitrary data. But all artifacts need to be identified somehow, so a unique “Id” parameter seems useful. As artifacts are tangible objects they will nearly always have a physical body, e.g. a document available on a versioning system. So another parameter that identifies the underlying resource will become useful. The term Uniform Resource Identifier (“URI”) seems to be a well known term for identifying resources. It will take any kind of resource identifier such as “Main Building / Floor 3 / Room 302 / Cupboard 8 / Folder Project ABC Blueprints”, “\\servername\share$\project\concreteDocument.doc” or “https://svn.somewebsite.com/project/concreteDocument”.

As the URI will be difficult to read for humans an additional value “Title” will make sense to quickly identify the artifact, e.g. as “Setup Testcase” document.

People will need to interact with the artifacts. So a “Person” is needed. Regarding “Id”, “Title” and “URI” they share the same attributes.

As such a common base class seems to make sense that already encapsulates the attributes.

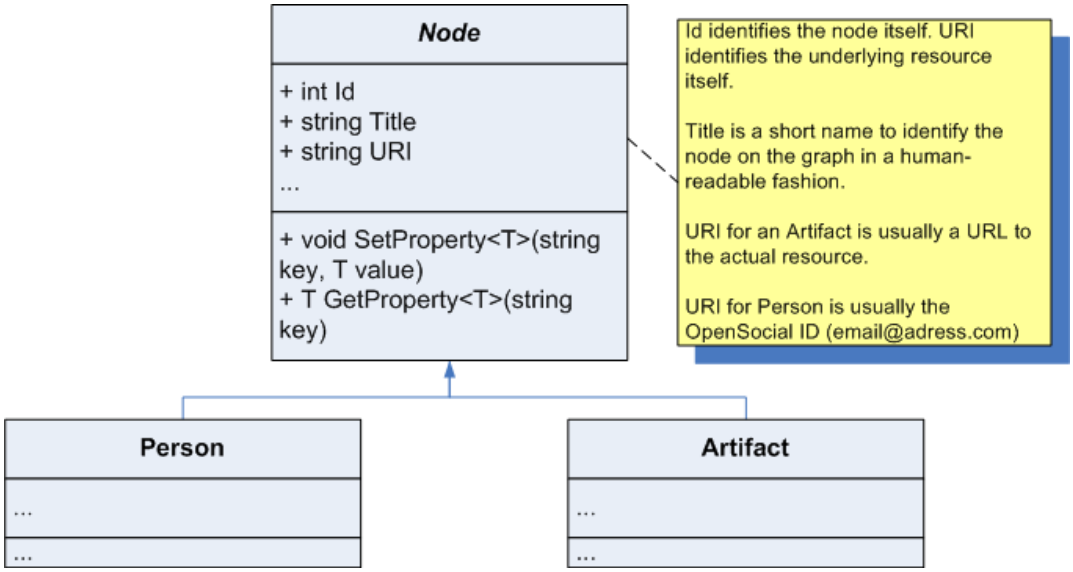

Figure 20: Class diagram of Node-Person-Artifact relation

3.3 Summary

Knowledge management can be seen as the very core of the Global Joint Artifacts, which are a generic tool for organizing knowledge. The research mainly looks at scientific papers in this field.

The concrete use for this project in this thesis is software testing, a discipline of software engineering, so a discussion of project management in software development projects and development process models is mandatory. Out of the many development processes Scrum will be the one used and implemented in most detail. Input for these chapters are mainly industry standard books.

The concrete use case is further refined for outsourced teams, in which a web-based tool is supposed to be of most value.

The foundation chapters explain the areas under investigation and give an insight where the solution gets its input from. That initial part is then a direct input for the requirements engineering of Global Joint Artifacts.

4 Global Joint Artifacts - Evaluation, Design and Documentation of the Implementation

This chapter summarizes the evaluation for and the design of the prototype. The design influenced which tools and methods were evaluated. Iteratively the design was refined with findings from the evaluation going back and forth between evaluation and design.

Out of the theoretical findings in the design and the practical findings of the evaluation the implementation was created. The final design and specification serves as the documentation for the implementation.

The following chapters abstract the problem and try to modularize it into detail of the Global Joint Artifacts prototype.

4.1 Architecture and Top Level Design

This chapter is a logical view on the structure of the system under development. The component diagram in this chapter shows the composition of the architecture and its interfaces.

The Microsoft Windows and .NET platform were chosen as main development environment due to the existing knowledge and skills in the field. From a mere technical standpoint a Linux-Apache-MySQL-PHP (LAMP) approach could have been used as well.

It was decided to use VMware as the runtime container for the prototype. From practical experience it proved to be a test environment, which is easy to use and due to its nature offers a lot of freedom for experimentation and has powerful undo capabilities. Alternative containers would be physical hardware (a “normal” server) or a cloud service like Amazon’s EC2 or Microsoft’s Azure.

As operating system Windows Server 2008 R2 Enterprise with its IIS 7.5 web server were chosen as they are the latest production quality technology available from Microsoft.

For the presentation layer initially the content management system (CMS) DotNetNuke was evaluated. For the prototype and probably also an early version the normal ASP.NET framework will fully suffice.

For the prototype only minimal Scrum features were implemented. The next step would be to add a project management tool to extend the functionality. An open source package that uses .NET was evaluated. TeamLab fulfils most of the requirements and thus will add value to the Global Joint Artifacts.

The Global Joint Artifacts core stores and links people and artifacts. It uses the Graph storage Neo4j with a .NET wrapper. The graphs are visualized with Microsoft Silverlight using the Information Connections Engine (ICE) component. The core connects to social networks, which implement the OpenSocial API. The core is discussed in more detail in the following chapters (Fig. 21).

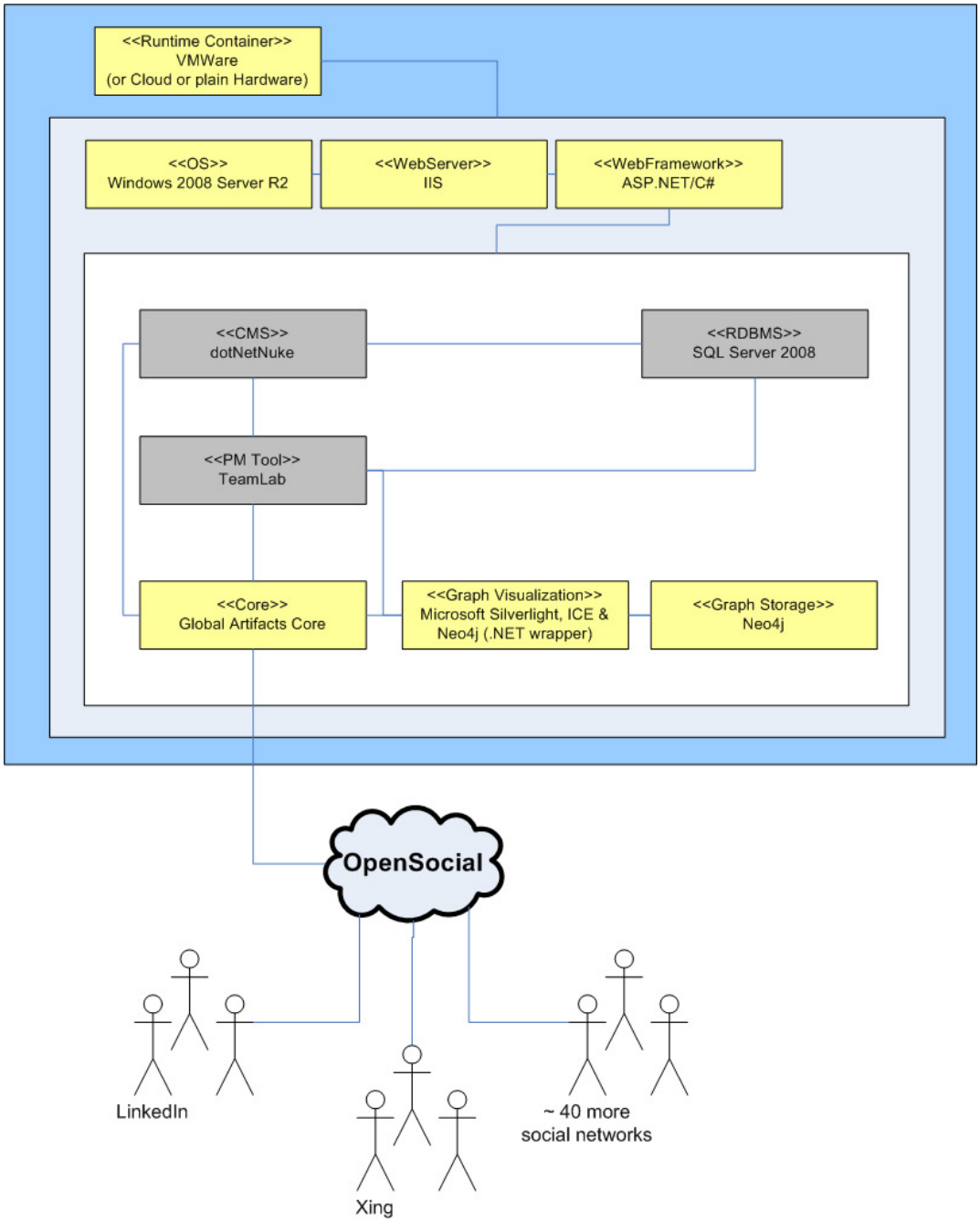

Figure 21: System Architecture / Component Diagram

4.1.1 ASP.NET three-tier architecture

Microsoft's Active Server Pages .NET is used with a three-tier architecture [70] as often found in the enterprise environment (Fig. 22).

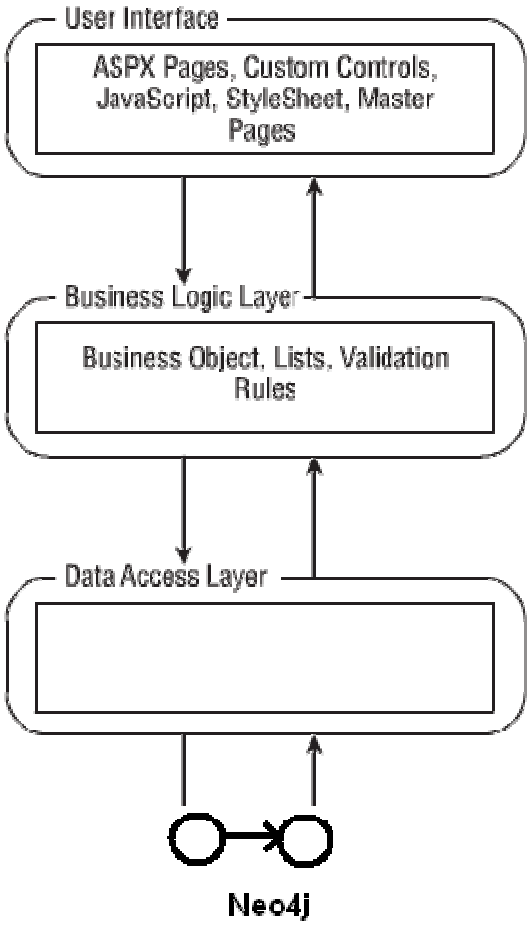

Figure 22: Three-tier architecture [70]

Alongside the three tiers (Data Access Layer, Business Logic Layer and the UI Layer) a framework for enterprise applications also needs proper exception handling, (role-based) security, a workflow engine, notifications capabilities, reporting including a query builder, a dashboard and in the long run also auditing functionality.

4.2 Component details

The state machine diagram shows the functionality of single components in more detail and how components interact.

4.2.1 Global Joint Artifacts OpenSocial Gadget

The gadget is a minimal application that exposes the Global Joint Artifacts functionality to the users of the social networks.

After the application is initialized it is in stopped state. Starting tracking creates a new begin DateTime-stamp for the artifact. If tracking is paused or stopped the end Datetime stamp is set. The difference between “paused” and “stopped” states is that resuming immediately sets the next begin Datetime, while from “stopped” first the artifact selection dialogue comes up. From the “Stopped”, “Tracking” and “Paused” states the application can be terminated (Fig. 23).

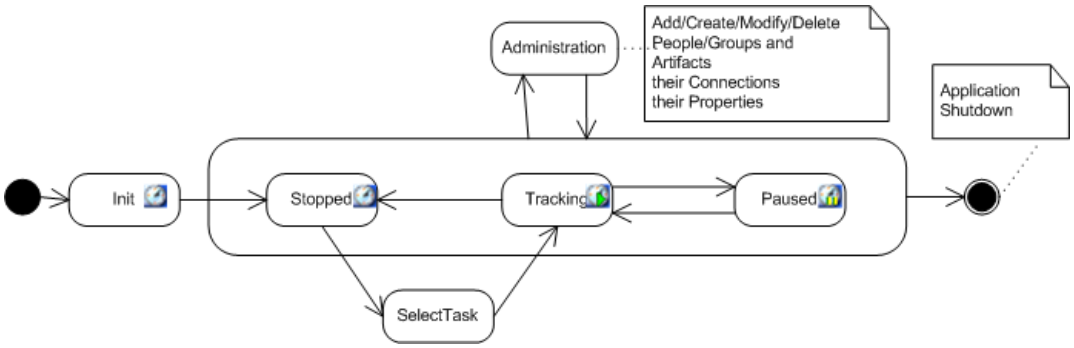

Figure 23: States of the application

To start tracking efforts for an artifact it needs to be selected from the project tree. Initially only the top level items for this user are visible. Once a node is selected its child nodes can be expanded (Fig. 24).

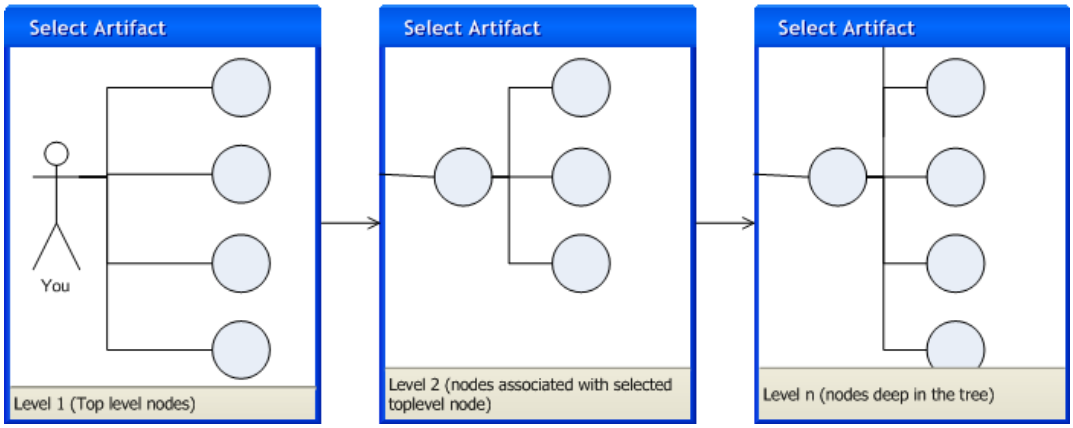

Figure 24: Artifact selection dialog (in successive states)

User administration allows adding users from any OpenSocial enabled network or create users on the system that might be linked later with an OpenSocial Id. Users’ attributes can be modified or users can be deleted from Global Joint Artifacts. Users can be grouped to ease assignment of or likewise to artifacts.

Artifacts can be created, modified or deleted. Typical tasks include adding child artifacts or related artifacts or assigning artifacts to new owners. Visibility and further properties can be set.

Artifacts and people can be linked to related artifacts or people.

Find more details on OpenSocial in the respective chapter 4.5.1 below.

The OpenSocial gadget is only available as design and is not part of the actual prototype implementation. The prototype is only available as the concrete social website, but not as the thin layer that an OpenSocial gadget would be.

4.2.2 Windows Client

To minimize the efforts for a windows client the client application does not have much functionality itself. Basically it is a tray icon that calls web pages through the default browser and thus is a shortcut for navigating the site. The user credentials are stored locally (if desired) and don't have to be entered when the website starts. From its usability it is very similar to the OpenSocial Gadget. A right-click on the tray icon (in the form of a clock) allows to enter the administrative options and moving from the states "Stopped" (idle) to "Tracking" to "Tracking Paused" (Fig. 25).

Figure 25: Windows Client (Clock icon) in idle/stopped state

The windows client is only available as design and is not part of the actual prototype implementation.

4.2.3 Compatibility Matrix

The following table shows the versions of the software that are compatible with Global Joint Artifacts and also the subcomponents requirements.

Global Artifacts 0.1	DotNetNuke 5.6.0	TeamLab 2.0	Neo4j 1.2.M05	Neo4j REST .NET 0.5	ICE 1.0.3	Silverlight 4
.NET 2.0	X	X				
.NET 3.5 SP1	OK	OK		OK		
.NET 4	?	?		OK	OK	OK
SQL Server 2005 Express	OK					
SQL Server 2005	OK					
SQL Server 2008 Express	OK					
SQL Server 2008	OK	n/A				
MySQL	Custom					
WinXP	X					
Win 2003 Server	OK					
Win 2008 Server	OK					
Win 2008 Server R2 (IIS 7.5)	OK	n/A	OK			
Win 7 (IIS 7.5)	OK					
Neo4j 1.2.M05			X	OK		
Neo4j REST .NET 0.5			OK	X	OK	
ICE 1.0.3				OK	X	OK
Silverlight 4					OK	X

Table 6: Compatibility Matrix Global Joint Artifacts

Horizontally all current versions that Global Joint Artifacts is comprised of are visible. The versions or even components might change over the life time of Global Joint Artifacts. On the vertical axis all theoretical or historical versions are shown for reference. The compatibility matrix only shows relevant combinations. E.g. it does not state what version of DotNetNuke is compatible with ICE as those two components are not coupled at all.

