Markus Hobisch, BSc

Migration of a Jira add-on with regard to security and compatibility aspects

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Development and Business Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl-Ing. Dr.techn. Wolfgang Slany

Co-Supervisor

Dipl-Ing. Annemarie Harzl, BSc

Institute for Software Technology

Graz, August 2017

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

AFFIDAVIT

Abstract

This thesis deals with the theoretical and practical resolution of legacy code in software projects. For this reason, different approaches on how to modernize software are outlined and discussed.

Legacy code leads to unavoidable problems in the entire software industry due to the fact that it is mostly based on outdated technologies and infrastructures. This typically leads to higher costs in the future because systems which contain legacy code must be continuously maintained.
The source code is often badly structured and makes use of outdated programming languages, which do not support modern software constructs and features such as an integration and linking to an online cloud service. Especially the financial sector uses many computer programs based on legacy code. These computer programs still run on mainframe computers and are wide-spread. The mainframe computers are used because they are still working, despite of increasing maintenance costs. Modernizing the systems to new PC architectures like the x86 architecture are rare because of the high effort linked with the upgrade of an existing system. However, as mainframe computers cannot be maintained indefinitely, rising maintenance costs will soon force companies to watch out for new approaches and solutions to these problems.

The outlined approaches in this thesis show how to deal with legacy code theoretically. The various possibilities of modernization are examined and described. We also investigate the costs of using legacy code. Several factors are analysed which could lead to higher maintenance costs.

The practical part describes the modernization of an existing program, specifically the migration of a Jira add-on to the current major Jira version 7. The migration approach was chosen due to higher efficiency and time saving reasons. During the thesis, also different migration strategies are explained and occurring problems are analysed and discussed. As a result of the migration process, the timesheet add-on can now be used for other projects due to its compatibility with the newest Jira version.

Keywords
timesheet, Jira, add-on, modernization, legacy code, migration

Kurzfassung

In dieser Masterarbeit geht es um die theoretische wie praktische Behandlung von Legacy Code in Softwareprojekten. Es werden verschiedene Ansätze der Modernisierung von Softwaresystemen vorgestellt und diskutiert.

Legacy Code führt zwangsläufig zu Problemen in der gesamten Softwareindustrie, weil es meist auf veralteten Technologien und Infrastrukturen basiert. Das führt in der Regel zu höheren zukünftigen Kosten, weil Softwaresysteme, welche Legacy Code beinhalten, laufend gewartet werden müssen. Der Sourcecode ist oftmals schlecht strukturiert und zumeist werden veraltete Programmiersprachen verwendet, welche moderne Konstrukte und Features wie z.B. Cloud-Anbindung nicht unterstützen. Besonders in der Finanzindustrie sind veraltete Computerprogramme häufig anzutreffen. Softwareapplikationen, welche Legacy Code beinhalten, werden noch immer aus Kostengründen eingesetzt. Diese Abhängigkeit führt jedoch zu teilweisen hohen und zeitlich tendenziell steigenden Wartungskosten.

Modernisierungen auf neuere PC-Architekturen wie die x86-Architektur sind rar und werden aufgrund des mit der Umstellung verbundenen hohen Aufwands oftmals nicht durchgeführt. Da Mainframes jedoch nicht ewig eingesetzt werden können, suchen Unternehmen nach neuen Lösungen, um das Problem zu lösen.

Die hier vorgestellten Ansätze beschreiben zunächst theoretisch, wie man am besten mit Legacy Code umgeht. Ebenso werden die, durch veraltete Computerprogramme entstehenden Kosten, erörtert und die entscheidenden Faktoren, welche zu den höheren Wartungskosten führen, eruiert und analysiert.

Der praktische Teil beschreibt die Modernisierung eines bestehenden Programms, der Fokus lag speziell bei der Migration eines Jira Add-ons auf die aktuelle Jira-Version 7. Wir entschieden uns aufgrund höherer Effizienz und schnellerer Durchführbarkeit für den Migrationsansatz. Im Zuge der Arbeit werden verschiedene Migrationsstrategien vorgestellt und aufgetretene Probleme analysiert und diskutiert.

Als Resultat des Migrationsprozesses kann das Add-on in weiteren Projekten verwendet werden, weil es nun mit der aktuellen Jira Version 7 kompatibel ist.

Schlagwörter: Jira, Add-on, Modernisierung, Legacy Code, Timesheet, Migration

Acknowledgements

First, I would like to express my gratitude to my supervisors, Annemarie Harzl and Wolfang Slany, for their excellent support and assistance during the entire process of working on this thesis.

I would also like to thank my fellow students (Christof, David and Philipp, Florian and Berni). It was a great collaboration and we had a lot of fun together.

Especially I would like to thank my parents who have always supported me over the years and without whom I would not be able to stand here where I am today. But I also want to thank my closest friends and my roommate Florian who always had an open ear when I needed some help.

Thank all of you so much!

Graz, August 2017

Contents

List of Figures

List of Tables

1 Introduction

Many companies still use outdated technologies, e.g. old programming languages. Many of these are no longer supported, are regarded as completely obsolete or are replaced by newer technologies. New technologies are easier to extend and save developers a lot of time and headaches. However, many companies still use outdated technologies because they often have no other choice. A complete new development is often far too expensive and involves a high risk of a total failure. Old knowledge about the software architecture can often no longer be reconstructed and is lost forever. Programmers from the past are now retired and therefore are no longer available to companies. Therefore, many companies are opting for a suitable modernization strategy. These strategies help companies to modernize their existing implementations. Various types of modernization are available and can be used depending on the application scenario. Each strategy brings its advantages and disadvantages and must be carefully thought through. Some of them are more risky than others depending on how much source code can be copied from the existing system and how much code has to be written anew. To develop a new system from scratch is always very risky and the code has to be thoroughly tested before it works as it should do.

In this thesis we deal with the following questions:

- What is a Legacy Information System?
- What types of modernization do exist?
- What types of migration do exist and which risks could arise in such software systems?
- What costs can arise for businesses when they keep using outdated systems, in particular outdated code?
- How to deal with legacy code and how to avoid it?

In the practical part, we describe the migration process of a Jira add-on, developed by Adrian Schnedlitz (Schnedlitz, 2016), from Jira version 6 to the current version 7. We describe the type of migration which was used and what problems occurred. Several types of migration were compared and the most suitable one selected. Furthermore, we will discuss security issues and how these can be minimized. Questions about compatibility, stability and performance are also illustrated and answered.

In Chapter 2, we examine the question of what is meant by modernization and what types of modernization exist. The risks of modernization strategies are explained. The main focus is on the migration of legacy code. Migration techniques such as Database First, Database Last or Butterfly Methodology are explained in detail.

Chapter 5 discusses the requirements for the Jira add-on. In essence, this is about the migration of the add-on. But other requirements such as performance, security, stability, maintainability also play an important role. Finally, brief reasons are given, why it was originally important to implement this add-on.

Chapter 3 answers the question of how to deal with existing legacy information systems. Here, an insight into the business world is given and it is explained how banks and insurance companies deal with LIS. The various types of costs incurred by maintaining the systems are listed.

Chapter 4 explores the concept of technical debt and illustrates how this concept can help companies to reduce costs. Furthermore, strategies are presented that lead to better code quality which help developers better in dealing with LIS.

Chapter 6 is all about the implementation. Here information about the concrete implementation of the migration process is given. Further aspects such as safety aspects or compatibility are discussed.

Chapter 7 summarizes the essentials and gives a brief outlook on future possibilities of extending the plug-in.

1.1 Glossary

Application Programming Interfaces (API)

An API is an external interface that allows other computer programs to exchange data. An API usually consists of protocols, routines, functions and/or commands (Technopedia, 2017b).

Asynchronous JavaScript and XML (AJAX)

With AJAX interactive web applications can be created that behave very similar to desktop applications. It is a client-side web technology and the transfer of data is asynchronous (Technopedia, 2017c).

Atlassian Plugin Software Developer Kit (APSDK)

APSDK is a development kit which is used by developers to create new add-ons for the Jira platform[1].

Catrobat

Catrobat is the organisation where the Pocket Code application is developed. Wolfgang Slany is the founder of it and the head of the Institute for Software Technology (IST) at Graz University of Technology.

Comma Separated Values (CSV)

CSV is a data format were data is separated by comma. It is very useful for exchanging data between applications (Webopedia, 2017).

Confluence

Confluence is a knowledge management system where information about the project can be easily shared. Blog posts can be created and shared with the team. All Atlassian products work together. Thus, there is a smooth transition between Jira and Confluence. In Confluence you can access all of your Jira tickets and include them in your blog posts. These two products

[1] https://developer.atlassian.com/docs/getting-started (accessed 2017-04-07)

are also used in our Catrobat team and support the students in their work (Confluence, 2017).

Extensible Markup Language (XML)

XML is a simple text format that can be read by people. It is extensively used for the permanent storage of information and the exchange of data between computers.

Fourth Generation (Programming) Language (4GL)

A group of programming languages that provides more programmer-friendly instructions and improve the overall efficiency through the usage of graphical interfaces and symbolical representations (Technopedia, 2017d).

Git

Git is an open source version control tool which helps to manage code in a easy way (Git, 2017).

Github

Github is a plattform where developers can share their code. Other developers can contribute to the project[2].

Graphical User Interface (GUI)

A GUI is a graphical representation of a computer program and helps user to facilitate the handling of those programs. The opposite are commandline tools.

HyperText Transfer Protocol Secure (HTTPS)

HTTPS is a protocol that ensures secure sharing of data across the world wide web. The secure socket layer (SSL) protocol or the transport layer security (TLS) is used as an encryption protocol (Technopedia, 2017e).

[2]https://github.com/ (accessed 2017-06-17)

JavaScript (JS)
JavaScript is a client-side scripting language used to display dynamic content in web browsers.

Jira
Jira is an issue and project tracking tool which helps developer teams to improve their collaboration. In Jira, tickets can be created and tracked trough the whole development process. Bugs can be reported and enriched with important comments. This makes the work of the developers easier. Several developers can work together to complete a task[3].

Legacy Code (LC)
LC is a source code which is only patched but no longer supported. LC does not have to be necessarily outdated source code, even if it is often the case. Through the use of many patches, the code becomes more and more unmaintainable (Technopedia, 2017f).

Legacy Information Systems (LIS)
LIS are outdated computer systems that use outdated technologies such as outdated programming languages. LIS can not be modernized so easily and are therefore still used in companies (Technopedia, 2017g). In this master thesis we use the term LIS in the singular form as well as in the plural form.

Project Object Model (POM)
The POM file is an integral part of the Maven framework. Here dependency conditions are defined, and further project-specific adjustments are made [4].

[3]`https://www.atlassian.com/software/jira` (accessed 2017-06-17)
[4]`https://maven.apache.org/pom.html` (accessed 2017-05-09)

Remote Procedure Call (RPC)

RPC is an interprocess communication technology and is used in server client architectures. The client sends a request to the server. The server processes the request and returns a response. The client continues his work (Technopedia, 2017a).

Uniform Resource Locator (URL)

URL is an address where a particular resource can be found in the World Wide Web (Technopedia, 2017h).

Velocity

Velocity is a template language and supports the MVC (Model View Controller) pattern. It runs on a Java-based templated engine. With Velocity templates can be created and afterwards filled with data through Java objects. It separates Java code from web pages and is an alternative to JSP (Java Server Pages) and PHP (Apache, 2017).

1.2 Motivation

The main focus of this master thesis is the migration of the existing Jira add-on Timesheet to the current Jira version 7. The Timesheet add-on was developed by Adrian Schnedlitz (Schnedlitz, 2016) for version 6. This version is no longer compatible with the current version and so the add-on had to be adapted. Since our Jira server was updated, it was also necessary to update the add-on to the latest version. Otherwise, the add-on could no longer be used.

The timesheet add-on is used in the Catrobat project. The big advantage of this add-on is that it perfectly fits our needs. Users can make timesheet entries and link them with a suitable category. A category could be “pair programming”, “meeting” or “developing”. With these categories, coordinators can see how much time users spend on specified activities. If a category is never used by a user or team, the team coordinator can encourage the user/team to use them more actively. If the "pair programming" category is selected the user has the choice to pick up his team partner from the user list and add him to the timesheet entry. So it is clear who did pair programming together.

Furthermore, the data can be visualized through diagrams. Here an overview about one's performance in hours over the time is given. The user can see when he/she was more productive and when he/she did not spend a lot of time on the project. The visualization can also be applied to a team. In a second diagram the team performance is shown. Thus, the team coordinator can see how successfully his team performs.

In the administrator area, timesheet admins have a lot of preferences which they can adjust. They can add or delete new categories, assign team members to teams or decide when team members should be set automatically to inactive or inactive/offline. If a user is set to inactive or inactive/offline he receives an email that informs him that he was set to inactive because he did not make a timesheet entry for a longer period of time. The period of time can be specified by the timesheet administrator. If the user still remains inactive, he will receive more emails which prompt the user to become active again. This should help students to get motivated and move on with their work.

Finally, students are able to export their timesheet as CSV and JSON format and reimport them later again. If there is a loss of data one day, the student's timesheet data are safe. Students can also import their timesheet data from Google sheets. This is necessary because the students used the Google tables for their timesheets in the past and are now forced to use only our timesheets anymore.

To use our timesheet in next few years as well, it was necessary to update our Jira add-on version from 6 to 7 because the major version of the Jira platform also changed. Therefore, considerations were done to decide how to best deal with it. We decided to choose the migration strategy because it was the easiest way and moreover a lot of time could be saved. Otherwise the add-on would have had to be newly developed, which would destroy all the hard work that was already investigated by Schnedlitz.

In the course of the adjustment, several improvements were made. The timesheet add-on is now compatible with the most used browsers and can be also used on smartphones or tablets. The security was improved, so that the manipulation of data can be excluded. The administration area is safe now, so that unauthorized persons have no access anymore. The performance could be increased, so that the timesheet loads significantly faster.

All in all it is a useful add-on which fulfills all our requirements. Students have a powerful timesheet add-on which helps them to have a better overview about their personal performance, but also over the whole team performance. Moreover, maybe it can help to increase student's motivation.

Finally, it should be pointed out that the original add-on contained legacy code which was successfully removed during the migration process. Otherwise, the add-on could no longer be used on modern Jira platforms.

2 Legacy Information Systems

2.1 Definition and General Idea

It is not possible to give a universally valid answer to what exactly a legacy information system (LIS) is. Keith Bennet from University of Durham tried to specify it. In his opinion, legacy information systems (LIS) are defined as “large software systems that we don’t know how to cope with, but that are vital to our organization.” (K. Bennett, 1995). A LIS is typically a 10 or more years old system which runs on obsolete hardware. The software is mostly based on an old standard, so the integration with newer systems is often a huge problem (X. Li, 2010). It connects system files and interfaces as Figure 2.1 shows. The lack of documentation and old standards could lead to misunderstanding of the fundamental system, so it is often difficult for programmers to maintain the software (Bisbal, Lawless, Wu, and Grimson, 1999). Organizations have to weigh whether they change the existing old system to a newer one or to stay with the existing one (Tripathy and Naik, 2014a). There are many possibilities to deal with such systems which will be shortly mentioned. An overview of common migration strategies is given in (K. H. Bennett, Ramage, and Munro, 1999; Cimitile, Muller, and (eds.), 1997):

- Freeze: The system will be frozen because it will not be used anymore or it will be replaced by a newer system.
- Outsource: The system will be outsourced to another company because it is better to hand the software to a third party than continue to develop. One reason for this decision is that the other company could be more specialized in this field.
- Carry on maintenance: The Company shifts the task to a later time. All problems continue to exist.

- Discard and redevelop: The LIS will be scrapped, instead a new powerful and modern system will be developed. New technologies, hardware and software platforms, databases and tools will be used. The new system is up to date.
- Wrap: The LIS system will be wrapped with a new system because this system better fits the user's requirements. The old system still exists and keeps the logic of the system, but is enhanced by a new software layer which hides the complexity of the system. The new layer could be a new graphical interface.
- Migrate: The LIS will be moved to a new environment. The system has to be refactored, but the functionality will be the same. Migrating software can be a very time consuming and complex process.

The migration of a LIS is a very complex and difficult process. In the worst case, this can lead to the destruction of the whole system. The problems are in the understanding of the target system and how the migration process can be put into practice (Souiou and Bounour, 2013). There are several influencing factors such as decomposability, budget or technical and time constraints (Colosimo et al., 2009). To avoid such kind of situations, strategies have to be found to deal with these problems. Modernization strategies are one such approach.

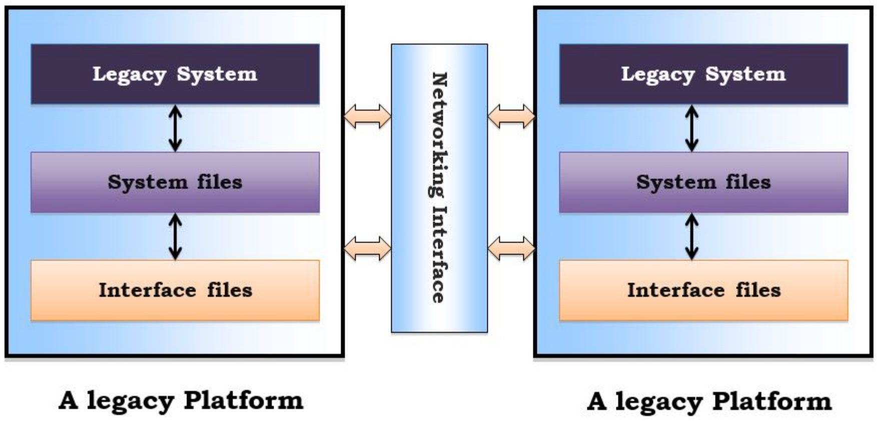


Figure 2.1: General Idea of a Legacy Platform (Makki, 2006)

2.2 Modernization Strategies

There are different definitions in literature about what exactly software modernization is. For some of them, redevelopment and migration are modernization techniques for others only migration is. When we use the term software modernization, we mean migration as well as wrapping or other reengineering techniques. Generally there exist three main categories. The most common modernization strategies are (Souiou and Bounour, 2013):

- Redevelopment (Comella-Dorda et al., 2000; Littlejohn, DelPrincipe, and Preston, 2000)
- Wrapping (Comella-Dorda et al., 2000)
- Migration (Zapata et al., 2015; Menychtas, Santzaridou, et al., 2013; Menychtas, Konstanteli, et al., 2014)

Each strategy has effects on the system. The effects range from slight changes like Wrapping to the completely new development which can lead to major changes and risks as shown in Figure 2.2. All three different strategies will now be discussed.

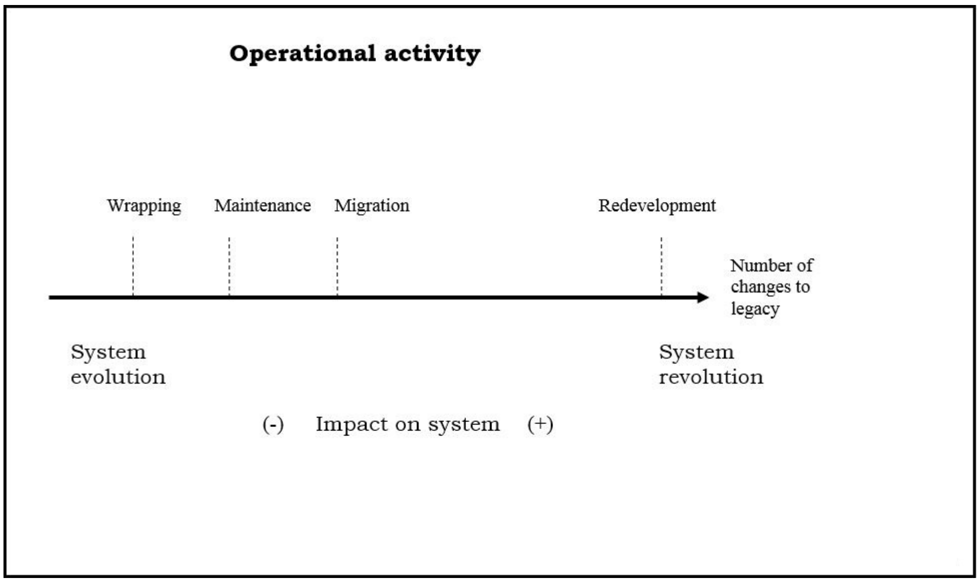


Figure 2.2: The impact on the system according to the chosen strategy (Bisbal, Lawless, Wu, and Grimson, 1999)

2.2.1 Redevelopment

When we speak about redevelopment we typically mean the reimplementation of an existing system. All the existing code is thrown away and reimplemented from scratch. Typically we use a new software or hardware platform for redevelopment because technology has changed and the old environment may not be used anymore. New tools, modern architectures and databases replace old ones (Bisbal, Lawless, Wu, and Grimson, 1999).

Before we can reimplement the old system, we have to analyze it's functions and requirements to derive the requirements the new system has to possess. Necessary documents are rare and make the process more difficult (X. Li, 2010). It should also be checked if the existing system can be migrated or wrapped because it is less expensive and minimizes the risk of failure (Bisbal, Lawless, Wu, and Grimson, 1999; Littlejohn, DelPrincipe, and Preston, 2000). Technology and business requirements can change very quickly over time and at the end the software does not fulfil the requirements any more. Therefore it could be very risky to reimplement the whole system from scratch (Bisbal, Lawless, Wu, and Grimson, 1999).

If developers decide to develop a new system, there are many obstacles that have to be overcome. K. Littlejohn mentions some key factors which influence the progress of redevelopment (Littlejohn, DelPrincipe, and Preston, 2000):

- Outdated Methods: LIS are typically based on hierarchical platforms which were originally developed for uniprocessor platforms. In contrast to modern object-oriented languages, LIS are often based on functional implementations with centralized data pools.
- Lack of Modern Integrated Analysis Capability: In LIS environments, there is the possibility of symbol and dependency tracking. Generally the problem is that there are several tools which do not work together properly and it is often cumbersome to use them. Therefore, the recognition of design changes is tedious and difficult which has an impact on the motivation when an existing system structure should be updated.

- Platform Coupling: Device drivers are software components which control the communication between the kernel of the operating system and the hardware. They convert requests from high-level software to low-lowel software in form of a series of input/output (I/O) operations. A device driver defines an interface to the kernel and allows high-level applications to communicate with low-level devices. Components like OS, bus protocols, system management and network programming are different depending on the underlying system. When for instance source code from the device driver should be migrated to another target platform, this is not an easy venture. Platform specifications like I/O offsets, interrupt connections or bus clock must be considered (Chen et al., 2014). So there is often no separation between the application and the underlying operating system. Legacy code often contains source code which controls the peripherals. Such as timing constraints there are many other constraints that have to be customized to the underlying system. Thus performance as well as the computation accuracy can be improved (Littlejohn, DelPrincipe, and Preston, 2000).
- Structural "Degradation": Through the maintenance lifetime of a LIS many changes are made to the original software design. That includes extensions and local modifications which could lead to obscure and complex system architecture. The original design idea is not valid anymore.
- Resource Constraints: Resources like memory, computation and I/O operations are limited. As a result many structural and performance challenges have to be handled.
- Commercial off-the-shelf (COTS) Exploitation: In the past, the defence industry developed their own hardware and software systems. The self-developed components had a durability of 15 to 20 years (Kent and Dewey, 2016) but the development was very expensive. Nowadays, standardized products which are available on the market are preferred because costs can be reduced. The disadvantage is that commercial products typically have a refresh cycle of 2 years, which is too short for the military purpose (Kent and Dewey, 2016).
- Available Knowledge and Experience Base: When developers decide to reimplement a system, a lot of information about the previous system is needed. Often the information is not available anymore

because the experts are no longer available. The lack of documentation could make a redevelopment difficult and expensive.

Sneed (2001) described further obstacles. He sketched a scenario where information was already collected. The information was made available to the programmers and could have been very helpful when it came to a new development of an existing system. Sneed found out that it depended on the programmers how carefully the newly generated information was used. In his study he came to the conclusion that programmers who were familiar with the existing system, tended to reject the extracted information, while developers who had no background knowledge were more open to it. He explain his insights with the fact that system-experienced programmers reconstruct information out of their heads instead of listening to someone else. Furthermore, he underlines his statement with three points:

- "first, they tend to overestimate their own knowledge of the business logic,
- secondly, they confuse the current solution with the business problem to be solved,
- thirdly, they believe that their existing solution is optimal whereas in reality it is usually dependent on the environment they have implemented it in."(Sneed, 2001)

His final conclusion is that developers who maintained the LIS over years are not the most suitable candidates for the reengineering process because they are closed for new information. He recommends reimplementing the LIS in a new environment with new programmers who are not familiar with the code base. They are more willing to use knowledge extraction tools and regenerate new knowledge from existing sources (Sneed, 2001).

2.2.2 Wrapping

Wrapping describes the process of providing a LIS with an additional wrapping layer. The LIS can commuincate with the wrapper over sockets, Remote Procedure Calls (RPC) or Application Programming Interfaces (API). The wrapper typically provides an object-based interface across which an exisiting system can communicate with the unchanged LIS and hides interface screens, APIs, communications adapters, files, and databases. The advantage is that LIS are embedded into new environments and can be reused as a new component of the system. They can be extended and are longer in use. (Goyla, 2000).

There exist different kinds of wrappers (Tripathy and Naik, 2014b):

- Database wrappers
- System service wrappers
- Application wrappers
- Function wrappers

From the view of the LIS, there is no difference between the new wrapped system and the original state, but from the perspective of the end user the new system looks modern and more user-friendly than the old one (Comella-Dorda et al., 2000). To illustrate the concept, an example will be used. The screen scraping is a wrapping technique where an existing system is wrapped with a new GUI. In earlier versions the output was shown on text-based screens like terminals but is now migrated to a new graphical interface. Although the functionality of that system is the same, the user experience is more pleasant. However, the problems of the LIS still remain and aggravate the general issues. It is more difficult to maintain such a system (Comella-Dorda et al., 2000). Developers can reuse existing well performing code to reduce the costs. The maintenance costs will increase because the issues addressed to LIS still exist and the problems that occured by maintaining a LIS will be only postponed to the future (Bisbal, Lawless, Wu, and Grimson, 1999).

Screen scraping is only one of the modernization techniques. For the sake of completeness, an overview of all common techniques will be given. The following Table 2.1 was retrieved from Comella-Dorda et al. (Comella-Dorda et al., 2000):

	Artifact Modernized	Target	Strengths	Weaknesses
Screen Scraping	Text-based user interface	Graphical or web-based user interface	• Cost • Time to market • Internet support	• Flexibility • Limited impact on maintainability
Database Gateway	Proprietary access protocol	Standard access protocol	• Cost • Tool support	• Limited impact on maintainability
XML Integration	Proprietary access protocol	XML server	• Flexibility • Tool support (future) • B2B	• Tool support (present) • Evolving technology
CGI (Common Gateway Interface)	Mainframe Data or TM services	HTML pages	• Cost • Internet support	• Flexibility • Applicability
OO Wrapping	Any Enterprise Resource	OO Model	• Flexibility • Easier understanding	• Cost
Component Wrapping	Any Enterprise Resource	Component Model	• Flexibility • Integrated services • Incremen. replacement	• Cost

Table 2.1: Comparison of Modernization Techniques; 1:1 cited from Comella-Dorda et al. (Comella-Dorda et al., 2000)

2.2.3 Migration

In this Chapter we will discuss the issues and challenges of the migration of LIS and known strategies for minimizing these problems.