4.2.4 Hardware and operating system

An Intel i5-650 Dual Core CPU with a total of 8 GB RAM is used as the hardware platform. The prototype runs on a Windows Server 2008 R2 Enterprise 64-bit US, which runs within a VMWare Workstation 6.5. The VMWare uses 4 GB RAM out of the available RAM. The host operating system is a Windows 7 Ultimate 64-bit US.

4.2.5 Security

Basic means of security are enforced for the prototype. The operating system Windows Server 2008 R2 comes out of the box without any roles or features installed. All func-

tionality needs to be switched on manually, so unnecessary services, like a Telnet or FTP daemon/service, are not running.

The SQL server as well as the neo4j REST engine are configured to only run locally, as they are only accessed by the locally running Global Joint Artifacts core. Most functionality is only available through the web interface of Global Joint Artifacts, but not directly through remote ports.

4.3 Scenarios

The most important and critical use cases are discussed here in more detail. The sequence diagrams show how the participating components exchange information and the activity diagram in what order they work together.

4.3.1 Person/Artifact-Relations: Submitter

The original submitter is the person who creates an artifact. A concrete example scenario is a test lead assigning test runs for the next increment. Once created the test lead will assign the artifact/task to the responsible tester. Yet this test lead will stay submitter. There is always a 1:1-relation between artifact and submitter.

If artifacts come from a remote system (e.g. a bugtracking database like ClearQuest) the submitter information will be taken directly from that source.

4.3.2 Person/Artifact-Relations: Owner

The person(s) currently in charge of working on that artifact. The currently active increment will have all the developers and testers assigned as owners, but a very specific test case run will have only one owner. Also the person responsible for updating the test case document, might be different to the person executing the test for the current test run. Those two items will be different, yet linked artifacts.

Submitter and owner can also be the same person. Submitter and owner will also be the same person, if no specific owner is provided on creation.

It is not only acceptable, but desired that owners change throughout the lifetime of an artifact, depending on its state and the overall lifecycle of the product.

4.3.3 Person/Artifact-Relations: Observers

Anyone in the observers list of an artifact will get the status notifications of that artifact. Automatically submitter, current and previous owners will be part of the observers list. Opting in and out is possible.

4.3.4 Person/Artifact-Relations: TimeTracking

Time is tracked through a list of pairs of begin and end DateTimes, a TimeSpan.

The information can be used twofold. For people it will be valuable to see for what projects they committed their time. For the individual artifacts it will be important to see how much effort is associated to them.

4.4 Detailed Design

Finally the identified components are described in their logical structure. The class diagram helps to show the basic structure of the classes and their connections and interdependencies within them. The object diagram is a kind of a snapshot of the system during runtime and shows when and how objects exist in it.

4.4.1 Nodes Types: Artifact & Person

The GraphDB neo4j only knows nodes. In Global Joint Artifacts nodes can either be people (Person from OpenSocial) or Artifacts (Fig. 26).

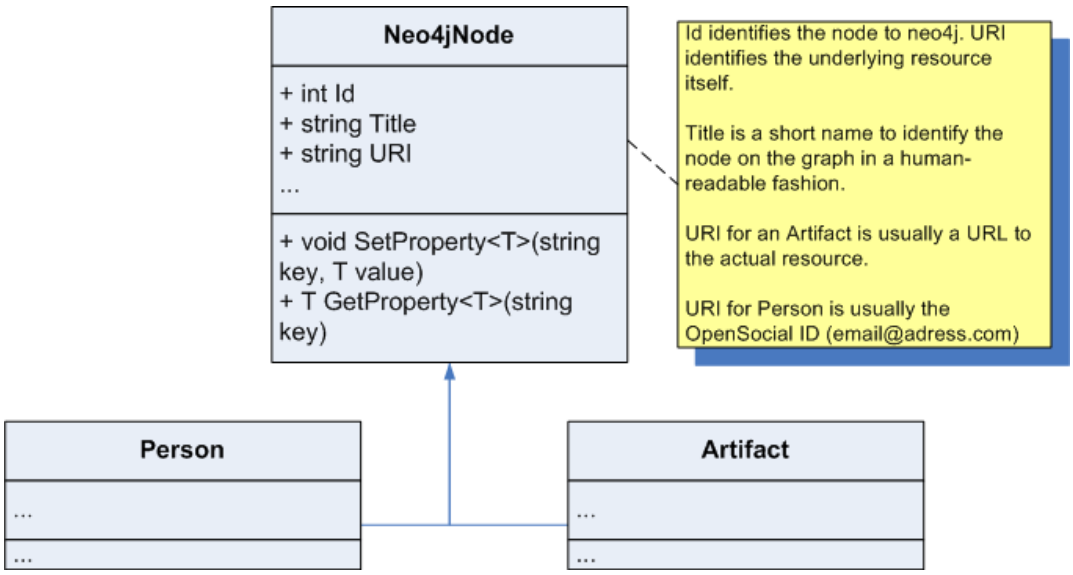

Figure 26: Neo4j object mapping

Every node is identified by a unique “id”. Technically the id is the only mandatory property on a node.

For functional reasons every node must also have a “title”, so it is human-readable. The title should be short and cohesive explaining the node’s purpose. It should be avoided to repeat information from other nodes. E.g. if a node is the representation of the test case design for a certain module in a certain project, it should be simply called “Test Case Design” and related to the module, that itself is linked to the project (instead of “ProjectA’s Setup Test Case Design”).

If the node needs to be described in more detail a “Fullname” or even a “Description” property should be used additionally to the title property.

Many nodes will link to physically existing documents and thus should include the unique resource identifier (“uri”) to it, e.g. \\servername\share$\documents\some.doc. Nodes of type person should also contain a URI property, which contains their OpenSocial Id.

People must have the “isPerson”-attribute set to “true” to be identified by Global Joint Artifacts as “Person”. The icon will change to an actor icon. Optional parameters include “jobTitle” and “currentLocation”.

4.4.2 Artifact/Artifact Relations

Such relations are usually not typed, nor named, but implicitly mean “is part of” or “is like”.

4.4.3 Artifact/Person Relations

People connected to artifacts are usually their submitter/creator, current or previous owners and generally all people interested in that node, observers.

1:1 relations like submitter and current owner could be tracked in the node properties, but are usually also directly linked to the nodes.

The observers are tracked as individual links. So whenever an artifact needs to publish its information it iterates the links of type “observes” (Fig. 27).

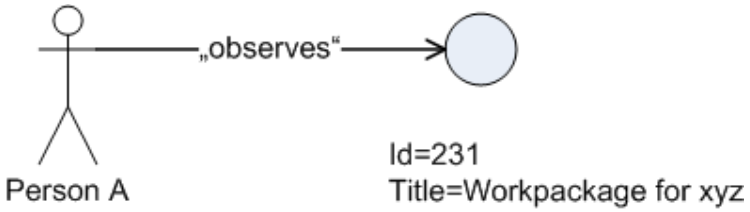

Figure 27: Observer relation

Even more important are means to track time. The information can be used twofold. For people it will be valuable to see for what projects they contributed their time. For the individual artifacts it will be important to see how much effort is associated to them. So the time contributed is not just something that can be tracked on a certain node, but on the relation between the Person and the Artifact (Fig. 28).

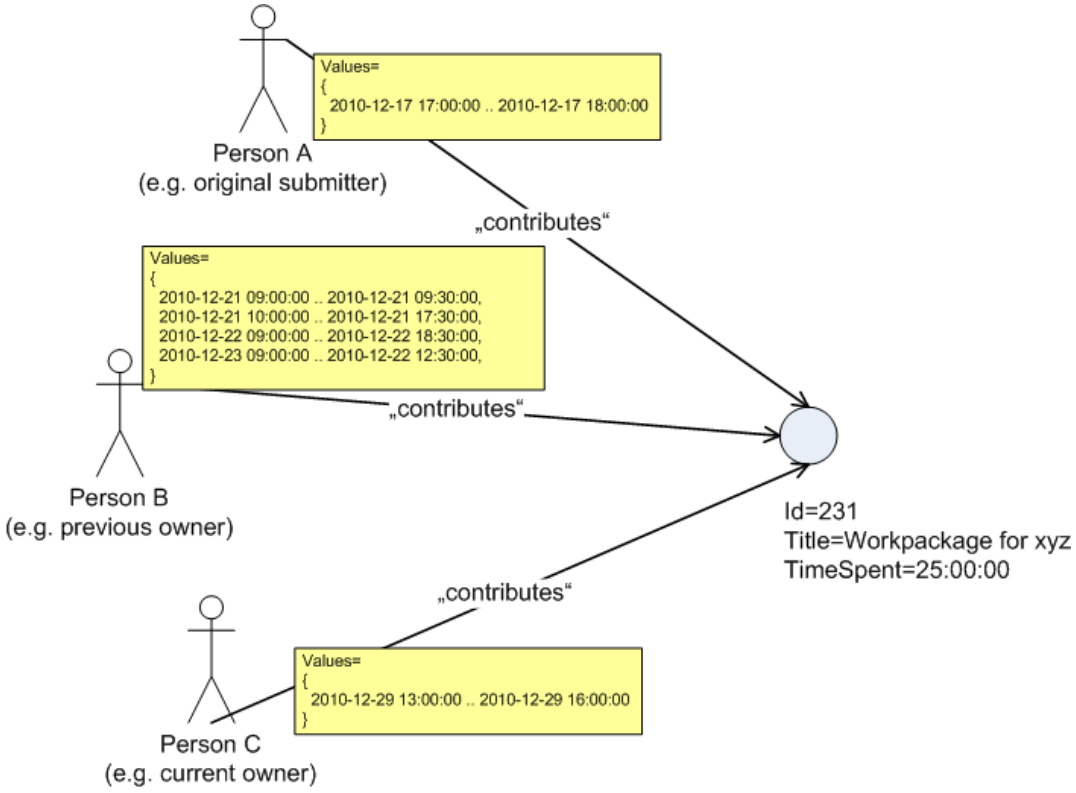

Figure 28: Time tracking on Person/Artifact relation

The “created” and “owns” relations are individual relation types, unrelated to “contributes” or other relations. Technically the “isSubmitter”, “isOwner” and “isObserver” attribute need to be set to “true” to signal that this associated person is submitter, owner and / or observer.

“efforts_min_sum” counts the efforts for that associated user to the artifact.

4.4.4 Dynamically Typed Objects

The “dynamic” keyword specifies variables, which have the class checking deferred until runtime. So they are in a way statically typed to be dynamic.[71] Dynamic is a new feature of .NET 4.0 and should not be mixed up with object or var. Var is also an undefined type like dynamic, but already checked at compile time. Object is the base class of all C# classes and basically means that such an object can be anything, but when trying to access its members it first has to be casted to a concrete class.

4.4.5 Basic or Dynamic Disk Configuration

Basic disks are a traditional concept in Windows. The physical disk is divided into partitions to separate it into manageable sections. A disk can be partitioned into up to four primary and an extended partitions. Within the extended partition logical drives can be created. Partition information is stored in the Windows registry.

A dynamic disk does not have the limitations of a basic disk. It is separated into dynamic volumes, which can offer additional features and improve system performance. A simple volume is similar to a conventional primary partition, with the ability to dynamically increase or decrease its size. A spanned volume can connect several drives to a single, contiguous drive. Striped volumes combine multiple disks into a RAID 0 stripe set, which increases performance. The data is written not just on a single drive, but simultaneously to all drives. As such the drive heads of all the drives just write and then read only a part of the data at the same time. A RAID 1 mirrored volume mirrors a drive to a second drive, thus creating redundancy. Lastly a fault-tolerant striped set on three or more disks is possible with RAID 5. Partition information is stored directly on the disk. [72]

Still the basic disk configuration offers certain benefits. With the basic partitions it is possible to multi boot older Windows or Linux as an equal operating system. It also allows using reliable recovery tools in the event of a problem, which might not work on the new dynamic disk. A basic disk can also be imaged with tools like Norton Ghost. Dynamic disks cannot be set up on removable media like pen sticks or external hard drives. [73]

4.4.6 Graph Visualization

For graph visualization the ICE for Silverlight framework is used (Fig. 29).

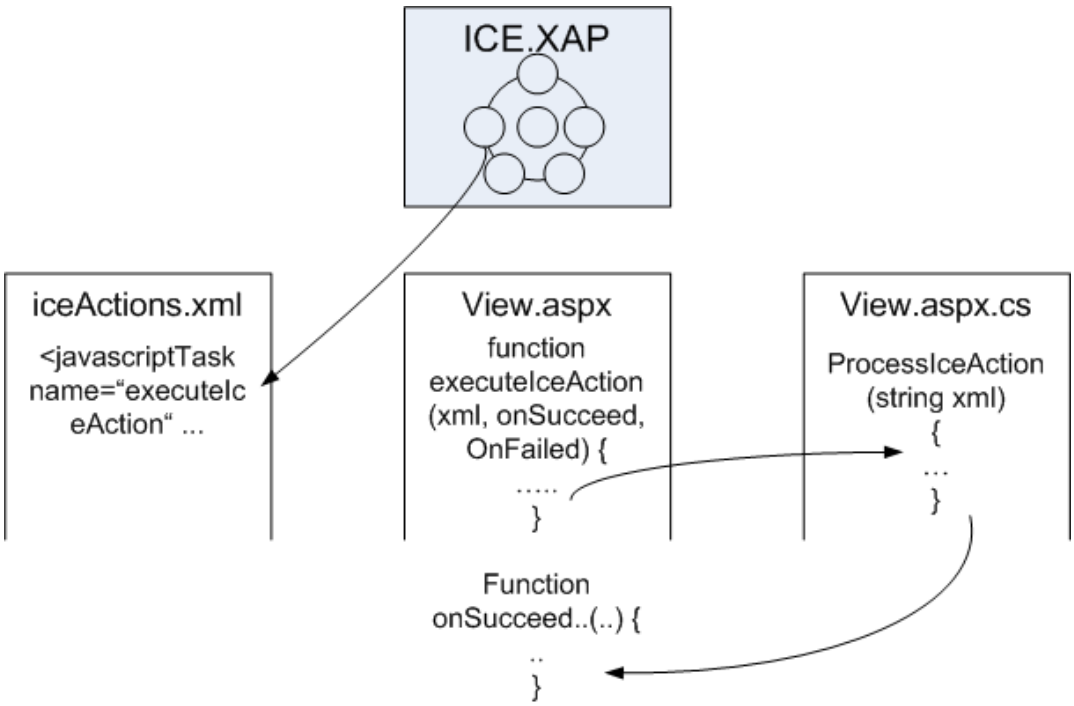

Figure 29: Conceptual view on how ICE' callstack

ICE.XAP is the core Silverlight module of the framework, which is responsible for basic visualization.

Only the expand/collapse action is available out of the box. Further, more sophisticated actions need to be defined in the iceActions.xml.

The View.aspx is a normal ASP.NET page that contains the Ice module. A Javascript function passes the control flow on to the View.aspx.cs, which is the code behind page for View.aspx. View.aspx.cs is a normal C# class. ProcessIceAction processes the XML-formatted content that is provided by ICE.

In the onSucceeded event the result of the code behind can be further processed.

Please find more details on that framework in the according chapter 4.5.9 ”Graph Visualization”.

4.5 Evaluation of existing methodologies and models, technologies and tools

In software engineering there is nothing worse than re-inventing the wheel. Time and resources are limited and scarce. With this in mind the project tries to re-use as much functionality as possible.

This chapter summarizes the evaluation of the state of the art.

4.5.1 OpenSocial

OpenSocial is Google's attempt to create a common interface for accessing social networks programmatically. Once implemented by a network the OpenSocial API can be used instead of the network's proprietary APIs and services to access its functionality. The general idea is to develop once, and then deploy wherever OpenSocial is available. [75]

As compared to using each individual network's APIs the developer loses functionality when using a common API. Still, the fact of writing code once and then re-using it on all networks that implement OpenSocial is compelling – especially when development resources are scarce. Dozens of social networks implement OpenSocial already[74]. Yet Facebook, the social network with the biggest buzz, does not implement it. This is of course a drawback for reaching many end users. At least through the OpenSocket project OpenSocial gadgets can be run on Facebook.

But for the enterprise usage, as a tool for distributed testing, this is seen as acceptable; the two professional networks LinkedIn and Xing implement OpenSocial.

The approach for Global Joint Artifacts is to stick to OpenSocial as long as possible and only when the relatively rich API does not provide necessary functionality implement it with a network's custom API.

The big advantage of OpenSocial is that it strives to be an open standard, so developers are not locked into proprietary software.