When we talk about migration and migration processes, we typically mean that we want to move an existing mostly old system to a new modern platform. The existing logic of the software will still be in use. The software itself is adapted to the new target system. The correctness and behavior of the system will be checked through testing. Figure 2.3 shows an overview of migration issues. Some of them are more researched than others. For instance, target system development, testing and database model selection are very well researched as they are very often used in software projects. On the other hand, target system database population and cut-over are less explored, since they are less often used in practice (Bisbal, Lawless, Wu, and Grimson, 1999).

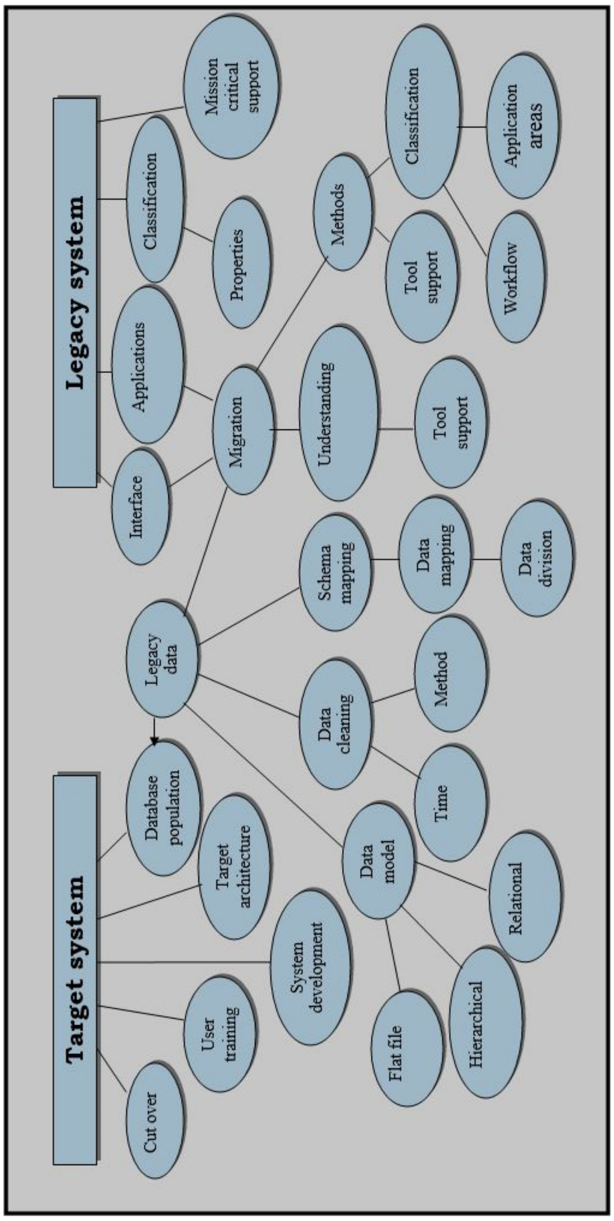


Figure 2.3: Different types of problems when it comes to migrating systems (Bisbal, Lawless, Wu, and Grimson, 1999)

LIS migration issues and challenges

Sarrab, Elbasir, and Elgamel (2013) mentioned that migration problems can be basically divided into two classes. On the one hand, into technical problems and on the other hand into non-technical problems.

The technical issues are (Sarrab, Elbasir, and Elgamel, 2013):

- Performance
- Technical Infrastructure
- Usability
- Integrity
- Support Availability
- Security
- Information Flow Control
- Data Migration
- Flexibility and Ease of Use
- Management and Maintenance of Open Source Software (OSS)

The non-technical issues are (Sarrab, Elbasir, and Elgamel, 2013):

- Organisational Culture
- Human Factors – Staff Skills
- Legal Issues

Migration Methods

Cold Turkey/Big Bang

The Cold Turkey strategy (Brodie and Stonebraker, 1995) is also called Big Bang strategy. In this case the LIS is thrown away and is replaced by a newly developed system. It has the advantage that it uses modern techniques such as new platforms, tools, databases and software architectures (Tripathy and Naik, 2014b). Additionally new programming languages, so-called 4GLs, replace older ones (Brodie, Stonebraker, and Ai, 1993). But redevelopment also increases the risk of failure because of the complexity of the old legacy system (Tripathy and Naik, 2014b). M. Brodie and M. Stonebraker discussed several reasons which can be regarded as barriers to a new development (Brodie, Stonebraker, and Ai, 1993):

- A better system must be promised
- Business conditions never stand still
- Specifications rarely exist
- Undocumented dependencies frequently exist
- LIS can be too big to cut-over
- Management of large projects is hard
- Lateness is seldom tolerated
- Large projects tend to bloat

Finally, it should be said that the Cold Turkey / Big Bang involves high risk and should be avoided in large organisations. Michael Brodie mentions many points, but one should be emphasized (Brodie, Stonebraker, and Ai, 1993):

> "A better system must be promised"

Management will not invest money into a new system, only to have less maintenance costs in the future. The system must provide additional requirements that fulfil the management expectations. But this increases the complexity and the risk of a total failure (Brodie, Stonebraker, and Ai, 1993). It should also be mentioned that it can be very risky to assume that the new system will run very smoothly from the beginning, because this is often not the case (Bisbal, Lawless, Wu, and Grimson, 1999).

Chicken Little

In contrast to the Cold Turkey approach, the Chicken Little method migrates the old LIS in small steps to the target system (Brodie, Stonebraker, and Ai, 1993). The LIS is built on the new target system with modern tools and technology (Bianchi et al., 2003). The advantage is that if one migration steps fails, only that step has to be repeated and not all of the previous ones. The entire migration process will not be affected by this.

According to Michael Brodie and Michael Stonebraker the Chicken Little strategy consists of the following 11 steps (Brodie and Stonebraker, 1995):

1. Analyse the LIS.
2. Decompose the LIS structure.
3. Design the target interface.
4. Design the target application.
5. Design the target database.
6. Install the target environment.
7. Create and install necessary gateways.
8. Migrate the legacy database.
9. Migrate the legacy applications.
10. Migrate the legacy interfaces.
11. Cut over to the target information system.

While the risk of failure is minimized, the complexity is increased. Moreover, if you want to migrate LIS which consists of unstructured and monolithic program code, it could be very hard to slice it into small pieces (Bisbal, Lawless, Wu, and Grimson, 1999). Compared with the Cold Turkey strategy it is more likely that the migration process will be a success (Brodie, Stonebraker, and Ai, 1993).

Database First / Forward Migration Method

The Database First method (Bateman and Murphy, 1994) is also called forward migration method (Brodie, Stonebraker, and Ai, 1993). First the database of the existing LIS will be migrated to the new target system such as a Database Management System (DBMS), and then the applications and user interfaces are integrated. For this approach a forward gateway must be created, otherwise the existing LIS applications will not be able to access the new database as shown in Figure 2.4. While the LIS is able to communicate with the new target system, applications and interfaces have to be adapted or redeveloped (Tripathy and Naik, 2014b; Bisbal, Lawless, Wu, Grimson, et al., 1997). The forward gateway serves as a translator between the legacy system and the new system. It's task is to redirect the calls from the LIS to the target system and to translate the answer of the new database system (DBMS) back to the old one. But the gateway can do much more than just translate calls. It can also be used to enhance or correct old LIS applications. For instance, a new data type could be introduced which will be later used in the target application (Brodie, Stonebraker, and Ai, 1993).

Ultimately when the migration has been completed, the gateway can be removed because it will not be needed anymore (Tripathy and Naik, 2014b). The prerequisite for using this approach is that the underlying architecture has to be decomposable. This means that the application modules must be independent from each other, so they only communicate with the database service (Brodie, Stonebraker, and Ai, 1993).

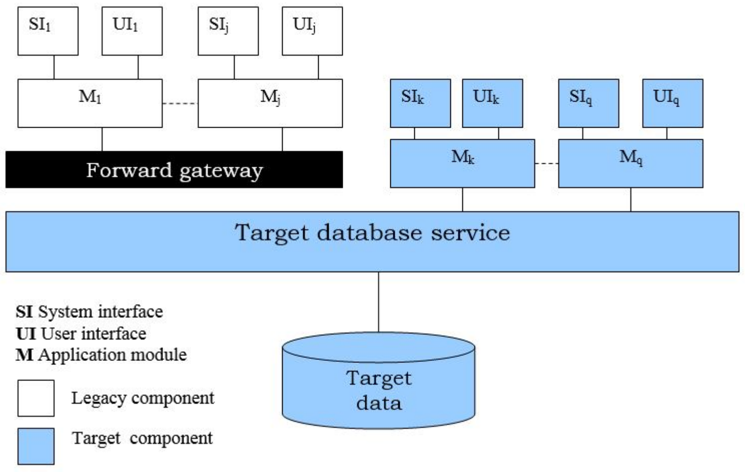


Figure 2.4: Database First Strategy (Tripathy and Naik, 2014b)

Database Last / Reverse Migration Method

The Database Last method (Bateman and Murphy, 1994) is also called reverse migration method (Brodie, Stonebraker, and Ai, 1993). In contrast to the previous strategy, the database is migrated during the last step as seen in Figure 2.5. The remaining database after the migration process will be the original LIS database, so only the application modules of the new system will be integrated into the old legacy database service. Therefore a reverse interface is needed which translates the target application calls into the LIS database. There are two problems with this approach. First, the target applications may use modern calls which are supported by a new relational database system but would not be supported by the old one. Examples are triggers, integrity or defining constraints. The second problem is that these translations could lead to performance losses, so applications have to be adapted (Tripathy and Naik, 2014b).

For this approach it is also necessary that the underlying architecture is decomposable (Brodie, Stonebraker, and Ai, 1993).

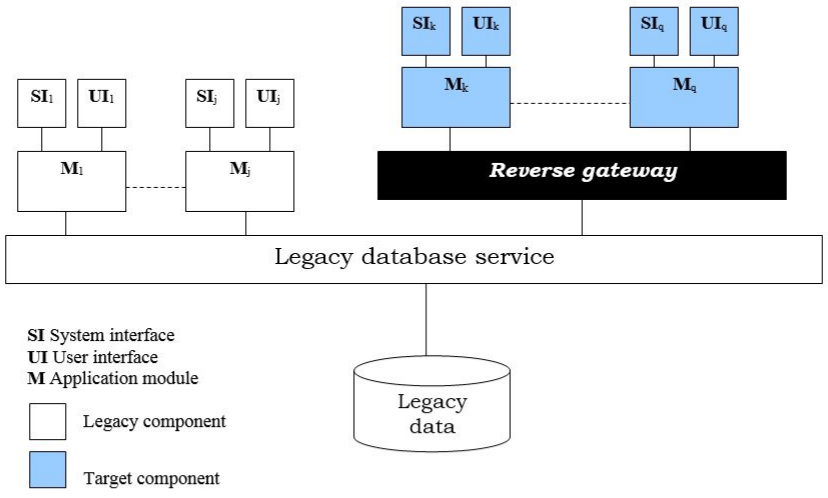


Figure 2.5: Database Last Strategy (Tripathy and Naik, 2014b)

Butterfly

For the Butterfly approach (Wu, Lawless, Bisbal, Richardson, et al., 1997; Wu, Lawless, Bisbal, Grimson, et al., 1997) it is not necessary that the target system communicates with the LIS system during the migration process. So there is no need to synchronize these two systems as no interoperation between those systems exists (Wu, Lawless, Bisbal, Grimson, et al., 1997). Furthermore also no gateways are needed and the overall complexity can be reduced (Bisbal, Lawless, Wu, and Grimson, 1999; Tripathy and Naik, 2014b). It is important to mention that the data migration and the target system are carried out as completely independent processes (Bianchi et al., 2003).

Before we describe the Butterfly methodology, some points should be mentioned (Wu, Lawless, Bisbal, Grimson, et al., 1997):

1. The target system is not running during the LIS migration.
2. The LIS is always running during the migration process.
3. There exists no cooperation between the LIS and the target system, so live data will never be stored at the same time on both systems. This is the difference to the previously mentioned approaches such as Big-bang, Database First and Database Last where data are translated through a gateway and always available on each system.
4. The butterfly methodology uses a legacy data migration engine which has the advantage that the LIS is available and needs to be shut down only for a short time of period.

Figure 2.6 shows the butterfly methodology. At the beginning of the migration process several data stores, so-called TempStore (TS), are created and the LIS data stock is set to read-only, also called frozen. Through the data access allocator (DAA) the database accesses are redirected to the temporary data stores TS. First TS_1 is created. New entries are saved into TS_1, existing entries will be updated. Modified data will be recovered from TS_1 (Bisbal, Lawless, Wu, and Grimson, 1999; Erdle, 2005). So the old database is only used for reading while changes are stored in the temporary data stores (Bianchi et al., 2003).

In the next step, the database of the LIS is migrated to the target system. For this task a new term should be introduced, the Chrysalizer. It is a component which migrates the old database to the new target system database. During this process all new data manipulations are not stored in the old LIS data stock anymore, instead they are redirect to TS_1. Therefore during migration the temporary data stores keep all the manipulated data.