4.5.1.1 Implementation Patterns

The most lightweight form to integrate into a social network through the OpenSocial API is a social mashup. Such gadgets do not need a server, but thus are limited in terms of functionality due to a lack of data storage.

A social application runs as an application on the social network, but additionally has a data backend and therefore can expose more advanced functionality than a gadget. As it requires an external server it can run into all issues of a normal web application, e.g. scaling problems.

A full social website consumes social data through OpenSocial REST/RPC, but runs outside of the other social networks as a full web site or web service. Just because a social network claims to support “OpenSocial” not automatically means the REST API is also implemented, so it always needs deep investigation and usually a proof of concept that a website can be connected to a container through it.

Global Joint Artifacts will be such a social application, as it offers the most flexibility and offers the users to interconnect with users not only on one social network, but all OpenSocial-enabled networks (Fig. 30).

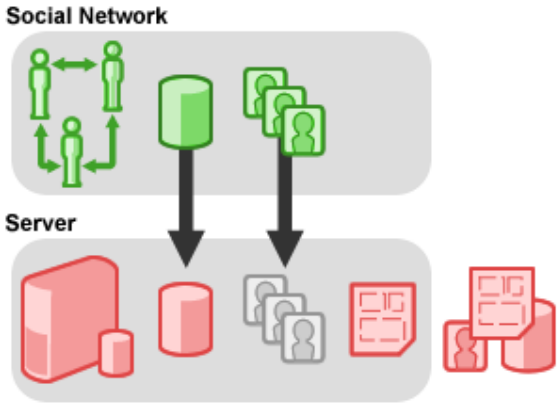

Figure 30: OpenSocial Social Website Architecture [75]

Drawback of this architecture is the high degree of complexity that comes with the flexibility.

Social applications are then fully independent of a specific container. As such any website can become a social application for utilizing social features like sharing plans and further information.

4.5.1.2 Data Requests [76]

Requesting data from a social network or the social application is an essential part of the OpenSocial API. Data requests access people information, app data and activity data. These individual requests share a similar syntax.

A data request object can aggregate one or more data requests. The individual requests are attached to the object and labelled with a key. Instead of handling several individual asynchronous calls the data requests are all sent at the same time and can be processed together in a callback function.

They can fetch information on a single person, or a group of people. These people can either be the enumeration of friends of a person, or a specified array of people. The API also differentiates between two roles a person can take – owner or viewer. A person using an application on the own profile will be both owner and viewer. Someone interacting with an application installed on another person's profile will be viewer only. When specifying a person possible identifiers are opensocial.IdSpec.PersonId.OWNER, pensocial.IdSpec.PersonId.VIEWER, or an OpenSocial ID. The opensocial.Person class stores the person's personal information and provides methods to access it.

The application can store user-specific data, retrieve it at a later point in time and remove it. Social networks that implemented OpenSocial provide a persistence layer, which can then be accessed through the API. The data is user-specific, yet not necessarily private. Depending on the implementation of the network OpenSocial might be limited to write data only for the current user of the application, but could fetch app data for any user who has the application installed.

Finally a collection of activities tailored for the current profile can be fetched. This activity stream again needs the OpenSocial ID of the appropriate person. Activities are mainly interests and should not be mistaken with what is called activities in software engineering.

The opensocial.message class allows sending different kinds of messages to the underlying container.

4.5.1.3 Versions [74]

The latest version 1.0 of OpenSocial is not yet implemented in any of the important containers. For Global Joint Artifacts the version 0.8 is probably the best base line, as it is implemented by many social networks, including LinkedIn. Version 0.9 is already available for the Xing and RenRen platforms.

4.5.1.4 Improvements from version 0.8 to 0.9

The OpenSocial Markup language (OSML) allows specifying common UI elements in the containers. An example is a GUI to select contacts, which basically any container provides.

Data pipelining is a declarative syntax to access user data, without the need to invoke data requests.

To secure the communication a related technology called 3-legged OAuth will be used together with the OpenSocial REST API. OAuth is discussed in greater detail in the next chapter 4.5.2 “OAuth”. OAuth is a general purpose technology for authorization on the internet and hence could be used by itself too. But the OpenSocial libraries provide all the necessary OAuth functionality and therefore make it easier to just use what is actually needed for OpenSocial.[77]

4.5.1.5 Extensibility

OpenSocial is meant to be extensible. All containers support a basic set of features, but all can have additional functionality on top. As an example all containers provide a DisplayName and Thumbnail URL for a Person, but each container will have additional attributes to a person’s profile.

4.5.2 OAuth

OAuth is an “open protocol to allow secure API authorization in a simple and standard method from desktop and web applications.”[78] OAuth is used by the Google Data APIs[79], including OpenSocial.

The following chapter only discusses OAuth’s use for OpenSocial.[80]

“OAuth includes a Consumer Key and matching Consumer Secret that together authenticate the Consumer (as opposed to the User) to the Service Provider. Consumer-specific identification allows the Service Provider to vary access levels to Consumers (such as un-throttled access to resources).

Service Providers SHOULD NOT rely on the Consumer Secret as a method to verify the Consumer identity, unless the Consumer Secret is known to be inaccessible to anyone other than the Consumer and the Service Provider. The Consumer Secret MAY be an empty string (for example when no Consumer verification is needed, or when verification is achieved through other means such as RSA).”[81]

In its most simple form OAuth can be used to make a signed request. Signed requests allow recipients to verify that the request was not tampered while in transit. In the concrete use case it allows third party developers to verify “that social data passed to their servers was sent from a specific container.”[80]

The request flow is relatively simple when a gadget wants to transmit information to and retrieve information from an application server. The initial request from the gadget, which is running in a container (a social network like Orkut) is sent to that container.

The container adds the requested information and then uses OAuth to sign the request. The application server receives that signed request and thus can be sure the data was not tampered by the gadget. After verification the application server responds with the requested data. The gadget then can display that retrieved information.

Two legged OAuth allows the social network and an application server to securely exchange information. Its purpose is to support a gadget's background processing.

For Global Joint Artifacts 3-legged OAuth will be used. In this scenario the social network is accessed by a user over the application server. So here usually no gadget runs on the social network, but the user is on a third party website or uses a mobile or desktop application. The advantage of OAuth is that the user does not directly provide his social network credentials to the (potentially) unsafe third party application, but uses OAuth for authorization. This approach is useful for "enabling" an existing site with social features.

Being able to connect to multiple social networks is an extra advantage, which also means that the website can itself connect several networks.

The request workflow is much more complex in this use case (Fig. 31).

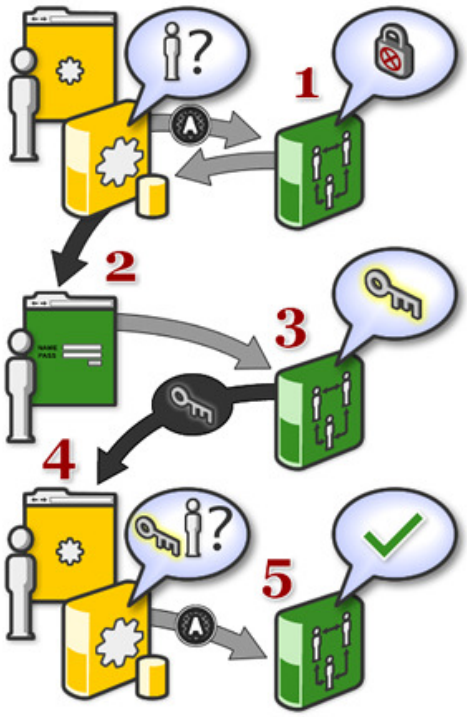

Figure 31: 3-legged OAuth Request Flow[80]

As discussed a user is on a third party application or website. The user then invokes an action that requires information from a social network. It is important to note that the user is *not* within the context of a social network, which implies that this application does not have the user's credentials for the container. Thus the application cannot access the user's data.

Because of this the request for information is redirected to a special login page of the container. There two keys need to be retrieved: OAuth Consumer Key and OAuth Secret Key to begin the request. "Obtaining these keys requires consulting the documentation for whichever site you require access to as they all handle distribution of this data in different manners."[77]

With the container's provider information, the consumer and secret key a request token can be retrieved. The request token allows to redirect the user to the authorization URL, which is similar to a login page.

After successful login the user is notified and must confirm that heorshe will share data with that website or application. Once the user accepts to share the information the application is redirected to a special URL on the application server. The application server then can use an OAuth token to access the user's data on the container.

4.5.3 Orkut

Orkut is the name of Google's own social network. As such it is featured in several of the OpenSocial examples as an OpenSocial container. All OpenSocial containers at least support the JavaScript API for use on websites the user views in the browser.[82] Orkut additionally also supports some features of the OpenSocial REST API[83], which allows direct server to server communication.

According to the OpenSocial containers list 17% of its users are from India and 50% from Brazil[74], which makes it suitable for keeping contact with outsourced software development and testing personnel.

And so out of those reasons Orkut's sandbox is the test environment for Global Joint Artifacts.

4.5.4 LinkedIn

As a commercial tool Global Joint Artifacts would also have to support LinkedIn, which aims to be *the* professional network among the social networks. LinkedIn claims to have 90 million users, which use the network to "exchange information, ideas and opportuni-

ties".[84] Among its users are "executives from all 500 companies in the Fortune 500."[85]

To limit the development efforts for the prototype testing the Orkut implementation will have to suffice.

4.5.5 Remote Desktop Services

Windows 2008 R2 lets remote users and administrators access the operating system as if they were physically at the workstation. For usability reasons it is beneficial if administrators can access the system without the need to be physically present. This feature is also useful for demonstrating the Global Joint Artifacts system in an early, still locally running, state.

Remote Desktop Services were before known as Terminal Server and use the same Remote Desktop Protocol (RDP). Thus any Remote Desktop Connection (RDC) client is able to access a RDP session. The technology allows both to remotely access the whole desktop or single published applications similar to Citrix XenApp that seamlessly integrate into the host operating system.

In Windows 2008 R2 "Remote Desktop Services" is a so called server role. Roles can be selected easily either on install or at a later point in time through the "Add Role Wizard".

Associated with the Remote Desktop Services role are several role services. Not all of them are needed for a simple remote connection. A license server (RD Licensing, formerly known as TS License) for example is only needed if more than two concurrent connections to the server must be supported.[86]

For simple administration enabling the Remote Desktop connections from the System menu is sufficient. Windows 2008 already supports Network Level Authentification (NLA). Before, users were allowed to already access the login screen of the remote system, which is a security weakness, e.g. for DoS-attacks. With NLA the check for the credentials does not need the entire remote desktop to load. NLA was originally a Windows 7 / 2008 exclusive feature, but can be enabled via a registry key in Windows XP SP3.[87]

If the remote access is to be made available outside the same network or even over the Internet securing the connection will become necessary.

The RD Gateway (RDG) is a very valuable addition to the Remote Desktop Services as it provides out of the box security, without the need to setup a real VPN. Using a HTTPS connection the port 443 is used instead of the conventional port 3389 for RDP. For HTTPS to work an SSL certificate is required. In a production environment it is either maintained by an enterprise certification authority (CA) or a CA infrastructure within the organisation.

Working with self-generated certificates is only recommended for technical evaluation or testing purposes. [88] A Self-signed certificate for SSL encryption must be manually installed on all clients that will communicate with that server. The RDG Manager allows creating a new self-signed certificate and authorization policies. Its main purpose is monitoring of the current connections to the system, with details on the currently and previously logged on users.

The self-signed certificate is initially only put into the Personal certificates on the server and needs to be manually copied (not moved!) to the Trusted Root Certification Authorities to be useable for RDG. Then this certificate needs to be exported and copied manually to the client computers' Trusted Root Certification Authorities. The certificate subject name needs to match the computer name to work, so a certificate for computer name "xyz" needs to be "Issued to" and "Issued by" "xyz". So the RD client also needs to use the proper computer name.

So that users are able to connect to the RD Gateway a Remote Desktop connection authorization policy (RD CAP) that specifies the users who can connect, as well as a Remote Desktop resource authorization policy (RD RAP), which maps users to computers, are necessary.

By default the Administrators group is selected. If it is not desired that generally all administrators have access

4.5.6 Neo4j

"Neo4j [is] a high-performance graph engine with all the features of a mature and robust database. The programmer works with an object-oriented, flexible network structure rather than with strict and static tables — yet enjoys all the benefits of a fully transactional, enterprise-strength database."[89]

Running the application on the Windows Server 2008 R2 it would suggest itself using Microsoft's SQL Server 2008. The SQL Server is an industrial grade relational database management system (RDBMS). SQL Server would likely be sufficient for the prototype's expected data load. Yet a fully operational system would have thousands or even

millions of users, who probably generate millions or billions of interconnected data. With this in mind it will make sense to already base the prototype on a more future proof technology.

Several projects are summarized under the umbrella term „NoSQL“ (Not Only SQL). They have in common that they try to compensate the limitations into which the development of RDBMS has come. Problems addressed by NoSQL are high-performance querying of “deep” SQL queries that span many table joins, poor alignment to modern software development approaches like object-orientation and dynamic languages, general scalability, schema evolution over time, modelling of tree structures, semi structured data, hierarchies and networks and more.[90]

In RDBMS there are two possible ways to store such data, and both have their clear technical limitations. Data could be normalized and would then be stored in many different tables. For queries the tables need to be joined, but the join operation is a very expensive operation in regards of performance. Alternatively so called big tables could be used to store semi-structured data, which store all information as the name implies in a single table. Big tables these have many columns, which only have sparsely populated rows, which can lead to poor performance.

So the bottom line is that graph information can be normalized to RDBMS, but results in poor query performance due to the necessary joins.

Also see the chapter 2.5 "Non-relational data storages” for further details on NoSQL.

For the problem of storing interconnected data like the Global Joint Artifacts data a graph database seems to be the most promising solution.

Below diagram shows the basic components of the property graph model (Fig. 32).

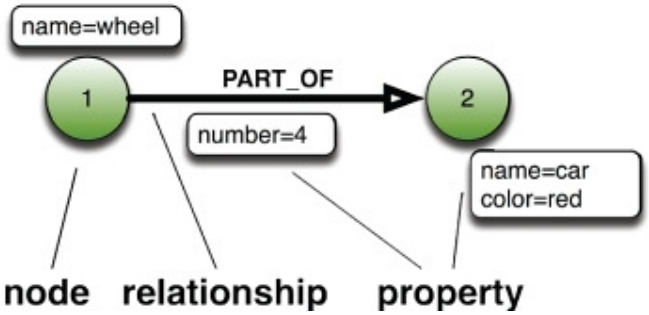

Figure 32: A property graph's basic elements[90]

The node (also known as vertex) is the actual topic. The edges between the nodes model the relationships. The edges are directed and labelled giving information on the type of

the relation. Usually such graphs are multigraphs, which allow multiple edges even between the same nodes in the same direction. Each node and edge may have a list of arbitrary attributes that give further information on that component.

The currently most active open source graph database project seems to be neo4j. “It is an embedded, disk-based, fully transactional Java persistence engine that stores data structured in graphs rather than in tables. A graph (mathematical lingo for a network) is a flexible data structure that allows a more agile and rapid style of development. “Neo4j can handle graphs of several billion nodes/relationships/properties on a single machine and can be sharded to scale out across multiple machines.”[89]

Neo4j can be either used as an embedded library or standalone server. As the name implies this is a project written in Java, and as such the JAR cannot be easily embedded into a .NET application. Apart from some not very active projects that attempt to port neo4j to the .NET platform a blog entry[91] describes how to compile the java source code directly to .NET 3.5. This approach would have the advantage to be able to directly incorporate the resulting assembly into the Global Joint Artifacts development project. But several replies suggest that there are still open issues in this conversion process. Thus finding another approach will be necessary.

An alternative would be to use the active, yet closed source, graph database Sones [92], which has the advantage of native .NET support.

But there is an alternative to be able to use neo4j. The neo4j standalone server exposes a REST API, which allows using neo4j’s functionality from basically any programming environment that supports a REST interface. The web service also has the advantage of loose coupling of the internal components.

For the web service two .NET wrappers exist, but both are not complete.[93] [94] For the prototype Neo4j REST .NET from BitBucket was selected as it has most of the traversal functionality implemented.

Every node has a unique identifier just like in a database. In the REST wrapper the type of the Id is long. The nodes can also be described as a so called “self”-string. It includes the fully qualified path to the node. As an example, assuming n is an arbitrary node: If n.Id=1, then n.self=”http://localhost:7474/node/1/”.

Neo4j maintains data integrity just like RDBMS through transactions that support the ACID properties. So all write operations on the graph need to be performed in a transaction. Transactions should always be wrapped in try/catch/finally-blocks, because failing to finishing it will not release the locks acquired. All intermediate steps of a transaction

are kept in memory, so very large modifications will have to be split to not run out of memory.

A write lock will be acquired for adding, changing and removing a property on a node or relationship for that item. Creating or deleting a node will acquire a write lock on that particular node, while creating or deleting a relationship not only affects the relationship, but also both its nodes.

Deleting a node also deletes its properties. The operation is only possible if all its relationships have been already deleted. All the node's properties will be removed with it.[95]

Physically the files that make up the graph database are stored in the <neo4j home>\data\graph.db\ directory. In <neo4j home>\data\log\neo4j.log all log messages from the server are stored. The log file <neo4j home>\data\log\wrapper.log contains all information on the neo4j server Windows service.

The neo4j server comes with a web administration interface. By default it is running on port 7474 and thus can be accessed through http://localhost:7474. It provides a dashboard to monitor the server and allows viewing and modifying data either through the user interface or the Gremlin programming language.

It is possible to run several database servers from the same physical machine. For n database servers n services need to run. Their ports also need to differ to avoid clashes while accessing those servers.[96]

The next chapter discusses Gremlin, a programming and query language for working with graphs. Gremlin uses XPath expressions to give access to the database. As such it can be compared to SQL for RDBMS in its scope and functionality.

4.5.7 Gremlin

Gremlin allows working programmatically with a graph. It is not specific to any product, but a generic tool for viewing and modifying graphs. Currently it is for example used in neo4j, OrientDB and Rexster.[97]

Property graphs can be created and traversed. Depth-first or breadth-first traversals are supported. Gremlin offers a rich set of functions to handle graphs and allows specifying user defined functions.

The following example shows how reading out the status of the isObserver flag on an edge:

```
g:print ( g:id-e(26) / @isObserver )
```

If the edge of Id 26 has the isObserver flag, and it is set to true the output will be

```
==> true
```

Gremlin also allows to import and export graphs in the GraphML format, which is discussed in the next section.

4.5.8 Graph Exchange Formats

Several formats exist to exchange graph data between applications. A selection of those most relevant to the thesis are discussed below.

4.5.8.1 GraphML

The Graph Markup Language is an XML dialect for graph data exchange between different applications. The Gremlin language allows saving and loading GraphML specified graphs, which also means that neo4j can work with that format out of the box.

4.5.8.2 DOT

DOT describes graph data similar to Java classes. It is GraphViz' proprietary format for graph import and export. GraphViz is a renowned visualization toolkit, and as such is also supported by neo4j.

4.5.8.3 GXL

The Graph eXchange Language is another XML dialect for specifying graphs.

The next chapter revolves around how the graphs can be visualized. GraphML allows using visualization components that were not specifically written for neo4j.

4.5.9 Graph Visualization

Only as people can see and interact with the graph the Global Joint Artifacts not only store the information, but really help gaining knowledge (also see chapter 2.1 "Knowledge Management"). Best would be that there is a tight integration between the used graph database and a visualization component to prevent the programming overhead of an intermediate layer.

Neo4j does **not** have a readymade visualization component.[98]

Gremlin allows exporting graphs in GraphML and there is also a neo4j component for GraphViz. The latter is still in the so called laboratory area of neo4j components.[99]

GraphViz' main purpose is also displaying static graphs. Thus it is used for rendering graph data in static images.

Gephi is an extremely sophisticated graph visualization and graph analysis framework. Disadvantage is its high overhead of functionality that is not needed for Global Joint Artifacts. Further it requires a Open GL enabled graphics card.

Prefuse/Flare is a Flash component, which offers a rich set of visualizing options and graph layout algorithms. The decision against this framework was mainly because it is not a .NET component, which would have meant a lot of additional development efforts.

With the Graph# and QuickGraph libraries a large part of the prototype was already implemented. Sadly Graph# works only on the Windows proprietary Windows Presentation Framework (WPF). And even the latest WPF for web only works on Windows based clients.

Ultimately the solution that was taken is called Information Connections Engine (ICE) for Silverlight. Although Silverlight is a Microsoft technology it is available for Linux and Apple OS based browsers. Major drawback is that it only features one algorithm out of the box. It is a weight-based algorithm. Implementing the Sigayama algorithm could be a future task.

All in all the effort to implement the visualization component was surprisingly high, as no out of the box 1:1 visualization exists for neo4j.

4.5.10 Dublin Core

Metadata describes the underlying data. For example if a picture is the data, the properties of the picture, like format, height, width or a title and keywords are its metadata.

"The association of standardized descriptive metadata with networked objects has the potential for substantially improving resource discovery capabilities by enabling field-based (e.g., author, title) searches, permitting indexing of non-textual objects, and allowing access to the surrogate content that is distinct from access to the content of the resource itself."[100]

Only through relations the Global Joint Artifacts make sense, so there needs to be metadata to describe them.

The so called Dublin Core specifies a basic set of metadata elements for any kind of objects. Elements are for example “Title”, “Description”, “Creator” and “Contributor”.[101] Main application is giving documents on the Internet meaningful attributes, so they can be catalogued and can be retrieved. It is also part of the Open-Document-Format.

Dublin Core can be expressed in many forms such as Text, HTML or XML. For the latter XML schema files are provided, which allow to create XML files that can be validated. Of course it is then also possible to extend the XML files with additional application specific metadata.[102]

4.5.11 AGPL v3

Many of the evaluated tools have with recent versions moved from a GPLicense to the stricter AGPL v3. That license is basically still the same as GPL, but with the addition of a new paragraph to close the so called SaaS loophole.[103] Normally developers are obliged to redistribute changes to the original code if the software is distributed, e.g. in its binary form. Software as a Service providers do not provide the software to their customers, but have it running in their data centers. The customers only access the service, usually without anything installed on their local machines. The paragraph adds now the liability to also redistribute changes to the code, it the software is part of a service that is served through the web.

The AGPL v3 does not mean though that the whole solution needs to be exposed – it’s only about changes to the product licensed with AGPL v3. So if MySQL would use AGPL v3 it would only mean that changes to MySQL need to be made available to the public. If MySQL is just used out-of-the-box to store data, there is no obligation to expose other parts of the software.

4.5.12 DotNetNuke CMS

Any modern website consists of several document areas, like a news section, an about section and so on. More documents will be added over time. A Content Management System (CMS) can greatly decrease the time needed to change such basic functionality and content on the site. Global Joint Artifacts will use the DotNetNuke Community Edition version 5.6, which was released on November 17, 2010.[104]

Installation requires the preparation of the file system, including setting the appropriate permissions, web server (IIS), and database server. DotNetNuke uses SQL Server as its data backend. After successful setup the application can be tested over http://localhost/dotnetnuke/.

DotNetNuke was evaluated, but the decision was not to use it and just rely on the features given by ASP.NET itself. A commercial grade product could make good use of a proper CMS though.

4.5.13 Calendar

An important attribute of software engineering artifacts like features to be implemented and defects to be fixed are the dates they need to be finished and the predicted and actual efforts. It will be important to include a calendar user interface, which allows both viewing and entering the end dates and efforts.

4.5.13.1 CalDAV

As working with calendar data is an old necessity in programming many standards already exist. CalDAV are the calendaring extensions to WebDAV, the Web-based Distributed Authoring and Versioning.

The WebDAV standard defines a data model of methods, headers, and content-types for resource properties, locking for collision avoidance, request and response handling and methods for distributed authoring through resource collections.[105] WebDAV's capabilities can and often are implemented through alternative technologies. WebDAV's file sharing capabilities could also be implemented by the SMB protocol. An alternative for handling web resources is the Atom standard.

CalDAV defines a calendaring data model, object resources and collections and methods for creating and accessing or restricting access to them. It allows managing and sharing calendaring and scheduling information, based on the iCal(endar) format.[106]

The iCalendar data format supports environment independent exchange of calendar and scheduling information. As such it is widely used by many applications. It includes calendar items such as events, to-dos, journal entries, free/busy and recurrence information, participation status, relationship types and participation roles.[107]

4.5.13.2 Google Calendar

A feature-rich, yet royalty-free and simple to integrate collaborative web calendar control is the Google Calendar.

The API to access the calendar data is twofold. The Calendar Data API allows typical interaction with the calendar similar to what a user can do in the web UI. With CalDAV

a second, industry standard interface exists to interact with the calendar data.[108] Thus the Google Calendar can also interconnect with many client applications.

Extensibility of the calendar can be achieved through so called gadgets. The publishing tools contrarily allow to extend own web sites with the calendar in a low effort way. Embeddable Calendars are read-only copies of the Google Calendar. They are easy to add, yet very limited in their functionality. Event publisher on the other hand allows with low effort to add events or entire calendars to a user's Calendar account.

The Calendar data API allows to create, query, modify and delete "events" (appointments, to-dos, etc.) in the calendar.

The Calendar API uses the same Data Protocol as the other Google Data APIs. The Google Data Protocol, short GData, allows accessing and modifying data stored by the diverse range of Google applications. The protocol can be either accessed directly or through client libraries. If accessed directly any programming language that can send HTTP requests to GET or POST can be used. Several methods can be used like the JSON or AtomPub. The client libraries wrap those calls, abstracting the technical implementation details and providing tools to ease the integration. The libraries and their documentation are not only available for Java or Python, but also .NET, so suitable for this project.

Extended properties allow adding arbitrary name-value pairs to Calendar events. It is intended for short information of up to 1024 characters. These properties are exclusive to the API and not visible in the normal user interface.[109]

For the prototype only minimal calendar functionality is needed to support DateTime entry.

4.6 Summary

The design influenced the tools, methods and technologies that were evaluated, and the evaluation then influenced the further design. An iterative development process was used that allowed to go back and forth through the development phases as needed and suitable.

Tools and libraries had to be evaluated to a certain extent to be able to decide if they should be included in the actual prototype. The following matrix gives an insight if something was evaluated, used in the design and planned for later steps in the development after the prototype phase or if it was actually used in the prototype's implementation.

Component / Method / Tool / Technology	Evaluated	In the design	In the prototype
OpenSocial	X	X	X *
OAuth	X	X	X *
Orkut	X	X	-
LinkedIn	X	X	-
RDS	X	-	X
Neo4j	X	X	X
Gremlin	X	X	X
GraphML	X	-	-
DOT	X	-	-
GXL	X	-	-
GraphViz	X	-	-
Gephi	X	-	-
Prefuse/Flare	X	-	-
Graph#	X	-	-
QuickGraph	X	-	-
Information Connections Engine (ICE)	X	X	X
Windows Presentation Foundation (WPF)	-	-	-
Silverlight	-	X	X
Sigayama algorithm	-	X	-
Dublin Core	X	X	X
AGPL v3	X	X	X
DotNetNuke CMS	X	-	-
CalDAV	X	-	-
Google Calendar	X	-	-

Table 7: Component evaluation/design/implementation-Matrix

OpenSocial and OAuth are marked (*) as they are used in the prototype, but as the prototype is still only a locally running application and not published to the web the OAuth keys are not available to fully utilize OpenSocial.

The Windows Presentation Foundation (WPF), Silverlight and the Sigayama algorithm are mentioned in the list, although they were not described in this document and thus are not listed as evaluated. Still, they are related topics for the prototype and important enough to be at least mentioned here.

As a practical work the final result is the description of a concrete and running application – Global Joint Artifacts. In short Global Joint Artifacts interconnects people and their work items and work in progress (artifacts). Find all details in the chapter 3 “Global Joint Artifacts RequirementsGlobal Joint Artifacts ”.

The prototype is specified according to traditional object oriented analysis. For specification and also as a basis for discussion with peers the modeling language UML 2 is used. The overall architecture takes elements from the service oriented architecture with very loose coupling of the components to increase maintainability and extensibility.

For the practical part the .NET Framework in the current fourth version is used. C# is used as the main programming language. Windows Server 2008 R2 Enterprise is the operating system for Global Joint Artifacts. Concrete result is an executable prototype that shows the basic functions of the solution. The prototype is a full social website empowering the features of a web site for project management with social network features and capabilities. The prototype follows the trend of a hosting the service, or nowadays called software as a service (SaaS), instead of an on-premises-solution.

Further components are taken from freely available services and from the open source community. The evaluation of existing tools and applications always keeps in mind how Global Joint Artifacts can add value to existing tools of project management. For details on the evaluated tools see chapter 4.5 “Evaluation of existing methodologies and models, technologies and tools”. It is important to evaluate the licenses as well, to be able to stay compliant with the rules and regulations set by the tools.

5 Results and analysis of the prototype

This chapter discusses the results from the prototype. It is a critical analysis and summary of the previous chapters. The advantages, but also limitations of the prototypical solution and its methods are explained.

The hypotheses are repeated here as a cross-reference to chapter 1.3 “Research Goal”.

5.1 Case Study

The hypotheses are verified with live data from the company at which the author is employed. The company is an internationally highly distributed software developer with 5.500 employees spread across several sites in the USA, Europe and Asia. The products include end-user box products as well as B2B applications. Usually the products address a larger market, and not just a single customer. All data is made anonymous and not given in detail in this thesis for obvious corporate confidentiality reasons. Entries are encoded with their identification numbers, but concrete titles or detailed descriptions cannot be found in the thesis nor in the prototype. All the people's and project names were changed. Throughout the following chapter the company will be even called "the software company".

For managing work packages and to track the estimated and actually spent efforts a range of diverse systems are utilized.

Rational ClearQuest is used as work package, change request and defect tracking system. It contains features to estimate and also track spent efforts separated by the development steps in the areas specification, implementation, test, and documentation.

Update Seven is used as customer relationship management (CRM) system and was used for customer call and service request tracking. It contains tracking for the support and pre-sales department. Service requests and knowledge base entries are now handled by a custom Oracle Forms application. The transition phase lasted nearly a whole year were both systems were used. Already tracked entries were left in the old system, new calls were tracked in the new system.

Time is also tracked in another Oracle forms based application on a project based level.

All systems have their own ways of authorization, even different user names and passwords are required. Projects and generally categorization need to be re-created in each of the systems. Start up times of both ClearQuest and Oracle are relatively slow and the

user is not immediately at the overview of his entries. In the case of Oracle a user needs to move through seven windows and all in all depending on the network connection speed it can take up to four minutes from opening a browser window until the view of owned cases is ready.

For reporting to the social insurance and other governmental agencies an Excel sheet is used.

Office (Word and Excel) documents are usually stored on ClearCase, a versioning system similar to CVS or Subversion, that allows controlled modification by teams, even if they are distributed.

For storing and communicating information across the company and the different parts of the company a Wiki-system is set up as part of the Intranet.

Goal of this chapter is to further strengthen the assumptions of the thesis. The thesis is mainly the reasoning and documentation of the practical work and not mere theoretical research. A full comprehensive study could be done in future work.

5.2 Hypothesis 1

Connecting people and artifacts increases the benefit of the artifacts.

5.2.1 Detailed Problem Description

The connection of artifacts with people helps remote teams to clearly understand where requirements, findings and assumptions derive from. This will let them work self-sufficient from the main team allowing them to better estimate and prioritize.

In our software testing department requirements are referenced in test case documents through IDs to requirement documents. This is a very lose coupling and only with a system like Global Joint Artifacts it’s really possible to grasp the whole story of a requirement: Who requested what feature, how is it related with other similar requirements, what similar test cases already exist. All this can usually only be found out by extensive conversation within the organization, which is problematic, if that company or organization is only loosely coupled and even globally distributed. For details on the challenges in distributed teams see chapter 2.4 "Global Software Development".

Global Joint Artifacts allows globally distributed team members to drill further down into the dependency graph of a feature and thus supports their decision making.

So the hypothesis tries to follow up on how much effort is saved and what additional value can be created by using Global Joint Artifacts.

5.2.2 Verification

This hypothesis was verified by comparing the prototype with the current situation with the traditional systems Oracle service request, Rational ClearQuest and the way test cases and product requirements are stored. The following chapter first elaborates on the background of this hypothesis and then presents concrete test cases and their results.

5.2.2.1 Finding people with the right knowledge

The traditional systems are all backed by a relational database model. As such their structure of interconnections is flat – every entry is on the same level. As an example the view on a query in ClearQuest is shown in Fig. 33.