Once the migration of the LIS is finished, the first temporary data store TS_1 has also to be migrated to the target system. For this reason a new data store TS_2 is created to save all new modifications because TS_1 is locked now and cannot be written any more. During the migration of TS_1 through the Chrysalizer all new modifications are written in TS_2.

Each time a data store is migrated to the target system, the current TS_n is frozen and a new TS_{n+1} is created. This process continues until the time for the migration of the last TS_n is so small that the remaining migration can be treated very quickly.

Finally the LIS is shut down (frozen), the remaining TS_n is migrated and the new target system is running. The data consistency is equal to the LIS and the migration process has finished.

The big advantage of this approach is that if there is a problem during the migration process, the whole process could be stopped. The data only have to be copied back from the data storages $TS_1 ... TS_n$ to the original database (Bisbal, Lawless, Wu, and Grimson, 1999; Erdle, 2005).

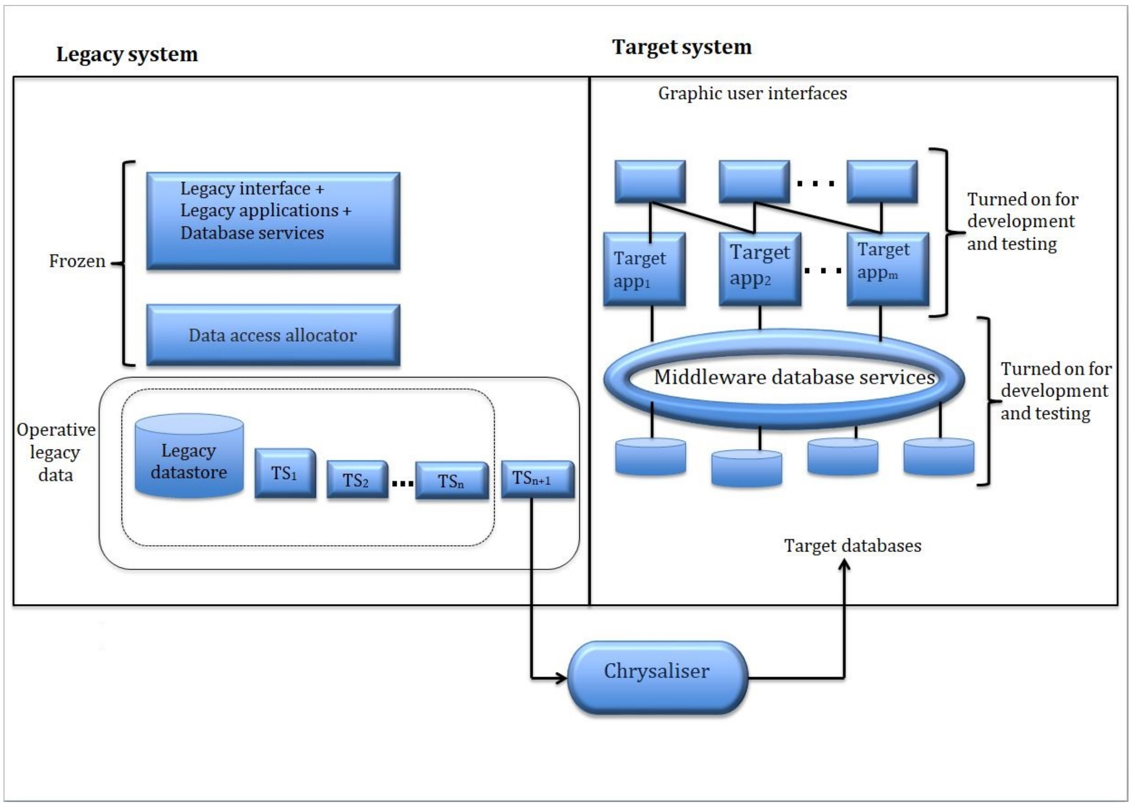


Figure 2.6: The butterfly methodology. The LIS database is migrated through the Chrysaliser in several steps to the target system (Bisbal, Lawless, Wu, and Grimson, 1999).

3 Dealing with LIS in the business world

"48% of employees are wasting at least 3 hours a day by working with inefficient systems

On average more than half of IT budgets are spent maintaining existing IT systems

Inefficient systems and a poor user experience impact staff morale and citizen satisfaction" (KCOM, 2017)

LIS cause costs. It does not matter if you simply work with the existing system or modernize it. Updating an existing system costs a lot of money, but not updating a LIS can lead to major challenges for the future that could lead to even more costs. Therefore, companies need to think wisely about which way is better for them (KCOM, 2017).

Legacy systems are widespread in the industry. The question is what is the true cost of LIS? Joe Stangarone tries to give an answer to that question and explains 7 points that he thinks cause costs (Stangaroneifi20, 2017):

- Maintenance costs
- Talent costs
- Support costs
- Integration costs
- Compliance costs
- Lost opportunity costs
- Agility costs

3.1 Maintenance Costs

Over time, it is difficult to maintain LIS. Ongoing changes make the existing system more complex. If you change one part of the system, another one could easily break. Therefore developers spend more and more time with maintenance rather than developing new features. In case of a problem, it becomes even more problematic. When using modern technologies it is easy to search for a solution on the internet. But if systems are outdated, it is much more difficult to find an adequate solution which in turn will waste a lot of time. Also, troubleshooting can only be reached with a high effort. Summarized, maintenance is inefficient and wastes a lot of time (Stangaroneifi20, 2017).

3.2 Talent Costs

As mentioned earlier, LIS often use old programming languages and technologies. In the finance industry mainframe computers are still in use. Many programs were written in COBOL and have never been updated. Nowadays it is very difficult to migrate these programs to new platforms because programmers are needed who can extract the business logic. Most developers who originally wrote these programs are now retired and usually documentation is not available (Sneed, 2001), (Paulson, 2001). So new talents have to be found who can deal with the old source code, but they are rare. It is hard to find developers who are skilled in those old programming languages and if they could be found, the salary would be very high as there are less and less qualified people. In the end, the costs will increase which could be avoided if new modern systems were used (Stangaroneifi20, 2017).

3.3 Mainframes are still popular

> *"In a survey, 45 percent of IT managers expected Cobol to be used at the existing level for the next 10 years."*(Paulson, 2001).

Mainframes in combination with Cobol programs are still in use. In a Cutter Consortium survey they found out that more than 50 percent of the respondents still use mainframe machines. 25 percent of the respondents said that more than half of their critical software are running on mainframes. Ronald J. Kizior, Loyola University assistant professor, says that there is a demand on Cobol skills. Head hunters seek for programmers with 2 or 3 years programming experience but it is very hard to find mainframe programmers (Paulson, 2001). Typically the average age of programmers who have skills in writing Cobol programs is 45 years old and the number is continuously decreasing. Younger people prefer more modern programming languages such as C++, Java or Visual Basic. From their point of view, learning Cobol is like learning Latin – it is a dead language and nobody wants to used it but is still needed (Paulson, 2001).

The situation is not different in the financial industry. It is estimated that 75 per cent of the IT budget of banks and insurance companies is used for the maintenance of software systems (Arnold and Braithwaite, 2017; Gangadharan et al., 2013). Therefore, it is important for companies to develop solutions to deal with the maintenance costs (Crotty and Horrocks, 2016). The advantages of mainframes are that they are able to process a huge number of complex transactions at high speed and that they are reliable. But the cost to maintain mainframes is much higher. It is estimated that mainframes cost 90 per cent more than x86 servers. Furthermore, new modern servers have a better price vs performance ratio, a higher interoperability and cost less money. Expanding to the cloud and other open platforms is much easier because new technologies come with modern integration tools that support popular platforms. Cloud mobility, automation and big data are new banking services which companies in the finance sector have to deal with. Additionally, new EU legislation forces monetary financial institutes to adapt the services to meet new laws and agreements (Ismail, 2017).

In the year 2000, another study from Kizior and Donald Carr was published. The authors said that 90 percent of the information system managers want to keep their existing Cobol programs and 45 percent want to keep Cobol systems at the existing level for the next 10 years (Carr and Kizior, 2000).

Cost is the main influencing factor which decides if a legacy system is going to be redesigned or continued to maintain. If the risk for reengineering is too high, the current system will be used. But there are many reasons to argue against the maintenance of LIS. LIS are less stable over time, changes to the code require more time. It is difficult to add new functionality because programmers often do not understand the impacts of their changes. The existing system is getting more and more complex over time (Schneidewind and Ebert, 1998).

But what is the optimal software for life time? A newly developed system is error-prone and normally has many bugs which have to be eliminated. New tests have to be written and extensively tested against the new implementation. The existing architecture and source code has to be reengineered. Costs can exceed the original development time and can increase very quickly (Karthikeyan and Nandhini, 2016). But there is also the risk that the new system will fail and will not deliver the required services.

3.4 Facts and Figures of Maintenance Cost in Software Projects

In a study, Boehm examined the ratio of maintenance and development costs in companies. He conducted a survey with 487 companies. His findings are shown in the following two diagrams. Boehm claimed that programmers spend 43% of their time with development and the remaining 57% with maintenance of programs. For economic purposes some parts were booked as maintenance, which would be better placed under development. Table 3.1 shows how much time programmers spend with individual tasks. The tasks that are highlighted are actually part of the development and should be excluded. So if you add the 19% to the 43% you get to 62% which seems more realistic. Figure 3.1 shows the software development and maintenance costs in large business organizations when software is in use for 10 years. Figure 3.2 shows the maintenance and development costs in 487 business organizations (UECD, 2010; Boehm, 1981).

Development	43 per cent
Maintenance	
a) Emergency program fixes	6 per cent
b) Routine debugging	4 per cent
c) Accommodate changes to input data, files	8 per cent
d) Accommodate changes to hardware, operating systems	**3 per cent**
e) Enhancements for users:	
New reports	**8 per cent**
Added data for existing reports	**6 per cent**
Other	7 per cent
f) Improve documentation	3 per cent
g) Improve code efficiency	**2 per cent**
h) Other	8 per cent
Other	2 per cent

Table 3.1: Amount of time developers and system analysts need for the individual items related to maintenance in the US (Adapted from (UECD, 2010)).

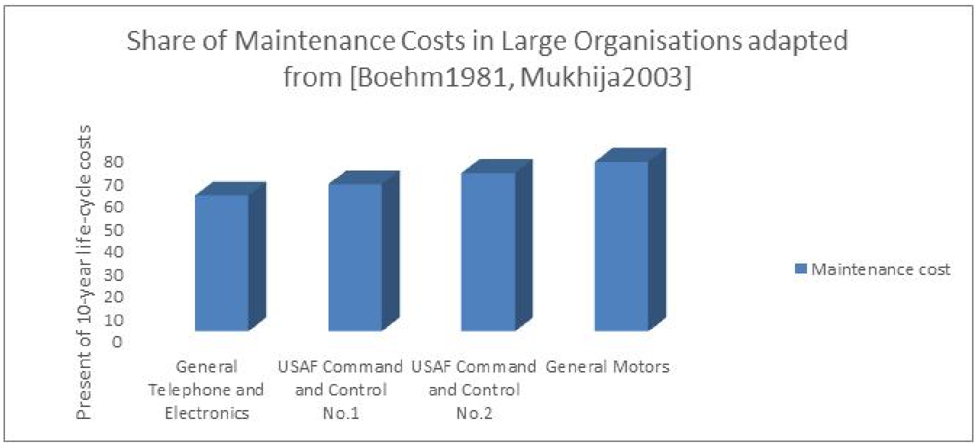


Figure 3.1: Share of Maintenance Costs in Large Organisations adapted from (Boehm, 1981; Mukhija, 2003)

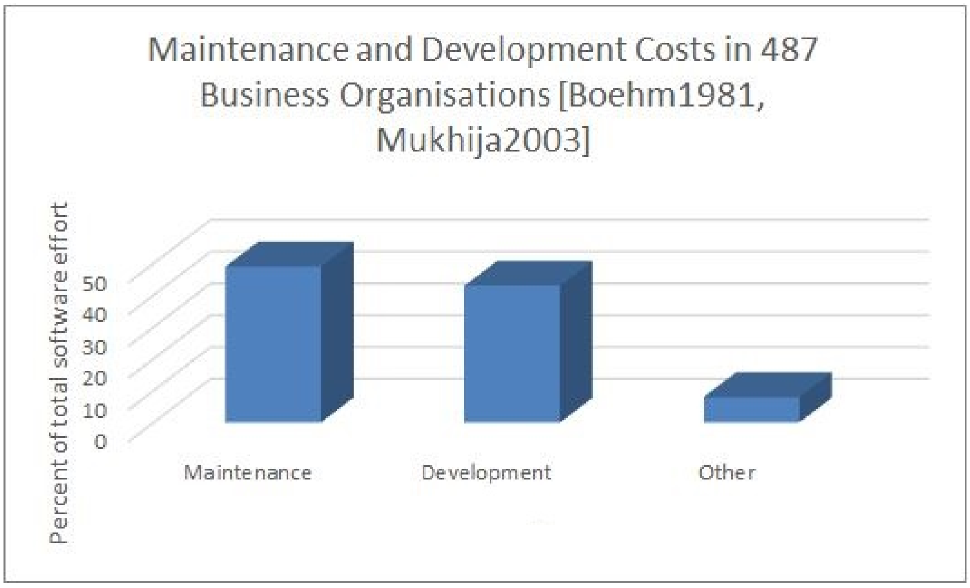


Figure 3.2: Maintenance and Development Costs in 487 Business Organisations (Boehm, 1981; Mukhija, 2003)

4 Strategies on how to successfully deal with LIS

4.1 Technical Debt Management

Technical debt (TD) is a metaphor for bad code practise and was introduced by Ward Cunningham in 1992. He said that the delivery of unfinished code may be great for the customer, but would be dangerous in the long run. Programs become unmanageable and inflexible (Cunningham, 1992). Furthermore he said:

> *"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt."* (Cunningham, 1992)

TD describes situations in software development where a workaround or a shortcut is used to solve a specific problem. Such a shortening can be accompanied by the advantage of fast releases to fulfil business deadlines. At the same time, the coding structure becomes more and more complex and opaque. A problem which primarily software developers have to fight. As a result, the efforts to maintain the code increases. In other words, TD is increasing while software quality decreases steadily. (Yli-Huumo, Maglyas, and Smolander, 2016).