ClearQuest Query Results (markusg,BugTracking@PSP)

id	Submit_Date	Product	Owner.fullname	State	SEff	IEff	DEff
PSP00055332	18/02/2011 16:57:59	SMSB Aurea		Unconfirmed	0	0	0
PSP00055317	18/02/2011 10:35:29	SMSB Aurea		Unconfirmed	0	0	0
PSP00055278	16/02/2011 14:28:19	IA for SM 6.1 (Dagobah)		Unconfirmed	0	0	0
PSP00052550	24/06/2010 15:17:21	SM Athena		Unconfirmed	0	0	0
PSP00052549	24/06/2010 14:42:22	SM Athena		Unconfirmed	0	0	0
PSP00052498	21/06/2010 14:57:40	SMSB Aurea		Will_Not_Fix	0	0	0
PSP00052471	17/06/2010 15:57:01	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00052451	16/06/2010 12:51:19	SMSB Aurea		Will_Not_Fix	0	0	0
PSP00052392	10/06/2010 09:55:49	SMSB Aurea		Test_Passed	0	0	0
PSP00052300	26/05/2010 11:13:50	SM Athena		Will_Not_Fix	0	0	0
PSP00052045	27/04/2010 16:59:30	IA for SM 6.1 (Dagobah)		Confirmed	0	0	0
PSP00051894	02/04/2010 11:46:00	SMSB Aurea		Will_Not_Fix	0	2	0
PSP00051666	08/03/2010 09:59:19	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00051617	04/03/2010 11:03:10	SMSB Aurea		Fixed	0	0	0
PSP00051551	26/02/2010 10:02:18	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00051481	23/02/2010 09:28:43	SM Athena		Will_Not_Fix	0	0	0
PSP00051332	09/02/2010 09:25:55	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00051126	21/01/2010 13:47:38	SMSB Aurea		Test_Passed	0	0	0
PSP00050964	12/01/2010 13:41:11	SMSB Aurea		Will_Not_Fix	0	0	0
PSP00050681	09/12/2009 10:04:51	SM Athena		Unconfirmed	0	0	0
PSP00050602	01/12/2009 16:25:26	SMSB Aurea		Test_Passed	0	0	0
PSP00050244	03/11/2009 11:42:19	IT Radiologia V5.10		Will_Not_Fix	0	0	0
PSP00050062	22/10/2009 17:04:14	SM Athena		Will_Not_Fix	0	0	0
PSP00049881	20/10/2009 13:58:33	SpeechMagic V6.1		Will_Not_Fix	0	0	0
PSP00049631	07/10/2009 10:58:25	SMSB Aurea		Will_Not_Fix	0	0	0
PSP00049428	29/09/2009 12:13:49	IA for SM 6.1 (Dagobah)		Will_Not_Fix	8	2	0
PSP00049302	24/09/2009 15:12:20	SM Athena		Test_Passed	0	0	0
PSP00049179	17/09/2009 14:54:02	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00048888	01/09/2009 13:25:53	SM Athena		Will_Not_Fix	0	0	0
PSP00048235	17/07/2009 16:01:35	SpeechMagic V6.1		Test_Passed	0	0	0
PSP00048183	13/07/2009 15:21:15	SMSB Aurea		Test_Passed	0	0	0
PSP00048108	08/07/2009 09:38:46	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00048057	06/07/2009 10:43:51	SpeechMagic V6.1		Will_Not_Fix	0	0	0
PSP00048053	03/07/2009 14:03:26	SpeechMagic V6.1		Test_Passed	0	0	0
PSP00047954	30/06/2009 14:16:25	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00047783	19/06/2009 10:34:11	IA for SM 6.1 (Dagobah)		Test_Passed	0	0	0
PSP00047777	18/06/2009 17:25:20	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00047528	04/06/2009 15:40:55	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0
PSP00047396	27/05/2009 13:50:34	IA for SM 6.1 (Dagobah)		Will_Not_Fix	0	0	0

Page: 1 Total Pages: 5 Total Records: 414

Figure 33: View of a query in ClearQuest

To find out who owns what entries or which entries were submitted by a certain person a query needs to be created as shown in Fig. 34.

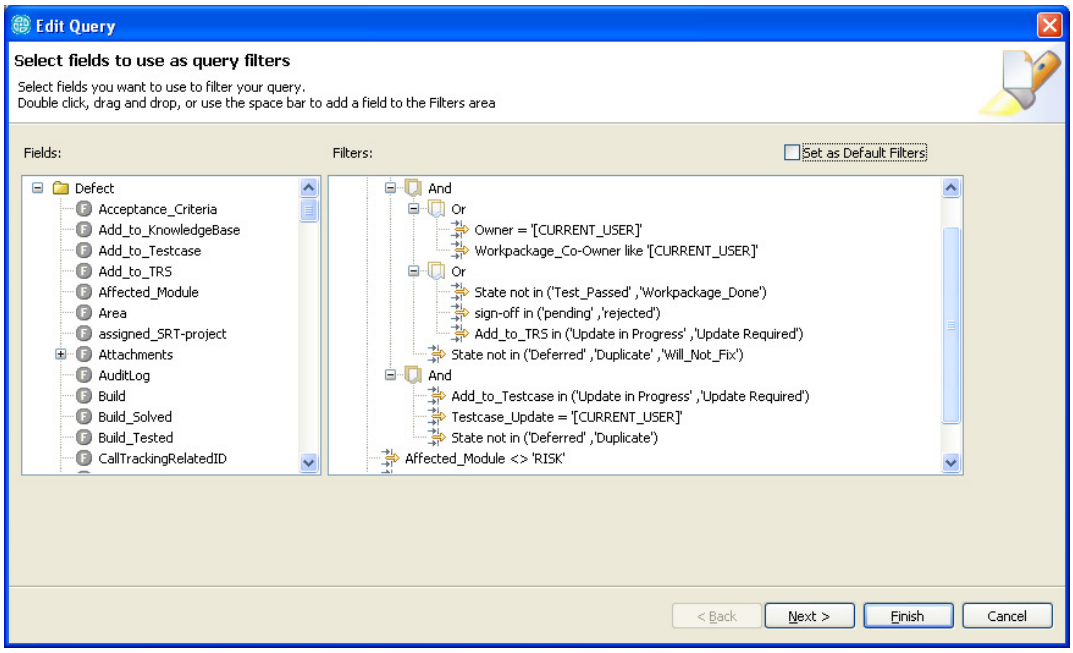

Figure 34: ClearQuest query for currently logged in user=owner

Of course also the look and feel to create queries differs among all tools. As an example the basic query options in the Oracle forms service request module are shown. Of course also Update Seven works differently too as can be seen in Fig. 35.

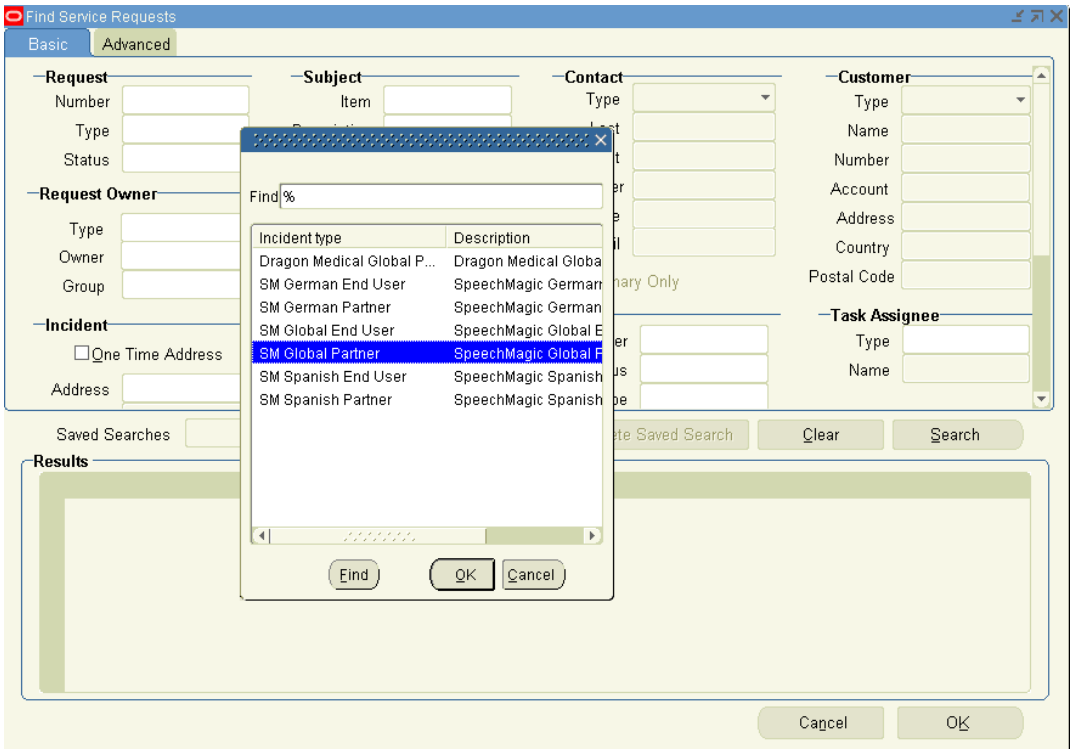

Figure 35: Basic query in Oracle forms service request

Apart from being cumbersome the information is always only from one layer to the next. To really find out the whole tree of relations and dependencies one would have to query several times.

In comparison finding the related people to an entry and related entries is simple in Global Joint Artifacts. For most standard questions there is no need to traverse the graph in any special way. Who submitted the artifact, who is currently the owner, and who is interested in updates (observers) is anyways always visible.

Figure 36 gives an example of the Global Joint Artifacts user interface.

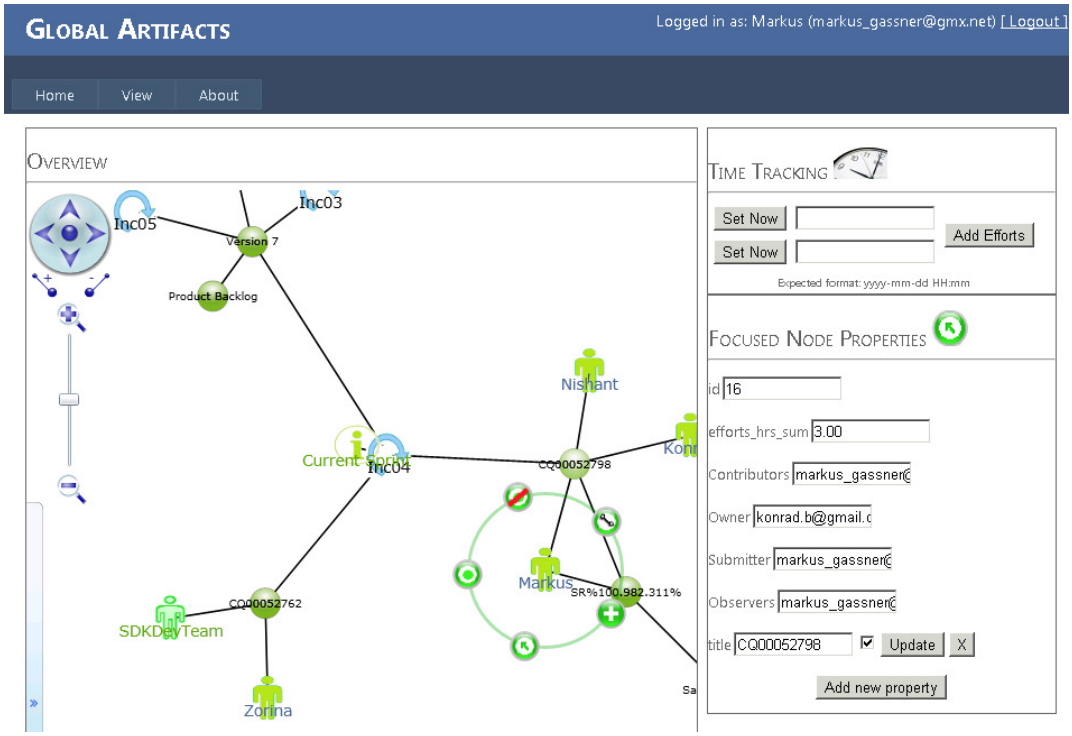

Figure 36: Clear relation between users and artifacts

Following the tree horizontally up or down allows to see the relations in the same e.g. iteration. This then allows to see from the tester who executed a test, who wrote the test, who entered a defect found during the test, up to what customer requested that feature.

At the same time vertically linked artifacts allow to find who was already working on similar work and might possess valuable knowledge on the work at hand.

5.2.2.2 Unified access to data

Global Joint Artifacts combines the data from several systems. It does not attempt to replace the other systems, but instead be a combined view on the available information from different information systems.

But this mechanism is not limited to reading and presenting information. Information could also be passed down into the different systems in a unified way. Time tracking is picked here as an example.

Time is tracked in all the systems for different purposes. Figure 37 shows a mashup of the timetracking Excel sheet, the efforts tab on ClearQuest and the efforts in Oracle forms service requests. Additionally there also exists a project controlling module where time needs to be tracked as well.

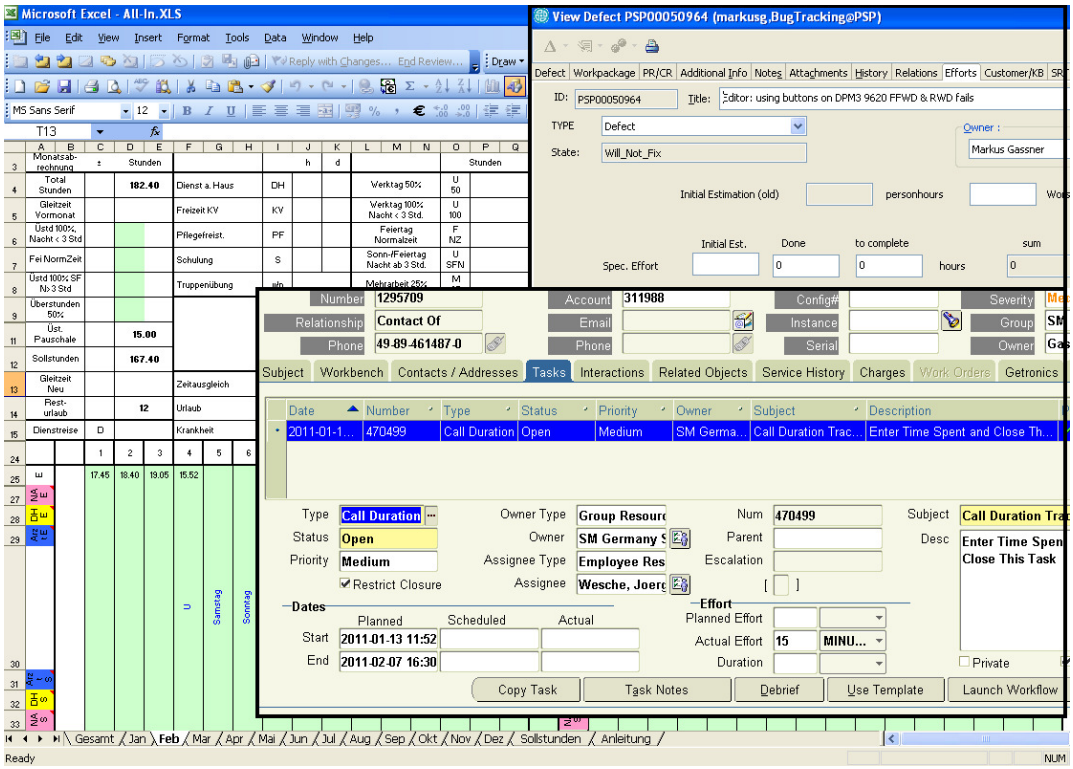

Figure 37: Mashup of legal, service request and effort time tracking

No matter how the time tracking data is used, for those performing that chore, there should be only a single point of entry. In Global Joint Artifacts all edges between a person (isPerson=true) and an artifact can contain efforts.

Figure 38 gives a look behind the scenes of a neo4j relation between a user and an artifact.

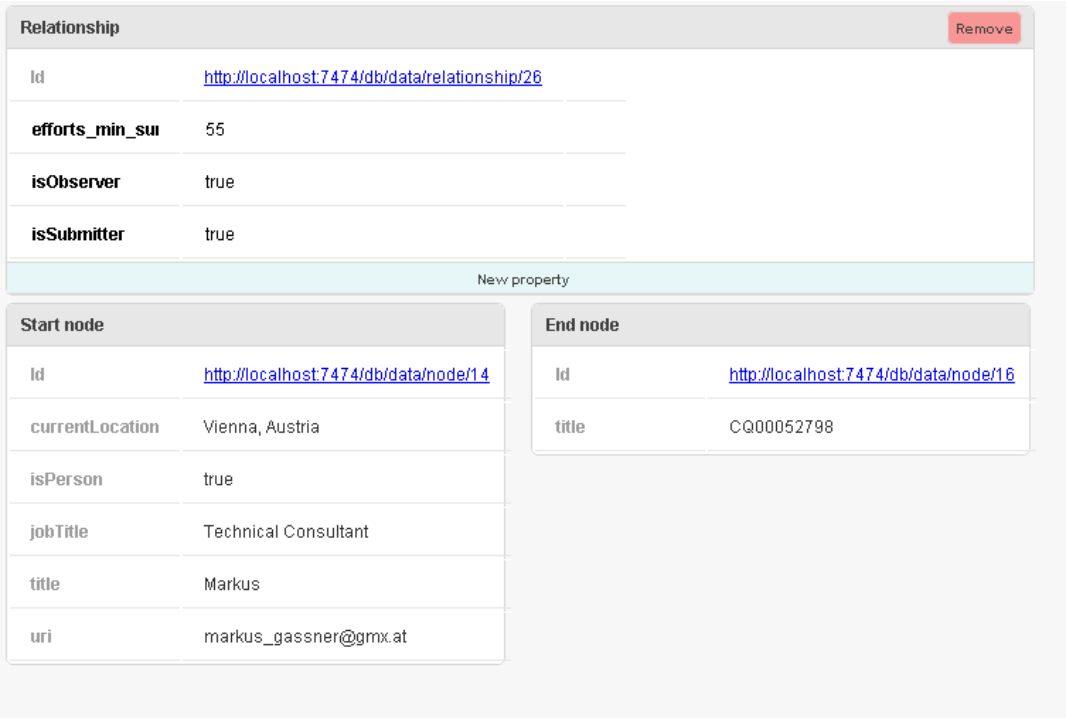

Figure 38: neo4j view at the relation between a user and an artifact

5.2.2.3 Gaining an edge on information

Retrieving information and turning it into knowledge is often time-critical, especially when customers are involved. For employees in the 3rd line / technical support department it would prove invaluable if they could quickly find out who was testing a certain module, e.g. the setup routines, in a certain iteration of the product.

For testers it will be interesting to know when a developer fixed a bug that they entered. In the traditional system it is the responsibility of the tester to log into the defect tracking system and check for new work that came in. Often it proves useful that people meet in the hallway discussing the immediate work at hand. But it is not possible that the project lead will meet by co-incidence both the test lead and a concrete tester in the hallway for an ad-hoc meeting.

When a user changes attributes on an artifact, these changes can be promoted up to the observers of the artifact. An example would be that a developer sets a defect to “Fixed”. This change will be promoted up to the network(s) and all people interested in that artifact, called observers, will be notified in their activity stream – just as if the developer had manually logged in and written “I just updated ‘CQ00052798’. I set ‘status’ to ‘Fixed’.”

5.2.3 Summary

For users connected to artifacts the uniform system creates benefits in terms of saved time, informational advantage and increased quality of their decisions.

First as means of easing their day to day work the users only have a single point of entry for efforts with Global Joint Artifacts.

At the same time it is always obvious who is related to an artifact, thus could be a valuable source of information. All carriers of knowledge are easily findable. Not only the current owner can be retrieved, but especially also the original submitter and all previous owners of the artifact.

And finally immediately after a change on an artifact everyone interested in this artifact is informed through the messages in his or her favorite social network.

This chapter presents some objective and measurable tests against the above claimed advantages of Global Joint Artifacts over the traditional systems. As the whole topic of this work is knowledge management in outsourced software test, also the evaluation is performed like a test case.

As measurement the steps needed to fulfill the task at hand are used. Time is sometimes given as reference, but not as measurement. The Global Joint Artifacts prototype only contains a small subset of the data of the traditional systems and runs locally, so it would be unreasonable to directly compare response times. Still, the more steps are involved in different UIs the more time can be assumed wasted on by the user.

Future work should also include more extensive testing and evaluation. Tests should also be written and performed by peers to further objectify the test case set. A mirrored or very large test data set from the live data would allow to not only compare the steps but also the actual time needed in the traditional system to the Global Joint Artifacts. Also open for future work would be an extensive evaluation by actual users and thorough usability testing, as such a system also contains a lot of non-tangible value that cannot be measured directly, but could be made measurable by user feedback. But this demands extensive user involvement, which is infeasible due to time constraints at this point in time.

Usually a test case would consist of test case identification number, a title, the steps to be performed and the expected result. For clarity the tests are reduced to the test case title and steps themselves and the expected result is deemed obvious from the step description (e.g. login means that the user provides username and password, hits enter and is then logged into the system).

The tests are formulated from the perspective of a user.

Show my service requests:

Traditional system:

1. Open https://oracle.thesoftwarecompany.com/
2. Open Apps Logon Links
3. Open E-Business Home Page
4. Login
5. Open Customer Support Engineer for Projectname module
6. Open Universal Work Queue
7. Expand Service Requests list
8. Select My Service Request

Global Joint Artifacts:

1. Open GJA website
2. Login
3. Select current user (http://localhost:1469/View.aspx?id1=14)

Measuring the seconds that passed shows that Oracle needs 218 seconds for that task while it only takes 16 seconds in Global Joint Artifacts.

Show who else is related to one of the defects (e.g. CQ00052798) that were entered by me:

Traditional system:

1. Open ClearQuest
2. Login
3. Use public filter to query for a specific entry, id=CQ00052798
4. Scroll through the notes log (large, plain text field in the UI that contains all modifications to a CQ entry with the person who changed it and the modification timestamp) and write down all people who were involved in that defect

Global Joint Artifacts:

1. Open GJA website
2. Login
3. Select current user (http://localhost:1469/View.aspx?id1=16)

ClearQuest, Oracle Service Request and Global Joint Artifacts allow to view the owner and submitter of an entry. Global Joint Artifacts additionally allows people to be observers, so they get updates on changes of the entry.

Show me only my currently assigned entries in Oracle and ClearQuest:

Traditional system:

1. Open https://oracle.thesoftwarecompany.com/
2. Open Apps Logon Links
3. Open E-Business Home Page
4. Login
5. Open Customer Support Engineer for Projectname module
6. Open Universal Work Queue
7. Expand Service Requests list
8. Select My Service Request

9. Open ClearQuest
10. Login
11. Select public query Owned by me

Global Joint Artifacts:

1. Open GJA website
2. Login
3. Select current user (http://localhost:1469/View.aspx?id1=16)
4. Only show relations of type "Owner" (not implemented yet)

When viewing a test case chapter (e.g. XCopy-deployment), how many steps does it take to find out which person of what partner company requested that feature?:

Traditional system:

1. Open the setup test case document from its location on Clear-Case:\Project\Version7\Test\TestCase_Setup.doc through the Windows Explorer
2. Go to XCopy-deployment chapter within the document
3. Lookup the product requirement ID
4. Open the setup product requirement document from its location on Clear-Case:\Project\Version7\PR\PR_Setup.doc through the Windows Explorer
5. See who was the original author and who is the current owner of the document in its attributes
6. Details on the feature can be found in the document, but usually not the origin of this requirement
7. Arrange a meeting with current owner of the document to discuss where this re-quirement derives from
8. In case this is not an information from the current, but a previous owner, those previous owners also have to be questioned

Amount of peers involved: 1-3 Global Joint Artifacts: 1. Open GJA website 2. Login 3. Select Setup Testcase 4. -12. Traverse down the tree (8 nodes) Amount of peers involved: 0
I want to find out when a defect I entered is set to Fixed: Traditional system: 1. Open ClearQuest 2. Login 3. Select CQ entry 4. -10. Repeatedly login until the entry is set to fixed. Global Joint Artifacts: 1. Open favorite OpenSocial enabled social network, like Xing, LinkedIn or Orkut (currently only Orkut supported) 2. Login 3. -9. Repeatedly login until the entry is set to fixed.Once the developer updates The tester or support engineer does not know when the developer will fix that issue. So the user has to log in e.g. every day during the next week in both systems. But it is assumed that the user is anyways often or even always online in his favourite social network. So getting the information is just a side-product while logging into ClearQuest and selecting that particular entry is an additional task.

Table 8: Test cases for person-artifact-relationship

5.3 Hypothesis 2

Increased benefit when connecting artifacts with artifacts

5.3.1 Detailed Problem Description

Only if the whole picture is visible in a single view (like Global Joint Artifacts) the background, preconditions, and context become really obvious.

In traditional social networks users connect for various personal and professional reasons.

Getting artifacts into this picture should not only connect people with their artifacts, but at the same time also related artifacts amongst each other. Not only is the serial connection of the artifacts of importance, but also the parallel relation. So the information is not limited to the sequence within a project, but the connections also span out to other projects, if the artifacts are somehow related.

In the traditional systems a connection between ClearQuest and Oracle forms service request can only be created by adding an additional field in the forms. Below picture shows Oracle forms service request on top and ClearQuest at the bottom.

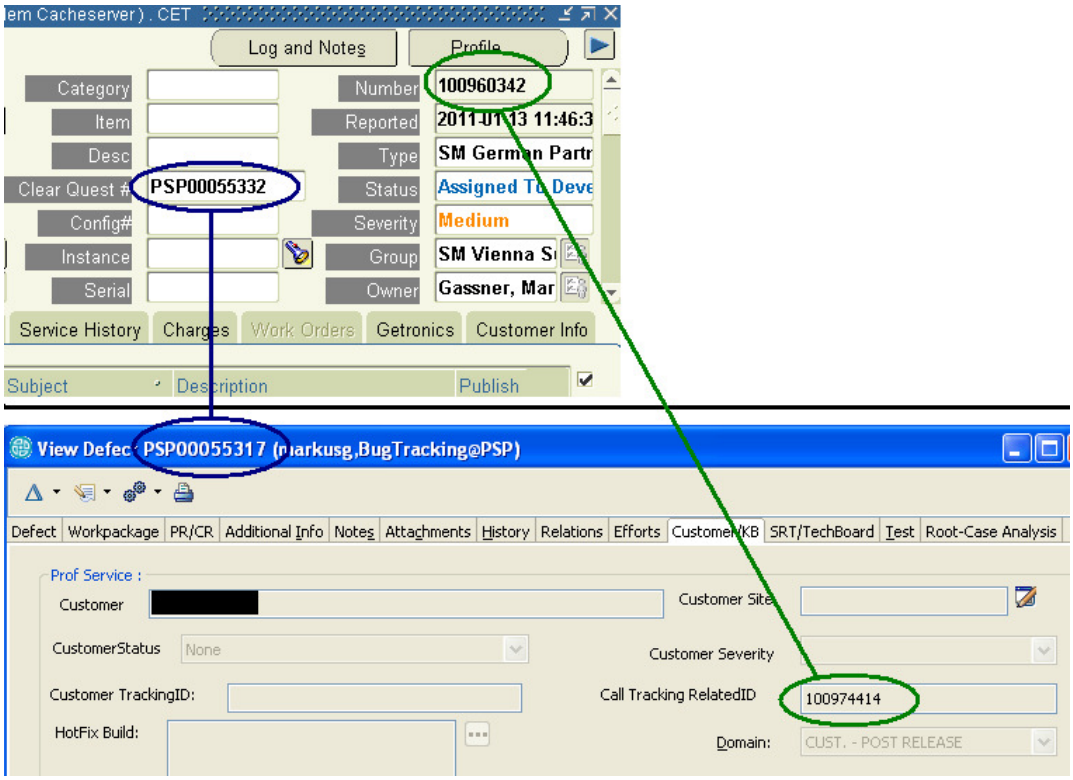

Figure 39: Relations between the systems are done with additional fields

Obvious disadvantages are in this case, that first of all both systems need to be running simultaneously and that going through a larger set of relations means to open and close an abundance of queries. As said all of the systems are not very reactive regarding startup, login and general usability.

A relation of a service request with a concrete defect is easily modeled in Global Joint Artifacts as Fig. 40 shows below.

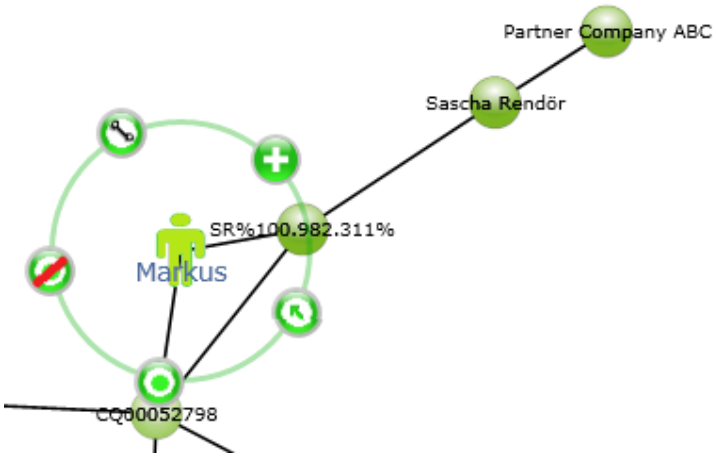

Figure 40: Artifact-Artifact relation in Global Joint Artifacts

In above picture an artifact-artifact relationship is modeled. Sascha Rendör from the partner company ABC submitted a service request with the ID 100.982.311. Technical consultant Markus took over that entry and tried to reproduce the issue.

The issue really is a defect in the product and thus a ClearQuest entry with the ID 52.798 is created.

The ClearQuest entry is scheduled to be fixed in the current sprint, called increment 4.

Apart from linking entries among systems, Global Joint Artifacts of course can also link related artifacts from the same origin. Oracle forms service request for example does not really support connections among entries. In ClearQuest a relation is a collection of numeric entries, which again need to be queried.

In ClearQuest the user has to create a query to find out which entries belong to which increment. In Global Joint Artifacts all entries associated to an increment can be easily spotted.

5.3.2 Verification

The hypothesis is verified by comparing three typical use cases of the traditional systems of the software company in which the author is employed with how this could be handled in Global Joint Artifacts. Again first the chapter elaborates on the background of this hypothesis and then presents concrete test cases and their results.

5.3.2.1 From customer requirements to concrete test case

This chapter discusses how a vague idea from the customer is transformed in a development cycle to a concrete test case at "the software company".

The chain of dependencies will look like this: Product Requirement -> Product Feature -> Technical Requirement -> Test case design -> Test case

Main contact for the customer is usually a product manager or product. The product owner discusses the commercial requirements with the customer, makes them a distinct feature, prioritizes these and adds them to the product backlog. When a product requirement is picked up by development it will be discussed in a larger group to derive technical requirements from it.

A tester takes the technical requirement and writes a test case design against it. The test case design specifies roughly what needs to be tested. From the test case design a test case can be derived, which additionally to what, also specifies how something should be tested and what results are expected for the test to succeed.

The above artifacts will be handled in a heterogeneous set of applications and in traditional systems are usually only connected through IDs that need to be maintained in at least two documents.

When the product owner was in contact with a representative of a customer either via a visit, over the phone or a plain eMail, the information is stored in Update 7 the customer relationship management tool (CRM). The individual chunks of information are reformulated to product requirements. These are then accumulated into a Word document, which gets a unique ID. The word document is then stored in ClearCase, which is the revisioning system, so the document is available to the whole team. For major revisions of such a document, an eMail is sent out to a larger audience, including the whole team and further stake holders and superiors, to announce the availability and scope of the document.

One or several product requirements can be reformulated to a product feature. But it is also possible that a single requirement creates several product features. For ease of handling the features are usually described in the same document only having different IDs.

The technical requirements are what the development team understood from the product requirements and features. For example a product feature "Allows sending eMail notifications" would be translated into one or many technical requirements. In this example it would be that somewhere in the architecture an SMTP server needs to be set up and further technical aspects of the product. Technical requirements are usually in the scope

of a so called module owner, who writes them down in his modules technical requirement document. Often though the specification is delayed to the actual implementation and documented in that work package during development and then stored in ClearQuest. Often here the link to the product requirements is already broken. Because not all work packages have a clear relation to a requirement and so it is sometimes forgotten to keep the product requirement ID, even if there would be a relation.

For each technical requirement module document a test case design should exist. This is just a short document that roughly explains what is to be tested, but not how. For example if a technical requirements document for the setup routines exist, a setup test case design document should exist. Each individual requirement should be addressed in this document.

The actual test case document is often written by another person than the test case design. For the test case design it is important to have a senior test engineer think of exactly what (and equally important, what not) needs to be tested. Going through the description and writing down all the actual details and steps of how to perform the test can also be done by normal testers.

So all this information is not stored in a database, but only large documents, which do not really allow to query for information across larger organisations. It's also nearly impossible to derive where from certain feature requests came from, because the document only has the product owner as author, while the sources are usually not referenced.

All this can be modelled with Global Joint Artifacts to quickly get a full view from top to bottom of the dependency tree.

Global Joint Artifacts does not attempt to replace all the mentioned tools, but be the single interface to retrieve, lookup, distribute and even add or modify data, information and knowledge.

The following diagram (Fig. 41) shows a sketch of the people related to the individual artifacts and more importantly it shows how the artifacts are interconnected amongst each other.

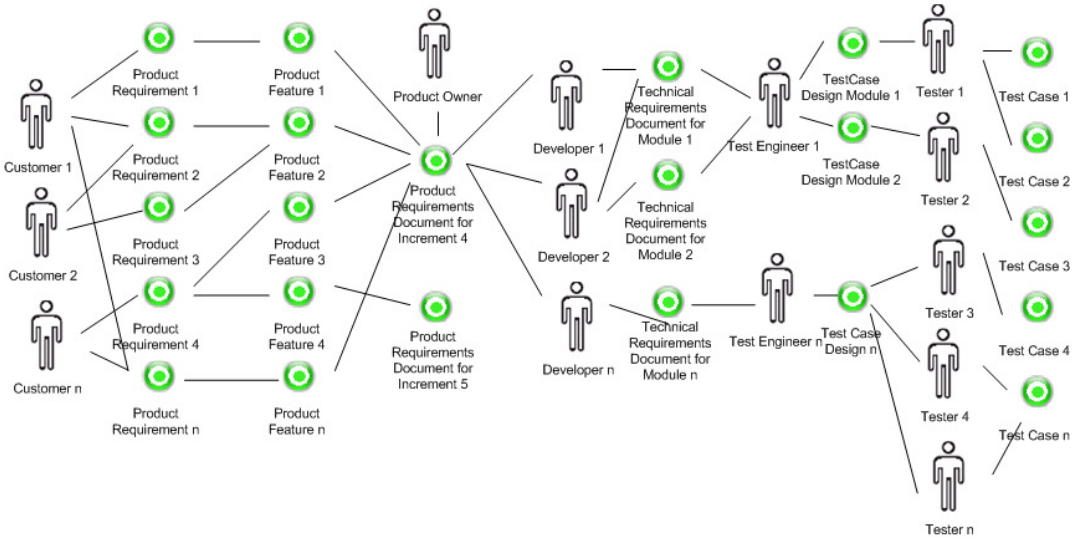

Figure 41: From customers' requirements to the concrete test case

Above clearly indicates the advantage of going up and down the dependency tree. In a test case it is immediately possible to view the related technical requirements. And it is also very easy to drill further into the information, up to the point when and how the feature was requested.