According to Allman (2012) TD has many similarities to financial debt:

- Repayment
- Interest
- High cost (in some cases)

Therefore it is important for companies to develop strategies to deal with TD. At this point the technical term technical debt management (TDM) is introduced. TDM was introduced to manage, prevent, measure and reduce TD. TDM consists of processes, techniques and tools. The aim is to reduce TD. TDM is hard to establish in companies because it is often difficult to estimate how much TD a system has and what impacts it will have in the future (Yli-Huumo, Maglyas, and Smolander, 2016).

According to Power (2013) there are 7 challenges with TDM:

- Agreeing what Technical Debt is
- Quantifying Technical Debt
- Visualizing Technical Debt
- Tracking Technical Debt Over Time
- Impact of Neglecting Technical Debt Over Multiple Releases
- Identiyfing Technical Debt as a Root Cause of Defects
- Understanding the Cost of Delay

TDM can be divided into several activities (Z. Li, Avgeriou, and Liang, 2015):

- Identification
- Measurement
- Prioritization
- Prevention
- Monitoring
- Repayment
- Representation/documentation
- Communication

Originally TD was limited to the code level of a software project. Terms like Code Smell (Fowler et al., 1999) were introduced to show how bad software decisions affect code quality and architecture. Since then, the term has evolved and is now associated with a number of other stages in the software development lifecycle. Today, the concept is no longer related only to the code level, but also to (Yli-Huumo, Maglyas, and Smolander, 2016):

- Requirements
- Design
- Architectural
- Process
- Test
- Documentation
- People debt

It should be pointed out that in the single steps, shortcuts and workarounds can also result in TD. So TD is a term not only referring to the code level of a software project (Yli-Huumo, Maglyas, and Smolander, 2016).

TD can be classified into two groups (Yli-Huumo, Maglyas, and Smolander, 2016):

- Intentional TD: This includes factors such as code complexity and business deadlines.
- Unintentional TD: This includes factors such as the lack of competence, the need for an upgrade or customer or market specific causes.

A study of Martini, Besker, and Bosch (2016) investigated the impact of TD on companies. 226 people from 15 companies were interviewed. The result was that the development time spent on TD was about 25 percent. The benefits of reducing TD were clearly noticeable in the next release. New features were easier to implement as there were no side effects. In addition, the architecture had been simplified (Martini, Besker, and Bosch, 2016). But there were also resistances when TDM was introduced. Particularly managers criticized the model and often were not ready to provide the required budget. Conviction was necessary why TDM is important. But Conviction was also necessary in teams. While enthusiasm prevailed in some teams, other teams had to be guided why TDM was really important. Thus, TDM is not always well accepted by managers and developers. A TD backlog was created to make it clear that they always have to think about it, otherwise the concept cannot be successful in the future (Martini, Besker, and Bosch, 2016).

4.2 Code Smell Avoidance

The term code smell was introduced for the first time by Fowler (Fowler et al., 1999). Code smell is an indicator of how good or how bad the code quality is. Bad code leads to increased maintenance efforts and should be avoided. Code smell can be eliminated by refactoring (Fontana, Ferme, and Spinelli, 2012). According to Carver et al. (2017) the most bad code smells are:

- Duplicate Code
- Cyclomatic Complexity
- Too Many Methods
- Excessive Imports
- God Class

Duplicated Code is one of the most common code smells besides Data Class and God Class (Fontana, Ferme, and Spinelli, 2012). Even in the development of our add-on, we found out that Duplicated Code occurrs often. For lack of time only small parts could be refactored. God Class was also a topic that could be reduced by refactoring, but could not be completely resolved due to lack of time.

4.3 Design Pattern

Design patterns are solutions for recurring problems. They can help in situations where a design decision has to be made and provide some necessary information to prevent bad coding styles. In addition, they help to make existing designs more flexible, understandable and performable. The risk of errors caused by a bad design can be minimized (Eilebrecht and Starke, 2010a).

Design patterns are based on principles that have proven their worth in the past. General heuristics can also be applied in many situations. Principles and heuristics form a unity that can be helpful when a decision has to be made if a design pattern should be chosen or not (Eilebrecht and Starke, 2010b).

4.4 Software Design Principles

Heuristics give programmers instructions how optimize the code. Many design patterns are based on heuristics. These can be divided into the following categories (Eilebrecht and Starke, 2010b):

- Design of classes and objects
- Inheritance and delegation
- Distribution
- Concurrency

For example, overly powerful classes (so-called God classes) should be avoided. Another good advice is that the superclass should not be supposed to know anything about their subclasses (Eilebrecht and Starke, 2010b).

A couple of design principles will be discussed now (Eilebrecht and Starke, 2010b):

- Open Close Principle: Classes should be open for extensions but closed for changes. New functions have to be extended by new classes, instead of modifying existing ones. Existing classes should only be adapted if they are improved or when bugs are fixed.
- Once and Once Only: Redundancy of code pieces should be avoided (i.e. duplicated code)
- Single Responsibility Principle: A class should have exactly one task and only a strictly bounded responsibility. Each class fulfills exactly one purpose. The same applies to methods. One method fulfills exactly one task. For a new task, new classes and methods have to be defined.
- Liskov's Substitution Principle: Super classes should only contain methods that fit for all subclasses. Subclasses which inherit methods that they do not need and therefore remain empty should be avoided.
- Composite Reuse Principle: Classes should be decoupled from each other and composition should be preferred over inheritance. The advantage is that the behavior can be changed during runtime which leads to a more stable and flexible code. Composition is often used in design patterns and has advantages compared to inheritance (Eric Freeman and Elisabeth Freeman, 2006).

Only a short list of principles has been presented here. The subject area is far more comprehensive. For further information, the book *Head First Design Patterns* by O'Reilly[1] can be recommended. On the basis of many practical examples it is shown how from dirty code, clear and well-structured code emerges. The learning effect is very high. If you follow these rules, you can easily avoid smell code. The MVC (Model View Control) pattern is also used in our add-on. Therefore it will be briefly mentioned and explained in more detail with a practical example.

[1]`http://shop.oreilly.com/product/9780596007126.do` (accessed 2017-06-19)

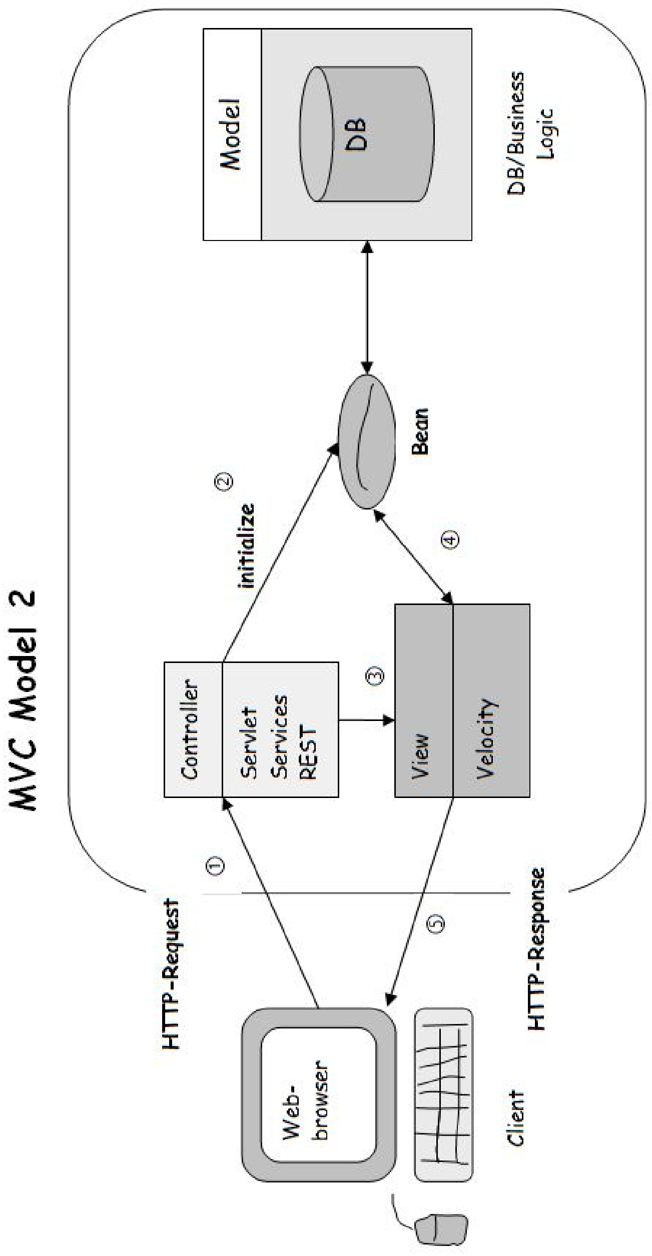


Figure 4.1: MVC pattern implemented on a practical example

Figure 4.1 shows the MVC model 2 pattern as it is also used in our application. According to Eric Freeman and Elisabeth Freeman (2006) the MVC pattern can be divided into the following steps:

1. **The client sends a HTTP request to the server**
   The user sends an HTTP request to the server by using the web browser. Typically form data such as username and password are sent. The server accepts the incoming request and processes it.
2. **The servlet is the controller**
   The servlet or REST interface accepts the request. It serves as controller. Because existing data often needs to be accessed, a query is made to the model to retrieve previously saved data. Often a database is used as a model. The result of the query is bundled in a JavaBean.
3. **The controller sends the data to the view**
   Velocity presents the view. It is a template language. First, it fills the template with data and then generates an HTML document. It is very similar to Java Server Pages (JSP), but has the great advantage that the code can be reused. The view is only responsible for presenting the model. The data are transmitted through the JavaBean (point number 4 in the chart) (Eric Freeman and Elisabeth Freeman, 2006).
4. **The result is returned to the client**
   The page generated by the view is sent back to the client and can be displayed there.

4.5 Google Closure Templates

With Google Closure Templates aka Soy templates can be created which can be used on client side as well as on server side. Atlassian supports Soy templates and our Timesheet add-on also uses these templates. This enables us to reuse HTML code. The security aspect was also very important. Closure templates effectively prevent Cross-site scripting (XSS) due to auto escaping. Angle brackets were replaced by &lt; and &gt; so the code is no longer interpreted as tags and the risk of harmful attacks can be reduced. Closure Templates also checks if the code has already been escaped and prevents double-escaping (Google, 2017).

4.6 Active Objects

Active Objects (AO) is a add-on for Atlassian which enables developers to access data in an easier and faster way. It is an additional layer based on object relational mapping (Atlassian, 2017a). Through getter and setter methods objects can be saved persistently and previously saved data can be retrieved from the database. SQL statements are no longer required, which also minimizes the risk of SQL injections.

5 Feature Specification

5.1 Overview of the Existing Implementation

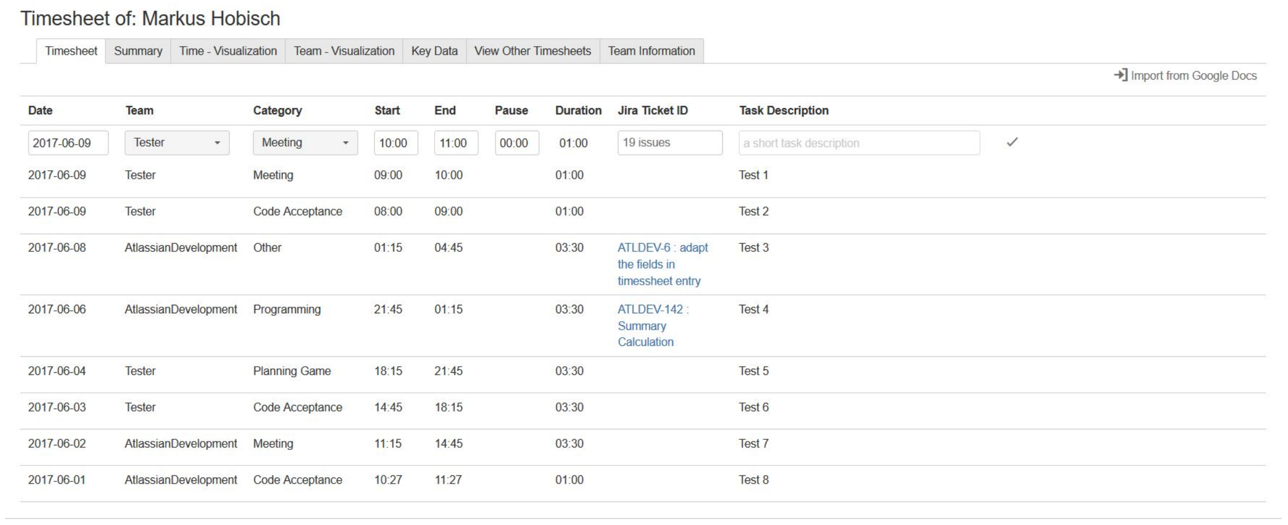
The Timesheet add-on was developed by Adrian Schnedlitz in 2016. It is a Jira add-on and it uses the Atlassian plugin software developer kit (Jira SDK) (Atlassian, 2017c) for development. The Jira version was 6.4 at that time. Atlassian typically supports major versions only for 2 years (Atlassian, 2017b) and Atlassian announced that the version 6.4 will only be supported until March 17th 2017 (Burwinklepple, 2017). Therefore the Jira instance had to be updated to Jira 7, and also the Jira add-on had to be updated to version 7. Regrettably, the Jira SDK is neither compatible downwards nor upwards. So many adaptions of the program code were made to keep it running. Otherwise it could not be used anymore in our project.

The add-on is a time tracking tool which replaces the Google timesheets. Students can import their existing timesheets from a CSV file or export them as CSV. All timesheets can be exported at once. Only single timesheets can be imported from an existing document (Schnedlitz, 2016). The tool will be used in the Catrobat organization. Currently more than 100 students are involved in the project. These students have many obligations such as exams, exercises and some of them are working at companies. Depending on their schedules, they invest more or less time into Catrobat. For many students it is harder to continue with the work after a long break. In this case, our add-on could help them to stay focused on their work. If they are inactive for a longer period of time, they will be informed by email notification. The project coordinator will be informed as well, who can then help students to continue with the project.

The administrative tasks can be reduced drastically because many steps can be automated. For instance, in the past an administrator needed 5 to 10 minutes to create or delete a new user. Now the same task can be done in only a few seconds. Furthermore, formatting errors such as point numbers or date errors can be avoided because many fields are filled in automatically and only valid inputs are accepted.

In the past, wrong date formats led to broken Google links and so coordinators and timesheet administrators were not able to get an overview about timesheet related data like remaining hours of a user. Sometimes when a user formatted the date and saved the timesheet, his changes where not recognized by Google and the user had to repeat his changes again and again until it accepted the new data format. Formulas did not work if the data were not formatted correctly.

The new add-on will also help to increase transparency and visibility. Supervisors get a great tool to analyze the whole project performance and how the agile development approach is realized. They also have more opportunities to analyse project specific behavior and get more detailed information about what is going on. Team visualization data help project coordinators as well as team members to get a better understanding of the team performance. Timesheet administrators have more configuration options and can manage the whole project in a better way.



Timesheet of: Markus Hobisch

Date	Team	Category	Start	End	Pause	Duration	Jira Ticket ID	Task Description	
2017-06-09	Tester	Meeting	10:00	11:00	00:00	01:00	19 issues	a short task description	✓
2017-06-09	Tester	Meeting	09:00	10:00		01:00		Test 1	
2017-06-09	Tester	Code Acceptance	08:00	09:00		01:00		Test 2	
2017-06-08	AtlassianDevelopment	Other	01:15	04:45		03:30	ATLDEV-6 : adapt the fields in timessheet entry	Test 3	
2017-06-06	AtlassianDevelopment	Programming	21:45	01:15		03:30	ATLDEV-142 : Summary Calculation	Test 4	
2017-06-04	Tester	Planning Game	18:15	21:45		03:30		Test 5	
2017-06-03	Tester	Code Acceptance	14:45	18:15		03:30		Test 6	
2017-06-02	AtlassianDevelopment	Meeting	11:15	14:45		03:30		Test 7	
2017-06-01	AtlassianDevelopment	Code Acceptance	10:27	11:27		01:00		Test 8	

Figure 5.1: Timesheet Form

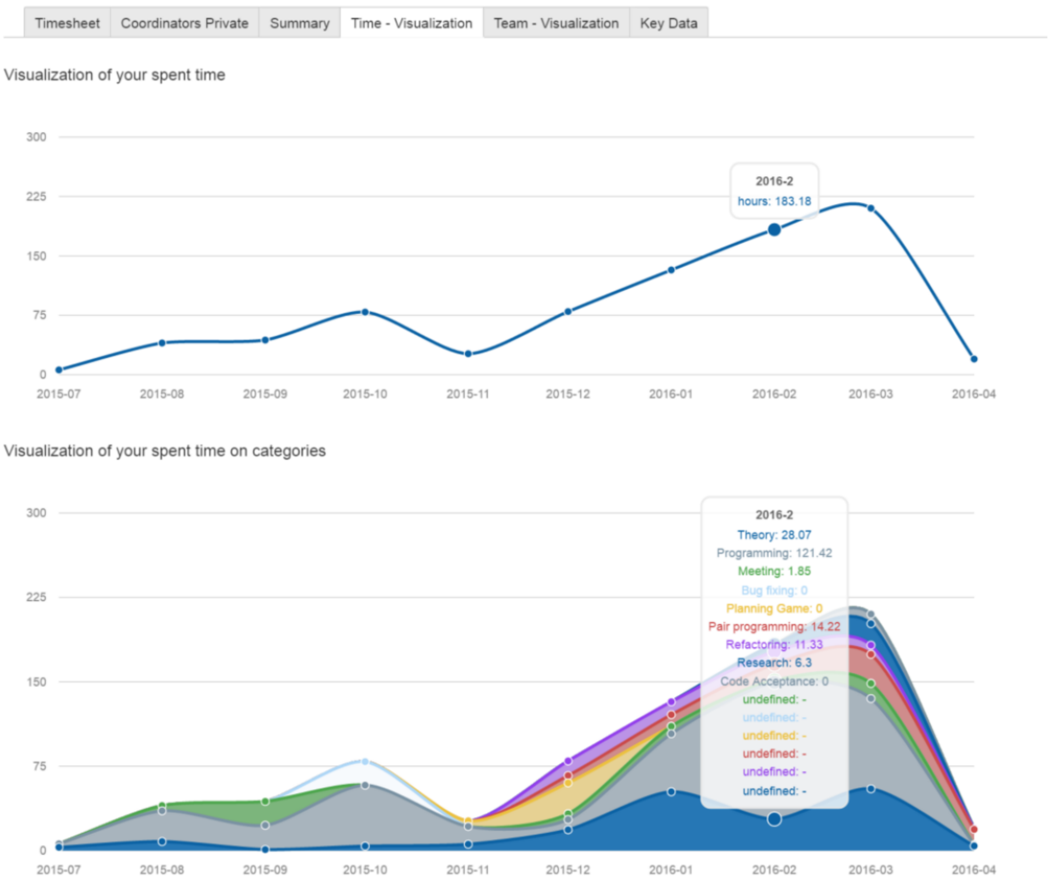
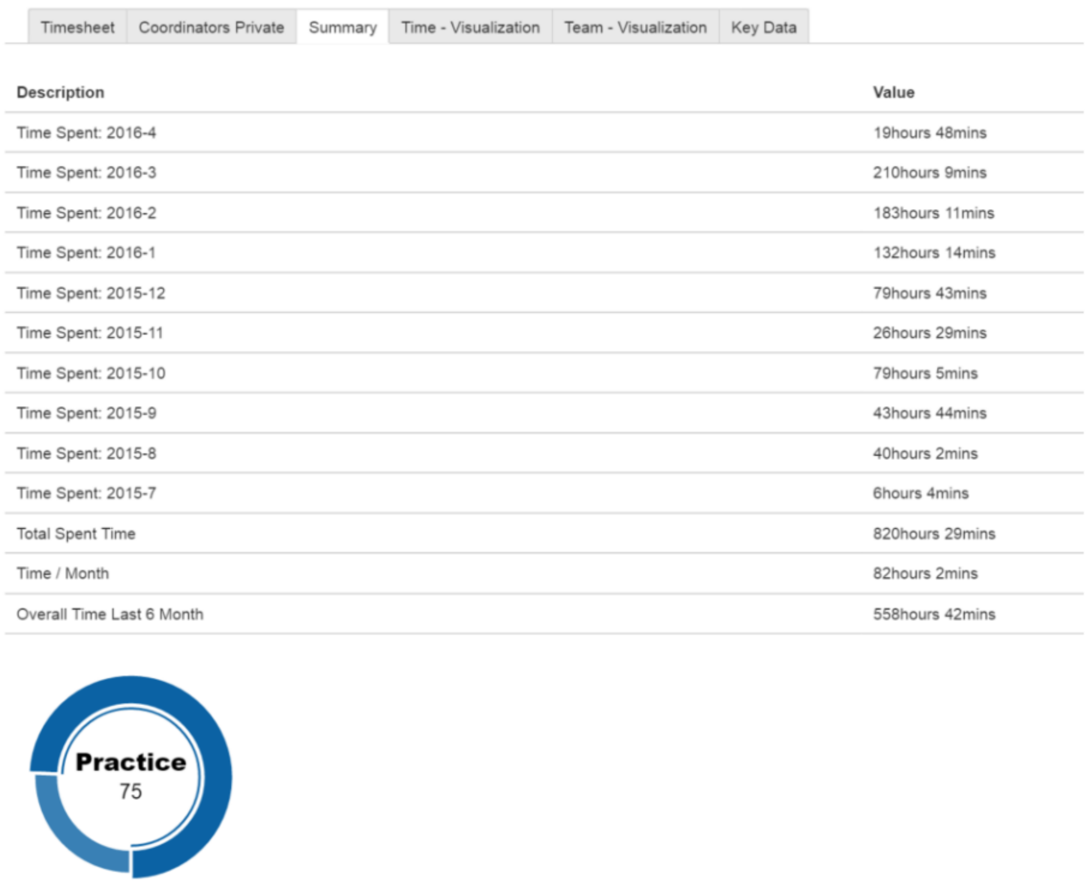


Figure 5.2: Visualization of the timesheet data



Description	Value
Time Spent: 2016-4	19hours 48mins
Time Spent: 2016-3	210hours 9mins
Time Spent: 2016-2	183hours 11mins
Time Spent: 2016-1	132hours 14mins
Time Spent: 2015-12	79hours 43mins
Time Spent: 2015-11	26hours 29mins
Time Spent: 2015-10	79hours 5mins
Time Spent: 2015-9	43hours 44mins
Time Spent: 2015-8	40hours 2mins
Time Spent: 2015-7	6hours 4mins
Total Spent Time	820hours 29mins
Time / Month	82hours 2mins
Overall Time Last 6 Month	558hours 42mins

Figure 5.3: Summary of hours worked

5.2 The Catrobat Organization

Catrobat makes it possible for students to work in teams and increase their experiences in programming but also in the acquisition of social skills. For most students it is the first time to work in a Free and Open Source Software (FOSS) team. The code is open source and available on GitHub[1] . The Catrobat organization is divided into several teams. More than 20 subprojects exist, some of them are:

- ScratchToCatrobat
- Catrobat marketing team
- CatroidArduino, Raspberry Pi
- CatrobatDrone
- Catroweb
- HTML5
- iOS
- Jenkins
- Catroid
- Musicdroid
- Paintroid
- Usability
- Design

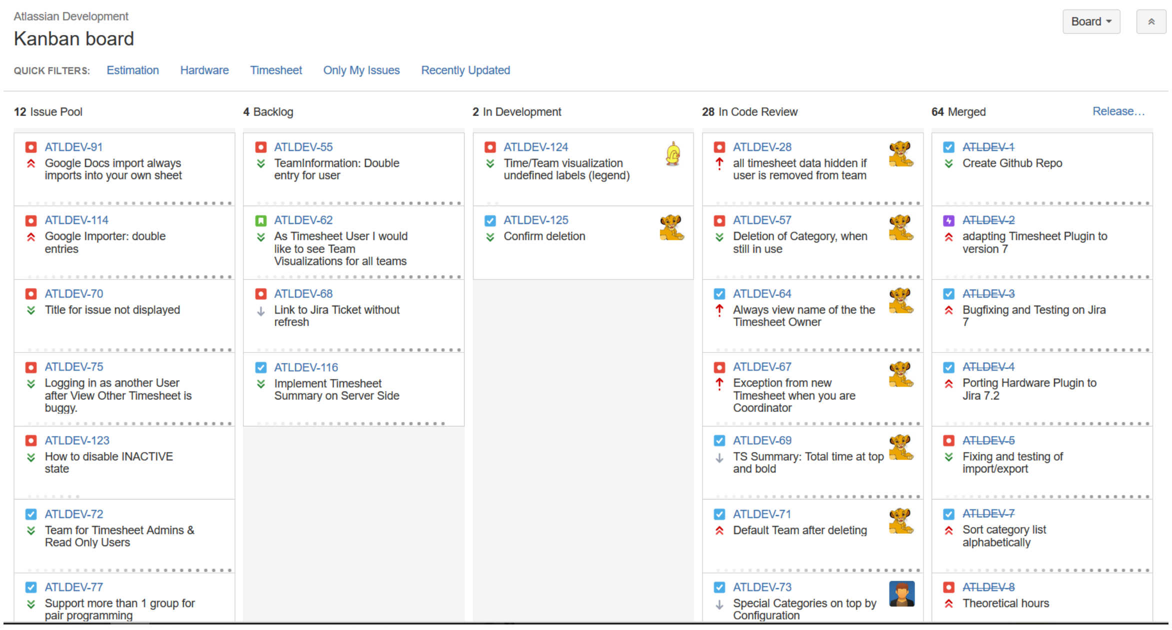
The main focus is on maintainability, usability and design (Wolfgang Slany, 2014). There are many subteams which work on several extensions for Pocket Code. Extreme Programming (XP) is used as agile development methodology. Teams usually consist of 3 to 5 people, but there are also larger teams up to 10 people. Not all of them are developers. Some are designers or take care of the usability. Each team has a coordinator who manages the team and is in connection with other coordinators. New ideas are developed jointly by the coordinators. Every two weeks, meetings take place where every coordinator has to join and shares his/her project-related updates with others. Proposals for solutions are discussed with Professor Slany, and if necessary, realized.