If a partner company is no longer a paying customer, all attached features can be prioritized down and testers might also reduce the

5.3.2.2 Learning from similar test cases

From the relation of test cases amongst each other this chapter discusses how similar artifacts and their responsible people can be used to gain knowledge to a very specific problem.

Artifacts are not only connected serially, which means in a direct order regarding the project structure, as in the previous example, but also in parallel, which means according to their semantic similarity.

A concrete example is connecting test cases of the same kind, like setup tests. Most setups need to verify similar issues, like running low on disk space. Also all MSI and InstallShield Setups share some similarities, which allows a new tester to learn a lot from existing test cases, when the tester should write a test case for a new setup program.

Another example are tests against software that uses specific technology. When test engineers need to write a new test automation for testing a COM component they will be able to learn a lot from existing automation applications and scripts.

Advantage is not only to have already existing artifacts that can be used for learning or even partly re-used, but also the people originally responsible are made visible.

5.3.2.3 Customer support request to regression test

This chapter discusses how a support call submitted by a customer is transformed into a test case for a retest.

The tree of dependencies is structured as Customer Request -> Call Ticket -> Defect Entry -> Workpackage -> Test case.

If the end customer finds a problem he or she will contact some administrator on the site - their first level support. If the problem is beyond the knowledge of the local support they will contact the second level at the partner company. If those supporters cannot help they will turn to the official third level support of "the software company". There is a central web page to submit support cases like defects or service requests. This will be the point where "the software company" learns about the eventual bug.

If the support team can reproduce the problem, it is proofed to be a defect in the software and not just a user error. Often the external system to track customer relations is different to the internal defect tracking system. So the support team member has to create another entry in the defect tracking system. This defect is then usually discussed by the development team, the problem is analyzed, the efforts for a fix estimated and then it’s scheduled.

At that point in time a test case design and/or a concrete test case for regression tests can be written. It will be used for reproducing the bug, retesting the initial fix and then later for regression tests throughout future development iterations, to make sure that particular problem will never be re-introduced to the product.

As always connecting the artifacts horizontally would allow the tester to backtrack the whole history up to the original case. At the same time the support could follow up on how the test department intends to verify that this defect cannot occur any more in the prodcut. This already would allow both teams to easily cooperate, without the need to directly talk face to face.

Even more important would be to be able to connect the artifacts vertically. Support engineers could interconnect similar entries. This would give development and in the end the test engineer who implements the test case a more complete view on the prob-

lem. Connecting artifacts vertically at the end of testing would also increase the amount of knowledge as similar test cases could be combined to a comprehensive test suite.

5.3.3 Tests and result summary

The tests are again formulated from the perspective of a user.

When viewing a product requirement (e.g. XCopy-deployment), how many steps does it take to find out if this feature is already covered by a test case:

Traditional system:

1. Open the setup product requirement document from its location on ClearCase:\Project\Version7\PR\PR_Setup.doc through the Windows Explorer
2. Go to XCopy-deployment product requirement
3. Usually there is no direct reference from the PR to the test case (only vice versa)
4. Open a test case document from its location on ClearCase:\Project\Version7\Test*.doc through the Windows Explorer that fits the topic, in this case it would be the Testcase_setup.doc
5. Search for the appropriate test case step

Global Joint Artifacts:

1. Open GJA website
2. Login
3. Select Setup Testcase
4. -12. Traverse down the tree (8 nodes)

When viewing a test case, how many steps are necessary to retrieve a similar test case:

Traditional system:

1. Open a test case document from its location on ClearCase:\Project\Version7\Test\Testcase_Setup.doc through the Windows Explorer.
2. If this is based on a test case of a previous product version it is usually referenced in the meta data section of the document. It is not intended to add references to other similar test cases from other projects though.
3. Open a test case document from its location on ClearCase:\Project\Version6\Test\Testcase_Setup.doc through the Windows Explorer.

Global Joint Artifacts: 1. Open GJA website 2. Login 3. Select Setup Testcase 4. Check for linked/related test cases
After a reported defect was fixed the support engineer would like to verify that a regression test for this issue was created: Traditional system: There is a relation between service request and defect entry, but which defects are implemented for regression tests by the test department is not directly tracked. Global Joint Artifacts: 1. Open GJA website 2. Login 3. Select service request 4. -6. Traverse down the tree (3 nodes)
Show me all entries that contain "searchtext" in "description": Full text search is not implemented yet in Global Joint Artifacts thus not evaluated.

Table 9: Test cases for artifact-artifact-relationship

6 Discussion

This chapter compares this thesis with related work. It takes the results of this study and puts them in relation to the results of other papers to show what the common findings are and were findings deviate.

It also shows open and loose ends and shows the strengths and weaknesses of the results and gives an outlook for future work.

6.1 Related Work

In the paper “Global Software Engineering: The Future of Socio-technical Coordination” James D. Herbsleb describes “a desired future for global development and the problems that stand in the way of achieving that vision.”[111] He explains “research and lay out research challenges in four critical areas: software architecture, eliciting and communicating requirements, environments and tools, and orchestrating global development.”[111]

In their article in the IEEE Software Magazine Erran Carmel and Ritu Agarwal propose three methods, to make distributed development manageable. They motivate their work with the following words: „We need to examine how distance contributes to heightened complexity in organizational processes. An organizational unit cannot function without coordination and control; unfortunately, distance creates difficulties in both.“[1]

Nguyen, Hackett, und Whitlock emphasize in their book “Happy About Global Software Test Automation” that one needs a „methodology and/or tool to improve communication“.[112] This will bring clear visibility into the process.

6.2 Prototype’s strength

The prototype allows a one point access to all the provided data. It can connect all data from the call tracking and CRM system, the defect tracking system and the knowledge base.

It visualizes the current sprint and allows to see in one tool all that belongs to the current sprint.

Users are automatically notified of changes without the need for intensive direct communication.

Common tasks can be unified, like shown with the example of time tracking.

6.3 Prototype's limitations

As the core functionality was in the focus of the thesis the following software engineering quality attributes (see chapter 2.2.2 "ISO 9126-1 Product quality -- Part 1: Quality model") are not fully covered by the prototype. Addressing these loose ends would be required in a commercial product.

6.3.1 Security

Global Joint Artifacts is a prove of concept and not a full commercial product. As with many prototypes the development efforts were stopped once a functionality works. The whole software needs an extensive security audit to find its vulnerabilities, starting from cross site scripting to DDoS attacks.

6.3.2 Usability

For commercial grade usability more aspects of the user interface design would have to be discussed and analyzed. It would require extensive user tests to e.g. find out what users really need to speed up their daily work with the product and how to increase their efficiency.

6.3.3 Social Network Abstraction Layer

The prototype only uses the OpenSocial interface of Orkut. Although OpenSocial attempts to be its own common standard there are many versions of it available in the field.

Additionally what is really supported is always up to the container. Sometimes it's not even the REST API, but only the JavaScript interface that only allows to run small gadgets on the container.

Also *the* social network, Facebook, is not utilizing OpenSocial.

With all this in mind an abstraction layer would be needed that would also allow to easier directly interface with the proprietary REST APIs of Facebook, LinkedIn and maybe Xing.

6.4 Outlook

This chapter shows in which academic fields further research could be performed to increase the Global Joint Artifacts' value.

It also lists ideas and suggested features in which direction further development efforts should be spent, to make the system a commercial grade product.

The chapter is concluded by a proposal for alternative applications of the concept and software.

6.4.1 Thorough Study

The thesis is mainly the reasoning and documentation of the practical work and not mere theoretical research. A full comprehensive study with a quantitative analysis on a large set of test data should be done in future work.

6.4.2 Automatic connection retrieval

The field of text matching and searching would be very valuable for Global Joint Artifacts. The idea that artifacts themselves have the capability to search for similar artifacts, was part of the initial proposal, but to keep the scope to a feasible timeframe it was necessary to drop the immediate implementation.

As Zobel and Moffat point out “[..] a system is effective if a good proportion of the first r matches returned are relevant.” [113]

6.4.3 Include non-human resources

Global Joint Artifacts can currently only send updates for people and their artifacts. It would also be easily possible to modify the system to track non-human resources, for example a test server. As a simple workaround for this is already now possible by entering such resources as people and create users on the social network. With this a test server could for example notify through the social platforms that the queued test run completed successfully. Another workaround would be to let the computers use their assigned owners social identity to promote messages like "Test run x on computer WhatEverWorkstationName ran succeeded with Passed=23, Failed=2, Ignored=0.".

Also ITIL (IT Infrastructure Library) recommends in several of its areas to include "information systems" in the organisation to keep track of software and hardware configurations. A system like Global Joint Artifacts could greatly influence not just storing the data, but making it available quickly and efficiently.

Full information on customer sites could be stored in the Global Joint Artifacts as well. It would store what version of our software the site has installed, from which version this was upgraded, how it is configured, how many users there are, what kind of li-

censes they use, how many, and what workstations are there and so on. Accurate information on customer sites would greatly increase the knowledge in a support department.

6.4.4 Prioritization and Service Levels

Similar to different ticket types in Software Kanban artifacts could be prioritized. Mechanisms could be analogous to established queuing standards like FIFO, LIFO, and the like.

Service Level Agreements (SLAs) could be taken into account how soon an artifact needs to be resolved.

The Cost of Delay (Reinertson/Smith) and opportunity costs could also play an important role.

6.4.5 Change over time

It would be interesting to see the change over time of the artifacts. This functionality should work similar to Gephi's ability to drill into the history of the graph. It would be beneficial to analyse a project by simply adjusting the current position on a timeline and having the change immediately reflected above in the graph view. Figure 42 gives a sketch how this could be visualized.

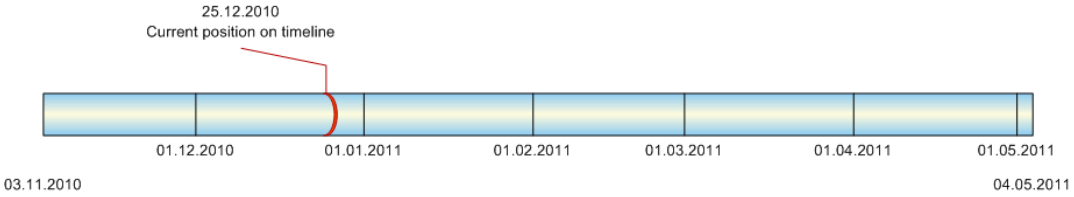

Figure 42: Example for a timeline to analyze the graph's history

6.4.6 Node size

The nodes' size could represent information. Nodes could grow depending on the amount of connections they have, and as such representing their importance in the graph.

They could also represent the hours booked, getting bigger if more hours were spent on the artifact (Fig. 43).

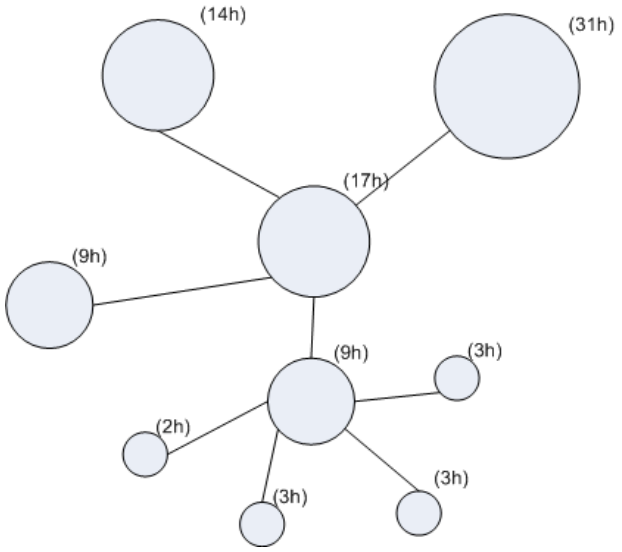

Figure 43: Example for node size representing spent hours

6.4.7 Possible alternative applications

Global Joint Artifacts in this thesis is used for outsourced software test. As pointed out this is a concept that could be easily applied to other fields or professions as well. This chapter proposes a few alternative options in more detail.

6.4.7.1 General workpackage management and time tracking

Global Joint Artifacts could be used to track any kind of tasks that are too big to be completed in one go. As such it could be applied to any kind of project, in any kind of field from private use to commercial or academic application.

6.4.7.2 Sports

In professional sports using technology to measure an athlete's performance and generate metrics is normal habit.

But nowadays even in semi-professional and hobby sports the use of technology and gadgets is becoming widespread. Many people use some sort of app with a GPS-enabled smart phone to track their running exercises for example. How long did it take them to run a certain amount of kilometers?

Global Joint Artifacts could be used to connect freelance trainers and coaches to their clients. With a system like Global Joint Artifacts they could build weekly training schedules and their students can follow the graph of activities.

With well specified metrics also the performance of hobbyists can be measured and evaluated.

6.4.7.3 Social Network Unification

Global Joint Artifacts could also be used to unify several OpenSocial-enabled networks, with a single interface. Unlike the previous options, this would not only require work on applying the concept to a concrete field, but also some further development efforts.

Yet with the knowledge gathered with the prototype it would be possible to generate a common platform for retrieving and exchanging information, not in a single network, but throughout all networks a user is interested, as long as there is a Global Artifact implementation.

7 Conclusion

The following chapter summarizes the thesis. The general idea and concept of the work is given in abridged form and the diploma thesis results are re-iterated.

In its foundation Global Joint Artifacts is a system to organize and share knowledge. It allows connecting people with the so called Global Joint Artifacts. A global artifact is any tangible object that needs to be tracked somehow. This is a very general concept that can be applied to most professional and academic fields.

The thesis focuses on how the concept can be applied to globally distributed software development and testing teams.

First the academic and professional foundations of this concrete application of the concept are discussed.

Knowledge management is the very foundation of the whole discussion of Global Joint Artifacts. Information is an invaluable asset. Organization of information and knowledge is the foundation of many information science related topics. The discussion starts with the definition of what data, information and knowledge really are, followed by the attempt to give a definition for knowledge management.

Knowledge can be abstracted into explicit and tacit knowledge. Explicit knowledge is typically formal knowledge that can be derived from books. Tacit knowledge on the other hand is learned and used in a profession or craft.

With the definition of explicit and tacit knowledge comes the interest to pass on knowledge and also to transform one kind of knowledge to the other.

Knowledge needs to be structured, otherwise it is difficult to share it. This implies the need to also transform tacit to explicit knowledge. Generally the sharing of ideas and thoughts demands communication. And natural, colloquial communication, for example in the hallway over a cup of coffee, is what distributed teams lack most.

A system like Global Joint Artifacts allows to narrow down the distance between people. It also lowers the need for direct communication to gather basic information.

Knowledge management propagates to broadcast information, which should be possible with Global Joint Artifacts.

Connectivism revolves around the assertion that a single person cannot know everything, so a system is needed that allows people to get connected according to their open workpackages.

Software testing imposes all those challenges from knowledge management, as often testers not only work with explicit knowledge, but also internalized tacit knowledge. Testers are also always only part of a larger team and as such need to communicate and collaborate a lot.

Testers should be able to gather quickly a lot of information as they need to not only understand what the development department creates, but also what the customer really wants. Only then they can specify and perform useful tests, as well as set useful priorites in their testing efforts.

The standard IEEE 829 helps testers by specifying standardized and structured documents and as such allows also remote people to know what to look for in which documents. The standard defines documents like test plans, test designs and test cases for the different development levels in the project.

The ISO 9126 standard allows developers and testers to straighten out the terminology and taxonomy they use in their daily work, reducing the need to communicate on every little detail of the project. Many common terms are specified, which also allow people far from the main development brach to follow with relatively low efforts the current activities.

A tool like Global Joint Artifacts should make use of these standards and helps to structure the related documents. The structure and dependencies are then visualized in a very natural graph.

Software development and testing is often separated into distinct layers and these can be best visualized as well as a graph.

Software testing is only part of a software development project. Many development process models exist to successfully finish a project.

RUP and V-Modell XT are traditional and widely used software development process models. They create a lot of artifacts and documents, which a tool like Global Joint Artifacts helps to organize.

The agile methods Scrum and Software Kanban propagate the importance of communication within the team, which of course is a problem in distributed teams. A tool is

needed in this case to limit the amount needed for direct face to face discussion. The people involved in the process need to be able to retrieve information for themselves, and if it is needed to get in touch with people it should be without much effort to find the experts and specialists needed for the job at hand.

All methodologies share some common elements that can be modeled with Global Joint Artifacts: Documents, source code, executables, roles and actual people.

Successfully finishing a software development project is in itself already a challenging task. Globally distributed teams amplify the challenges and problems.

Temporal distance can be overcome by a thourough documentation in a proper tool as face-to-face communication is no longer that important. Even though people are disconnected from each other for several hours, they can still communicate with each other through the artifacts on which they collaborate.

Spatial distance prevents that people meet in person, or not on a regular basis, as it would be with all people working in the same office. Even though people are far from each other they can collaborate on a common goal with Global Joint Artifacts.

What was said above further demands new technologies, as conventional RDBMS store data in flat table-like structures. The interconnected data is more naturally stored in graph-like structures.

Also the amount of data is quickly growing if all, including even the tiniest, parts of a project are stored in a single database. And usually many projects run at the same time. Then those artifacts need to be connected to their respective owners, submitters and observers. A system like Global Joint Artifacts will then even track projects for many customers.

On starting the thesis neither the concept Global Joint Artifacts nor the application existed. The thesis is the theoretical foundation for the Global Joint Artifacts social website project. The fundamentals discussed herein are the theoretical background that supported concretizing the concept and finally the Global Joint Artifacts prototype. In future work an already running Global Joint Artifacts social website could undergo a thorough formal analysis.

The development was performed iteratively. The initial theoretical research influenced the analysis and design of the prototype. Initial implementation of the prototype influenced further theoretical research and so on.

To finally summarize the work currently covers the following areas:

Theoretical research on the related topics. Extending from the general knowledge management area to the concrete application of software development, software test and global software development to its technical foundation, which are NoSQL storages.
Object-oriented analysis of the prototype.
Technical selection of existing technologies and frameworks. The core component uses the neo4j server, a working neo4j .NET wrapper and a suitable graph visualization toolkit (ICE). As the first social network for testing Orkut was used as it is tightly bound to OpenSocial due to the acquisition from Google and being popular in Brazil, one of the BRIC countries.
Design and architecture of the prototype.
Implementation of the prototype for basic investigation purposes. Implemented as a social website with ASP.NET according to the analysis and design using the selected technologies and frameworks.
Test and evaluation of the prototype according to the hypotheses.

Table 10: Thesis Summary

List of references

[1] E. Carmel and R. Agarwal, "Tactical Approaches for Alleviating Distance in Global Software Development," *IEEE Software,* 2001.

[2] M. T. Hansen*, et al.*, "What's Your Strategy for Managing Knowledge?," *Harvard Business Review,* 1999.

[3] J. Highsmith, *Agile Software Development Ecosystems*: Addison-Wesley Professionals, 2002.

[4] 2010-07-03). *Conference Acceptance Ratio Statistics*. Available: http://www.adaptivebox.net/CILib/CICON_stat.html

[5] O. Scharmer, *Addressing the blind spot of our time* vol. Executive Summary of the book "Theory-U", 2007.

[6] A. Smith, *The Wealth of Nations*, 1776.

[7] D. Ricardo, "The Principles of Political Economy and Taxation," 1821.

[8] W. R. Kendall and R. C. Scott, "Information As a Factor of Production," *Journal of Information Technology Management,* vol. 1, 1990.

[9] 2010-07-21). *Dictionary Definition of "Data"*. Available: http://www.merriam-webster.com/dictionary/data

[10] 2010-07-21). *Dictionary Definition of "Information"*. Available: http://www.merriam-webster.com/dictionary/information

[11] 2010-07-21). *Dictionary Definition of "Knowledge"*. Available: http://www.merriam-webster.com/dictionary/knowledge

[12] H. Willke, *Systemisches Wissensmanagement*, 2nd ed. Stuttgart: Lucius & Lucius, UTB, 1998.

[13] R. L. Ackoff, "From Data to Wisdom," *Journal of Applied Systems Analysis,* vol. 16, pp. 3-9, 1989.

[14] F. Bouthillier and K. Shearer, "Understanding knowledge management and information management: The need for an empirical perspective," *Information Research,* vol. 8, 2002.

[15] I. Nonaka, "The Knowledge-Creating Company," *Harvard Business Review,* pp. 96-104, 1991.

[16] I. Nonaka, "The Knowledge-Creating Company," *Harvard Business Review,* p. 98, 1991.

[17] G. Siemens. (2004, 2010-07-20). *Connectivism: A learning theory for the digital age*. Available: http://www.elearnspace.org/Articles/connectivism.htm

[18] K. Stephenson, "What Knowledge Tears Apart, Networks Make Whole," *Internal Communication Focus,* p. 1, 1998.

[19] A.-L. Barabási, *Linked: The New Science of Networks*: Plume, 2003.

[20] J. Bach. (2005, 2010-10-19). *Against Certification*. Available: http://www.satisfice.com/blog/archives/36

[21] IEEE, "IEEE 829-2008 Standard for Software and System Test Documentation," ed. New York, USA: IEEE, 2008.

[22] ISO, "Product quality -- Part 1: Quality model," vol. 9126, ed: ISO, 2001.

[23] C. Bunney. 2010-11-05). *Wikipage on ISO9126*. Available: http://www.chrisbunney.com/wiki/index.php/ISO_9126

[24] R. Black, *Advanced Software Testing Vol. 1*: Rocky Nook, 2008.

[25] imbus, "ISTQB Certified Tester - Advanced Level: Testmanager Manuskript," vol. Vol. 1.5, ed, 2006.
[26] B. Galen. (2007) Making the Case: Develop Your Defense For Test Automation. *Software Test & Performance Magazine*. 24ff.
[27] E. Dustin*, et al.*, *Automated Software Testing*: Addison Wesley, 2007.
[28] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, 10th ed.: John Wiley & Sons, 2009.
[29] *A Guide to the Project Management Body Of Knowledge (PMBOK® Guide)*, 4th ed.: Project Management Institute, 2008.
[30] S. Müller. (2004, 2010-07-30). *Was ist ein Vorgehensmodell?* Available: http://v-modell.iabg.de/index.php?option=com_content&task=view&id=11&Itemid=47
[31] W. W. Royce, "Managing the Development of Large Software Systems," *Proceedings of IEEE WESCON,* vol. 26, 1970.
[32] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd ed.: Addison-Wesley Professional, 2003.
[33] "Rational Unified Process: Best Practices for Software development Teams," *Rational Software White Paper,* vol. TP026B, 2001.
[34] A. Spillner*, et al.*, *Praxiswissen Softwaretest: Testmanagement*. Heidelberg: dpunkt.verlag, 2006.
[35] H. Kniberg and M. Skarin, *Kanban and Scrum: Making the most of both*, 2010.
[36] F. M. o. t. I. o. Germany, "V-Modell XT Dokumentation," 1.3 ed, 2009.
[37] F. M. o. t. I. o. Germany, "Standards and Architectures for eGovernment Applications (SAGA)," 4 ed: Federal Ministry of the Interior of Germany, 2008.
[38] A. Spillner*, et al.*, *Software Testing Foundations: A Study Guide for the Certified Tester Exam*, 2007.
[39] C. Bartelt*, et al.*, "Modellbasierte Entwicklung mit dem V-Modell XT," *Objekt Spektrum,* 2005.
[40] F. Stimpfl, "Bridging the Gap between Management and Engineering: An Extension of the V-Modell XT Framework," Master Master Thesis, Institut für Softwaretechnik und interaktive Systeme, Vienna University of Technology, Vienna, 2007.
[41] K. Schwaber, *Agile Project Management with Scrum*: Microsoft Press, 2004.
[42] J. Levinson, *Software Testing with Visual Studio 2010*: Addison-Wesley Professional, 2010.
[43] K. Schwaber and M. Beedle, *Agile software development with Scrum*: Pearson Education, 2008.
[44] M. Cohn, *User stories applied: for agile software development*: Addison Wesley, 2004.
[45] H. Takeuchi and I. Nonaka, "The New New Product Development Game," *Harvard Business Review,* 1986.
[46] K. Beck*, et al.* (2001, 2010-10-18). *The Agile Manifesto*. Available: http://www.agilealliance.org/the-alliance/the-agile-manifesto/
[47] B. K. Jayaswal and P. C. Patton, *Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software*: Prentice Hall, 2006.
[48] M. Andrezak*, et al.*, "Gedämpfte Schritte: Evolutionäre Entwicklung durch Software-Kanban," *iX,* pp. 108-112, 2010.

[49] D. J. Anderson and E. Schragenheim, *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*: Prentice Hall, 2003.
[50] C. Mann and F. Maurer, "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction," 2005.
[51] D. J. Anderson, *Kanban*: Blue Hole Press, 2010.
[52] J. Freiberger. (2010) Auf neuen Wegen ans Ziel: Die Methoden der agilen Softwareentwicklung. *dotnetpro*. 20-26.
[53] E. Ó. Conchúir*, et al.*, "Exploring the Assumed Benefits of Global Software Development," presented at the IEEE International Conference on Global Software Engineering, Florianopolis, Brazil, 2006.
[54] E. Carmel, *Global Software Teams: Collaborating across Borders and Time Zones*. New Jersey: Prentice Hall, 1999.
[55] H. Holmstrom*, et al.*, "Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance," presented at the IEEE International Conference on Global Software Engineering, Florianopolis, Brazil, 2006.
[56] M. Paasivaara and C. Lassenius, "Could Global Software Development Benefit from Agile Methods?," presented at the IEEE International Conference on Global Software Engineering, Florianopolis, Brazil, 2006.
[57] R. Patton, *Software Testing*, Second Edition ed.: Sams, 2005.
[58] L. M. Rocha, "Selected Self-Organization and the Semiotics of Evolutionary Systems," *Evolutionary Systems: Biological and Epistemological Perspectives on Selection and Self-Organization,* p. 3, 1998.
[59] (2010, 2010-12-06). *Hypertable: Architectural Overview*. Available: http://code.google.com/p/hypertable/wiki/ArchitecturalOverview
[60] F. Chang*, et al.*, "Bigtable: A Distributed Storage System for Structured Data," presented at the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Seattle, WA, USA, 2006.
[61] A. Lakshman and P. Malik, "Cassandra - A Decentralized Structured Storage System," presented at the Large Scale Distributed Systems and Middleware (LADIS), 2009.
[62] (2010, 2010-12-05). *RavenDB: What is Raven?* Available: http://ravendb.net/documentation/docs-what-is-raven
[63] T. Hannan. (2010, 2010-12-04). *MongoDB Introduction*. Available: http://www.mongodb.org/display/DOCS/Introduction
[64] (2010, 2010-12-05). *Apache CouchDB Technical Overview*. Available: http://couchdb.apache.org/docs/overview.html
[65] (2010, 2010-12-02). *HypergraphDB Homepage*. Available: http://www.kobrix.com/hgdb.jsp
[66] (2010, 2010-12-03). *InfiniteGraph Technical Product Overview*. Available: http://www.infinitegraph.com/docs/InfiniteGraph-Technical-Product-Overview.pdf
[67] 2010-12-02). *Online Backup High Availability Emulation*. Available: http://wiki.neo4j.org/content/Online_Backup_HA
[68] D. Merriman and A. Lerner. (2010, 2010-12-02). *mongoDB: Sharding Introduction*. Available: http://www.mongodb.org/display/DOCS/Sharding+Introduction

[69] P. Buneman. (1997, 2010-12-04). *Tutorial, PODS '97: Semistructured Data*. Available: http://www.cis.upenn.edu/~db/abstracts/semistructured.html
[70] V. Varallo, *ASP.NET 3.5 Enterprise Application Development with Visual Studio® 2008: Problem - Design - Solution*: Wrox, 2009.
[71] A. Hejlsberg, "The Future of C# (PDC 2008)," Professional Developers Conference 2008 ed, 2008.
[72] W. R. Stanek, *Windows Server 2008 Inside Out*: Microsoft Press, 2008.
[73] L. Scales and J. Michell, *MCSA/MCSE Training Guide (70-290): Managing and Maintaining a Windows Server 2003 Environment*: Pearson Certification, 2003.
[74] (2010, 2010-11-16). *OpenSocial Containers*. Available: http://wiki.opensocial.org/index.php?title=Containers
[75] (2010, 2010-11-16). *OpenSocial Articles & Tutorials: Social Design Patterns*. Available: http://wiki.opensocial.org/index.php?title=Articles_%26_Tutorials
[76] (2010, 2010-11-16). *OpenSocial API Data Requests*. Available: http://wiki.opensocial.org/index.php?title=Requesting_Social_Data_using_JavaScript_(v0.9)
[77] (2011, 2011-02-24). *OpenSocial Tutorials: Social Website Tutorial*. Available: http://wiki.opensocial.org/index.php?title=Social_Website_Tutorial
[78] (2010, *OAuth Community Site*. Available: http://oauth.net/
[79] (2011, 2011-02-24). *OAuth: Using OAuth with the Google Data APIs*. Available: http://code.google.com/apis/gdata/articles/oauth.html
[80] (2011, 2011-02-23). *OAuth Use Cases*. Available: http://wiki.opensocial.org/index.php?title=OAuth_Use_Cases
[81] (2010, 2011-03-02). *OAuth Documentation and Registration*. Available: http://oauth.net/core/1.0/#anchor4
[82] (2010, *JavaScript API Reference*. Available: http://wiki.opensocial.org/index.php?title=JavaScript_API_Reference
[83] 2011-02-04). *Developer's Guide: Server to Server Protocol*. Available: http://code.google.com/apis/orkut/docs/rest/developers_guide_protocol.html
[84] (2011, 2011-02-04). *LinkedIn Homepage*. Available: http://www.linkedin.com/
[85] A. Nash. (2007, 2010-11-08). *LinkedIn & Open Social*. Available: http://blog.linkedin.com/2007/10/31/linkedin-open-s/
[86] 2011-01-14). *What is TS Licensing?* Available: http://msdn.microsoft.com/en-us/library/cc725933(v=ws.10).aspx
[87] (2011, 2011-01-16). *Description of the Credential Security Support Provider (CredSSP) in Windows XP Service Pack 3*. Available: http://support.microsoft.com/kb/951608/
[88] (2007, 2011-01-14). *Terminal Services Gateway (TS Gateway)*. Available: http://technet.microsoft.com/en-us/library/cc731264(WS.10).aspx
[89] (2010, 2010-11-11). *neo4j Homepage*. Available: http://neo4j.org/
[90] P. Neubauer. (2010, 2010-11-11). *Graph Databases, NOSQL and Neo4j*. Available: http://www.infoq.com/articles/graph-nosql-neo4j
[91] T. Cowan. (2010, 2010-11-10). *neo4j on .NET 3.5*. Available: http://www.thewebsemantic.com/2010/06/03/neo4j-on-net-3-5/
[92] (2010, 2010-11-10). *Sones Homepage*. Available: http://www.sones.com/
[93] J. Conwell. (2010, 2010-12-17). *Neo4j REST Sharp on Codeplex.com*. Available: http://neo4jrestsharp.codeplex.com/
[94] (2010, 2010-12-17). *Neo4j REST .NET*.
[95] T. Ivarsson*, et al.* (2010). *neo4j (v1.2.M05) Manual*.

[96] (2011, 2011-02-24). *Multiple Server instances on one machine*. Available: http://docs.neo4j.org/chunked/snapshot/server-installation.html#_multiple_server_instances_on_one_machine

[97] Okram. (2010, 2010-12-09). *Gremlin Wiki on GitHub.com*. Available: https://github.com/tinkerpop/gremlin/wiki

[98] (2010, 2010-12-26). *Neo4j: Visualization options for graphs*. Available: http://wiki.neo4j.org/content/Visualization_options_for_graphs

[99] (2010, 2010-11-10). *neo4j Components*. Available: http://components.neo4j.org/

[100] S. Weibel and C. Lagoze, "An element set to support resource discovery," *International Journal on Digital Libraries,* vol. 1, pp. 176-186, 1997.

[101] D. Hillmann. (2005, 2010-11-10). *Using Dublin Core - The Elements*. Available: http://dublincore.org/documents/usageguide/elements.shtml

[102] A. Powell. (2003, 2010-11-10). *Guidelines for implementing Dublin Core in XML*. Available: http://dublincore.org/documents/dc-xml-guidelines/

[103] D. Blankenhorn. (2008, 2010-08-02). *Will the GPL be overtaken by AGPL?* Available: http://www.zdnet.com/blog/open-source/will-the-gpl-be-overtaken-by-agpl/2133

[104] J. Brinkman. (2010, 2010-11-21). *dotNetNuke 5.6.0 Release Notes*. Available: http://www.dotnetnuke.com/Resources/Blogs/tabid/825/EntryId/2874/DotNetNuke-5-6-0-Released.aspx

[105] IETF, "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)," in *RFC4918*, ed, 2007.

[106] IETF, "Calendaring Extensions to WebDAV (CalDAV)," in *RFC4791*, ed, 2007.

[107] IETF, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)," in *RFC5545*, ed, 2009.

[108] (2010, 2010-11-10). *Google Calendar APIs and Tools Overview*. Available: http://code.google.com/apis/calendar/overview/

[109] (2010, 2010-11-15). *Google Calendar API: Developer's Guide: Extended properties*. Available: http://code.google.com/apis/calendar/data/2.0/developers_guide_protocol.html#ExtendedProps

[110] C. Vicknair*, et al.*, "A Comparison of a Graph Database and a Relational Database," 2010.

[111] J. D. Herbsleb, "Global Software Engineering: The Future of Socio-technical Coordination," *FOSE '07 2007 Future of Software Engineering,* 2007

[112] H. Q. Nguyen*, et al.*, *Happy About Global Software Test Automation*: Happy About, 2006.

[113] J. Zobel and A. Moffat, "Inverted Files for Text Search Engines," *ACM Computing Surveys,* vol. 38, p. 3, 2006.