[1]https://github.com/ (accessed 2017-06-17)

5.2.1 Catrobat - Jira

Catrobat team members use the Jira Agile boards for collaborating. Figure 5.4 shows the Kanban board used by Catrobat team members. There, our students can:

- create an issue
- assign an issue to a third person or themselves
- get an overview of which tickets are currently being processed
- create the next major release


Atlassian Development

Kanban board

Board

QUICK FILTERS: Estimation Hardware Timesheet Only My Issues Recently Updated

12 Issue Pool

ATLDEV-91 Google Docs import always imports into your own sheet

ATLDEV-114 Google Importer: double entries

ATLDEV-70 Title for issue not displayed

ATLDEV-75 Logging in as another User after View Other Timesheet is buggy.

ATLDEV-123 How to disable INACTIVE state

ATLDEV-72 Team for Timesheet Admins & Read Only Users

ATLDEV-77 Support more than 1 group for pair programming

4 Backlog

ATLDEV-55 TeamInformation: Double entry for user

ATLDEV-62 As Timesheet User I would like to see Team Visualizations for all teams

ATLDEV-68 Link to Jira Ticket without refresh

ATLDEV-116 Implement Timesheet Summary on Server Side

2 In Development

ATLDEV-124 Time/Team visualization undefined labels (legend)

ATLDEV-125 Confirm deletion

28 In Code Review

ATLDEV-28 all timesheet data hidden if user is removed from team

ATLDEV-57 Deletion of Category, when still in use

ATLDEV-64 Always view name of the the Timesheet Owner

ATLDEV-67 Exception from new Timesheet when you are Coordinator

ATLDEV-69 TS Summary: Total time at top and bold

ATLDEV-71 Default Team after deleting

ATLDEV-73 Special Categories on top by Configuration

64 Merged

Release...

~~ATLDEV-1~~ Create Github Repo

~~ATLDEV-2~~ adapting Timesheet Plugin to version 7

~~ATLDEV-3~~ Bugfixing and Testing on Jira 7

~~ATLDEV-4~~ Porting Hardware Plugin to Jira 7.2

~~ATLDEV-5~~ Fixing and testing of import/export

~~ATLDEV-7~~ Sort category list alphabetically

~~ATLDEV-8~~ Theoretical hours

Figure 5.4: Kanban board used by Catrobat team members

5.2.2 Catrobat - Confluence

Atlassian Confluence is used in Catrobat as knowlegde management platform. Students can:

- communicate with each other
- write blog entries
- share information with others
- find others and get in contact with them
- create a survey or
- use the team calendar

Here students get a great overview about the whole organization and find a lot of useful information about what they may need.

5.2.3 Pocket Code

The project was launched by Wolfgang Slany in 2010. He is the head of the Institute for Software Technology (IST) at Graz University of Technology. The main motivation is that teenagers can easily build and share their mobile applications (Wolfgang Slany, 2014).

Catrobat formally known as Catroid (W. Slany, 2012; Wolfgang Slany, 2014) is a visual programming language developed for teenagers between 13 and 18 years (Koitz and Wolfgang Slany, 2014).

Pocket Code is a smartphone and tablet application where users can create their own programs such as animations or games and distribute them on the Pocket Code platform (Catrobat, 2017). There everyone can download any project, modify it and upload it again (W. Slany, 2012). These programs are created by composition. This means that many simple blocks are connected to create a complex program. These colored Lego-style blocks can be combined with conditions, loops and other statements. The blocks are divided into different categories depending on the behavior of the brick (Koitz and Wolfgang Slany, 2014). Figure 5.5 shows the home screen on the left side and the script view on the right side.

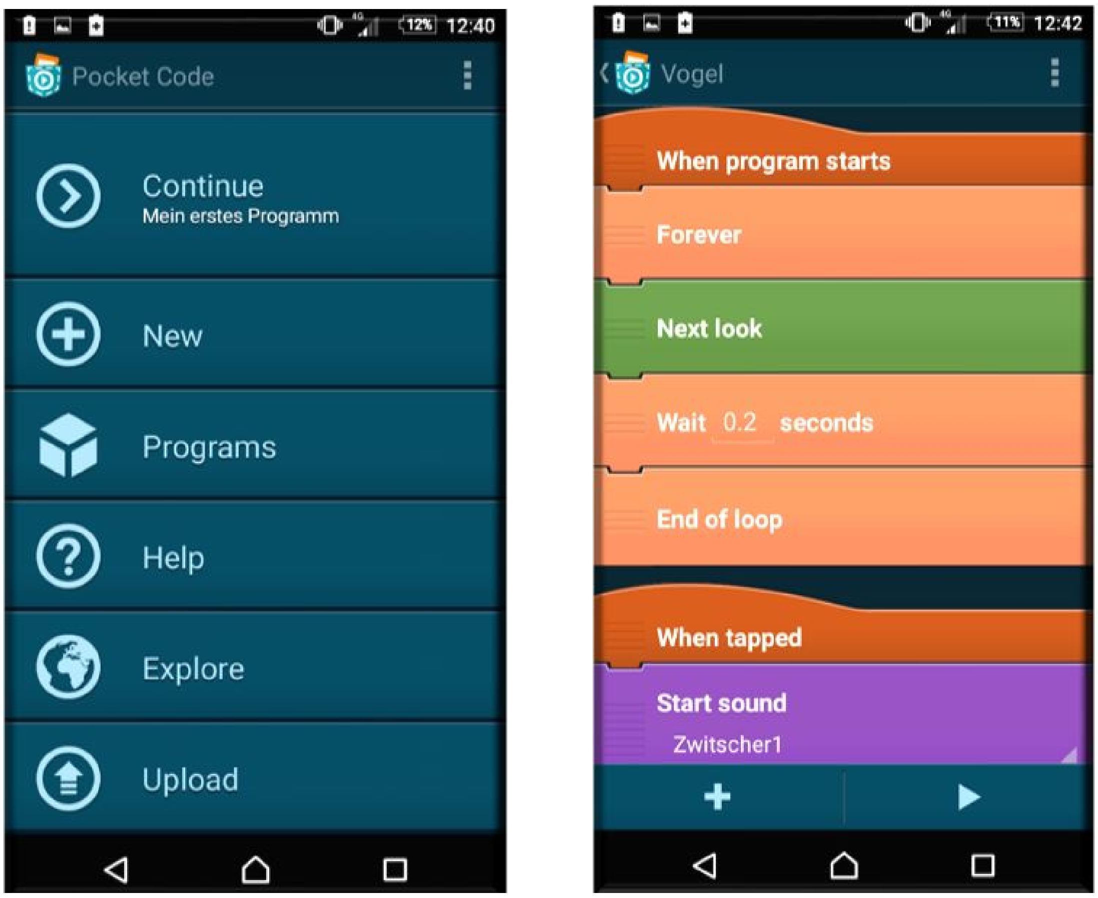


Figure 5.5: Android Application Pocket Code

Currently the software is only available for Android[2], but other platforms are under development (W. Slany, 2012). The original name was Catroid but has changed to Pocket Code and can be found on the android play store (Store, 2017). Pocket Code has been inspired by Scratch which was developed by MIT (W. Slany, 2012; Scratch, 2017).

With Pocket Code it is possible to control external hardware via Bluetooth [3] or WiFi[4] . The supported devices are Parrot AR.Drone 1 and 2[5] , and the Parrot Minidrones[6] , Lego Mindstorms robots[7] , Phiro robots [8] and the Bluetooth Arduino boards[9].

5.3 Requirements

Primarily it was important that the existing Timesheet add-on can be used in our project. Due to the fact that Jira has stopped the support for version 6 it was clear that the add-on had to be adapted to the latest version. During the migration process the focus was placed on several points:

- The add-on should be stable and reliable
- It should be compatible with all modern browsers and operating systems
- It must be safe against security vulnerabilities such as SQL injections or Cross-Site-Scripting (XSS)
- Students are not allowed to see any private data of others
- It should be well-tested so it can be used in productive mode

[2]https://www.android.com/ (accessed 2017-04-28)

[3]https://www.bluetooth.com/what-is-bluetooth-technology/discover-bluetooth (accessed 2017-04-07)

[4]http://standards.ieee.org/about/get/802/802.11.html (accessed 2017-04-14)

[5]https://www.parrot.com/de/ (accessed 2017-04-07)

[6]https://www.parrot.com/de/minidroneS (accessed 2017-04-07)

[7]https://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com (accessed 2017-04-07)

[8]http://www.robotixedu.com/phiro.aspx?AspxAutoDetectCookieSupport=1 (accessed 2017-04-07)

[9]https://www.arduino.cc/en/Main/ArduinoBoardBT?from=Main.ArduinoBoardBluetooth (accessed 2017-04-07)

- All critical bugs must be removed
- Comfort functionalities: some extra features extends the add-on
- Refactoring: source code should be simplified
- Speed should be increased, especially in case of many timesheet entries

The requirements are divided into several groups:

- General requirements
- Graphic requirements
- Security requirements
- Compatibility requirements
- Migration requirements
- Performance requirements
- Bug fixing

5.3.1 General Requirements

General requirements include extending the existing implementation with new features. One feature was to define and implement the behavior when a timesheet is automatically set to inactive and when it should be set (active) again. Figure 5.6 shows the different states a user can have. A user can set himself to inactive or inactive & offline and later active again. The difference between these two states is that in inactive mode he is willing to answer emails or chats. In active & offline mode he/she is not available, when he is on holiday for example. If a user does not make a new timesheet entry for a while, the system will automatically set the user to inactive (auto inactive). The user receives an email notification. If the user makes an entry, he is set to active again. Otherwise the inactive phase will continue and extend the inactive end date. After x iterations the timesheet will be disabled. The timesheet admin and the user are notified. Only the timesheet admin is allowed to reset the timesheet's state to active again.

The intention is that a conversation between the team member and the administrator must take place if the user is disabled. Otherwise the user may be removed from the team, which would not be very satisfying and should be avoided. Occurred problems should be discussed and strategies developed which help students to reach their goals. If a problem is related to other team members or to the team coordinator, the coordinator is involved in the discussion. The aim is to develop solutions which satisfy all members. Furthermore, the administrator gets a better overview of the project and can see which teams need more support.

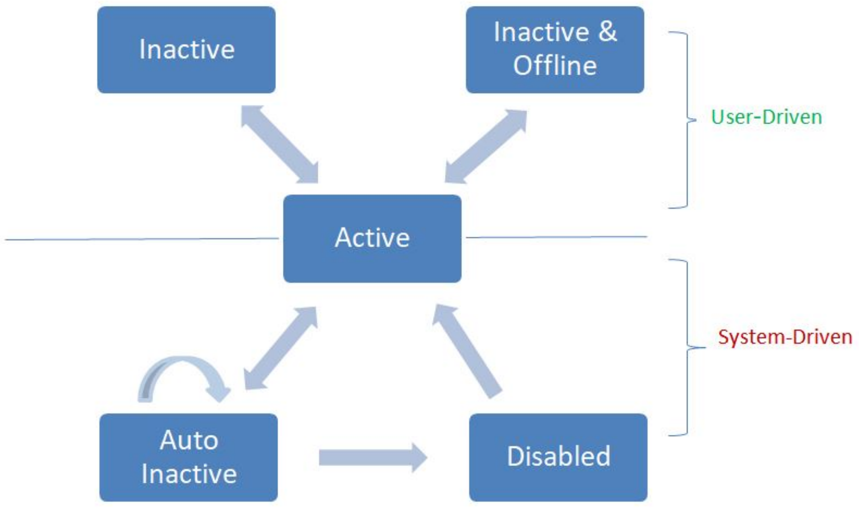


Figure 5.6: State Diagram

Requirement	Description
Timesheet	If the user has no timesheet, the link in the main menu should be hidden.
User Information	The table should be sorted by name.
Timesheet	A timesheet administrator or coordinator should be able to see another timesheet.
Timesheet	If the F5 key is pressed, the last timesheet should be shown.
Timesheet	If a user adds an inactive entry, it should be possible to extend this inactive duration.
Timesheet	If a user state is updated to inactive because he/she was not working for two weeks, the inactive duration is extended (x-times) before an email notification is sent.
Timesheet	There must be an inactive date selection limit.
Timesheet	If you are manually inactive, but added a new entry you should be asked if you want to remain inactive or set to active.
Timesheet	If you are automatically inactive, but you have added a new entry you will immediately be set to active again.
Timesheet	If you are automatically set to inactive, a new entry is to be created with a text: "I am inactive presumably up to xxx". Start and end dates are not necessary here.
Timesheet	If you are automatically set to inactive, the coordinator of the team will be notified by e-mail at each extension.
Timesheet	If you are automatically set to inactive, the coordinator of the team will be informed by e-mail at each round.
Timesheet	After a certain period the timesheet administrator will be notified by e-mail.

Table 5.1: General Requirements

Table 5.1 shows the general requirements list. These requirements define when a user or administrator should be informed about the status of a timesheet.

5.3.2 GUI Requirements

The original view from the Timesheet add-on should be adopted. For example, if the inactive category is selected only the columns Date, Category, Inactive Until and Task Description should be visible. The others like Start, End, Pause, Duration, Team, Partner (PP) and Jira Ticket ID should be hidden.

The pair programming column should only be visible if a pair programming (PP) category is selected. The Jira Ticket ID field does not exist and should be added as extra column. It shows only tickets which could be relevant for the user. Table 5.2 and Table 5.3 show the graphical requirements. Because there are so many requirements they are splitted up into two tables.

Requirement	Description
User Interface of Timesheet	In the standard behaviour only the columns Date, Start, End, Pause, Duration, Jira Ticket ID and Task Description should be visible. The team column should only be shown if the user is in more than one team.
User Interface of Timesheet	If a default category is selected the task description should be empty.
User Interface of Timesheet	If the inactive category is selected, the Inactive column should be visible. Start, End, Pause, Duration, Team, Partner (PP) and Jira Ticket ID should be hidden.
User Interface of Timesheet	If the Inactive & Offline category is selected, the Inactive & Offline column should be visible. Start, End, Pause, Duration, Team, Partner (PP) and Jira Ticket ID should be hidden.
User Interface of Timesheet	If a category has changed, the Task Description field should be emptied.
User Interface of Timesheet	If a column is hidden, its content should be deleted.
User Interface of Timesheet	If the Pair Programming category is selected, the Partner field should be visible. Otherwise it should be hidden.
User Interface of Timesheet	If the category which contains the big letters PP is selected, the Partner field should be visible. Otherwise it should be hidden.
User Interface of Timesheet	If a field is hidden, the input must be cleared.

Table 5.2: Graphical Requirements Part One

Requirement	Description
User Interface of Timesheet	If the Inactive & Offline entry date is older than today, an error message (red box) should be displayed.
User Interface of Timesheet	If the inactive entry date is more than 2 months ahead, an error message (red box) should be displayed.
User Interface of Timesheet	If the Inactive & Offline entry date is more than 2 months ahead, an error message (red box) should be displayed.
User Interface of Timesheet	A new field called Jira Ticket ID should be created.
User Interface of Timesheet	The Jira Ticket ID should only display tickets which could be relevant for the user.
User Interface of Timesheet	The Jira Tickets in the Jira Ticket ID field should be linked with the issue page in Jira.
User Interface of Timesheet	If the Task Description is empty, an error message (red box) should be shown.
User Interface of Timesheet	The name of the timesheet owner should be displayed.
User Interface of Timesheet	There should be no space under the table.

Table 5.3: Graphical Requirements Part Two

5.3.3 Security Requirements

In the testing phase we found critical security vulnerabilities that had to be closed immediately. All admin sites were still accessible if the user knew the exact Uniform Resource Locator (URL)[10]. Furthermore, a normal user was able to drop the whole database. Also the self-developed Representational State Transfer (REST)[11] calls were vulnerable because the server had never proved if the client had the permission to access the requested resource. Hence a normal user had full read access to all information. He could not only read data, he was also able to write and manipulate data. In the worst case he made himself administrator, deleted other timesheet data or had access to other timesheets. Table 5.4 shows a list of all requirements that were implemented.

[10]https://www.techopedia.com/definition/1352/uniform-resource-locator-url (accessed 2017-04-28)

[11]https://www.techopedia.com/definition/1312/representational-state-transfer-rest (accessed 2017-04-28)

Requirement	Description
Timesheet Security	The configuration area should not be accessible through the URL without permission.
Timesheet Security	The user information area should not be accessible through the URL without permission.
Timesheet Security	A normal user should not be able to access any REST API interface without permission.
Timesheet Security	A normal user should not be able to access any private information about a third person.
Timesheet Security	It should be checked if security vulnerabilities such as SQL injection, cross side scripting etc. are possible.
Timesheet Security	It should be checked if a user can access another timesheet without permission. Read-Only users and timesheet admins are allowed to see all timesheet data.
Timesheet Security	If a user is not allowed to see administrator areas he should be redirected to Jira login.
Timesheet Security	A non-privileged user should not be able to drop the whole database.
Timesheet Security	A user should not be able to modify another timesheet except the timesheet admin.

Table 5.4: Security Requirements

Requirement	Description
Timesheet Compatibility	The add-on should work in Firefox.
Timesheet Compatibility	The add-on should work on iOS devices.
Timesheet Compatibility	The add-on should work in Safari.
Timesheet Compatibility	The add-on should work on Linux.
Timesheet Compatibility	The add-on should work on MAC.
Timesheet Compatibility	The add-on should work on smartphone devices.

Table 5.5: Compatibility Requirements

5.3.4 Compatibility Requirements

As Adrian Schedlitz (Schnedlitz, 2016) mentioned in his master thesis the add-on was only tested with Google Chrome[12] and Opera[13] . It should be also compatible with the most popular browsers. In Firefox browser[14] there exists a problem with the sending of timesheet data. It does not work at the moment and should be fixed. The add-on is also not compatible with Apple's Safari[15] browser. Table 5.5 shows the compatibility requirements.

5.3.5 Migration Requirements

Timesheet was developed for Jira 6. As mentioned earlier only version number 7 is supported anymore. Therefore the add-on must be migrated to version 7. During the migration the code should be refactored and

[12]https://www.google.de/chrome/browser/desktop/ (accessed 2017-04-14)
[13]http://www.opera.com/de (accessed 2017-04-14)
[14]https://www.mozilla.org/de/ (accessed 2017-04-14)
[15]https://www.apple.com/safari/ (accessed 2017-04-14)

any unnecessary source code eliminated. Table 5.6 shows the migration requirements.

Requirement	Description
Timesheet Migration	The Jira SDK version should be changed from 6 to 7.
Timesheet Migration	The whole functionality should be migrated to the target system.
Timesheet Migration	Unsupported and deprecated classes and methods have to be exchanged.
Timesheet Migration	Refactoring as part of the migration process. Elimination of useless code.
Timesheet Migration	The java version should be changed from 7 to 8. 8 is current state.
Timesheet Migration	Update dependencies to meet current requirements.
Timesheet Migration	Define a strategy how the migration process can be smoothly implemented.
Timesheet Migration	Redefine test cases to guarantee the correctness of the software.

Table 5.6: Migration Requirements

5.3.6 Performance and Stability Requirements

The reaction time of the timesheet is slow and should be increased. Therefore the code has to be changed. Data should be loaded once from the server and cached locally. Doubled source code fragments should be refactored. Table 5.7 shows the performance and stability requirements.

Requirement	Description
Timesheet Performance	The speed of the timesheet should be increased.
Timesheet Performance	Cache data instead of reloading it twice from the server.
Timesheet Performance	Delete unnecessary and duplicated code fragments.
Timesheet Performance	Improve system stability through code analysing tools.

Table 5.7: Performance and Stability Requirements

5.4 Demarcation and Self-Implementation

Finally, we will briefly explain why we decided to develop a new add-on and did not rely on existing Jira solutions. I would like to mention some points which were already discussed by Schnedlitz in his master thesis (Schnedlitz, 2016).

Schnedlitz investigated already existing applications for their usefulness for our project. He introduced pros and cons lists to facilitate decision-making. After all, new development should only be considered if all other solutions did not satisfy our requirements.

Some of the existing solutions are not designed for course management and do not support categories for project assignments. Categories cannot be assigned to teams, which is necessary in our project. Administrators or coordinators cannot be added to teams or groups which is a must have criterion.

Others are not designed for education purposes, but rather for industry and are therefore not suitable. Third parties allow us to track the time only depending on Jira tickets. This is not suitable too because we want to realize the time recording, regardless of Jira tickets only by means of categories. Others are not free of charge or are not compatible with our Jira version.

For the reasons mentioned above, we started with self-implementation. Our tool has some advantages:

- All our requirements can be implemented and easily extended
- The data processing can be very well understood
- The code is available and can be viewed by any team member at any time
- No license fees or other costs incur at any time
- Developed and tested with the utmost care
- Running on the newest Jira version 7

6 Implementation

6.1 Migration of the Jira Timesheet Add-on

Several migration strategies have been analyzed in chapter 2 and weighed up against each other. The best method was chosen and the advantages and disadvantages should be discussed.

6.1.1 Analysis of the chosen strategy and comparison with other migration strategies

Initial Situation

The Jira SDK version of the Timesheet add-on was 6 and was no longer compatible to version 7. Therefore the add-on had to be adapted to the newest version. The internal database was switched from HSQLDB[1] to H2. Both are relational databases and written in Java. The advantage of the H2 database compared to others is that the structure is simpler and the processing speed is increased (H2database, 2017). The Timesheet add-on was written in Java 7. Many classes and methods which exist in Jira SDK 6 were removed in version 7. Some of them were already deprecated so it was clear that they would probably be removed in the next major version. But this is not always the case. As an example, the UserProjectHistoryManager should be mentioned. No deprecated methods were removed without any prior notice. So Atlassian is not always following their own policy (Atlassian, 2017d). However, all deprecated methods, interfaces, classes etc. are

[1] `http://hsqldb.org/` (accessed 2017-04-12)

visible online[2]. A short description explains which construct should be used instead. This is very helpful in many situations.

The add-on had not yet been released, so there were no timesheet data which had to be saved. This was a huge advantage because otherwise we would have had to think about problems which could arise with a different database scheme. Probably we would have had to create a migration task where we defined how the database scheme was changed. Therefore all steps would have to be recorded which would have caused a lot of extra work. The migration would have been more complex, since the data could not be easily imported again.

As migration strategy we decided to take the Chicken Little approach because it is easier compared to other approaches. In the following subchapter the pros and cons of this strategy will be discussed and also the decision making will be explained.

Pros and Cons of the Chicken Little approach

The Chicken Little methodology was selected as most appropriate strategy for our migration approach which was already descripted in section 2. Table 6.1 shows the pros and cons of this strategy.

6.1.2 Decision Making

After all strategies were weighed against each other and the advantages and disadvantages of the Chicken Little strategy were scrutinized, this strategy was chosen. This approach was selected because of the straightforwardness and simplicity of the strategy. The database first and database last approach assume that the LIS system uses a very old and obsolete database, and the new target system a modern DBMS. In our case we already used a relational database and the new one is still a relational database. Therefore no gateway

[2]https://docs.atlassian.com/jira/server/deprecated-list.html (accessed 2017-04-12)

layer is needed because the language is quite the same.

The butterfly approach is used in large systems, where it can take a long time until one migration step is finished. Therefore it makes sense to build several temporal storages and migrate one after another to the target system. For us this approach is very inappropriate, it is like to break a butterfly on a wheel.

The Cold Turkey approach is mostly too time-consuming to implement it. In fact this means that the whole add-on has to be discarded and replaced by a new implementation which can be full of bugs. In the opinion of the author of this thesis this methodology should only be chosen if all other strategies fail and there is no other way to reuse the whole code or just code fragments.

Pros	Cons
Reuse of existing code	Existing bugs are taken over to the target system.
The system is migrated in several steps. Therefore the risk of failure can be minimized.	The complexity is further increased because additional adjustments have to be made which cause many changes to the original design.
The migration process is simpler compared to other approaches like Database first or Butterfly.	The maintenance effort increases with each migration process.
The migration completes more quickly because no special preparation is needed. No gateways or layers must be developed and no extra storages are necessary.	Improvements such as modern programming language techniques, up to date database functions or improved SDK methods are not taken. Instead old code still remains.
It can be determined very early on whether a migration will be successful at all.	Compared to the redevelopment approach (Cold Turkey) of the system, it is more inefficient and slower but less error prone because functionalities were well tested in the past.
Core functionalities and must have requirements can be provided very quickly on the target system. Not system critical or user-relevant data and functions can be submitted later.	
Problems and difficulties from the first migration step can be improved during the next step. This allows more flexibility during the migration process.	
The migration process has no effects either on the current system or on the target system because all changes can be undone very quickly (delete database entries, restore old database).	

Table 6.1: Pros and Cons of the Chicken Little Approach

6.2 Realization & Implementation

6.2.1 Researching

It is always a great idea to start a search about the target environment. The most important question is what has changed in the new version? The Atlassian website[3] is always a good point to start. Here you can find all information about the upgrading process. Announcements are published regularly. There is also a forum where developers can seek advice. A list of all API changes can be found here[4].

6.2.2 Preparation

In the first step, a new stand-alone Jira add-on for version 7 was created. We used the online tutorial[5] for creation. Afterwards we started the add-on to see if everything worked fine. There should be no problems because no changes were made before.

6.2.3 Migration Process

We used the Chicken Little approach as migration strategy. First we created a new Jira add-on based on version 7. This is our target system and the existing code should be migrated here. In the opinion of the author it was the best way to deal with the problem because the new platform supports all the dependencies which are used in our add-on. Deprecated dependencies have been replaced by new ones. The new Maven structure guaranteed that the server would accept the add-on and thus would also start. Otherwise, we would have had to search manually, which dependencies have changed and which have to be added. This would have been very cumbersome and probably would have taken much more time.

[3]https://de.atlassian.com/ (accessed 2017-04-14)
[4]https://developer.atlassian.com/jiradev/latest-updates/preparing-for-jira-7-0/jira-7-0-api-changes (accessed 2017-04-14)
[5]https://developer.atlassian.com/docs/getting-started (accessed 2017-04-07)

Next, the migration process will be explained. First, we tried to migrate the add-on in one step, but it was too complex. So we decided to divide the program code into several pieces of code and migrate them one after another to the target system. First we migrated the core functionality, afterwards the extensions. The core functionality includes only important parts of the system without which the add-on could not start. This includes the database connection, the refactoring of deprecated methods and classes, as well as the timesheet. Extensions include the administrator area, Timesheets overview, and visualizations.

The migration process is divided into 4 steps:

1. Migration of a piece of code
2. Adaption of the maven structure in the POM file
3. Adaption of the test cases
4. Verification if server accepts the add-on and starts correctly
5. Repeat step 1 to 4 until all code slices are migrated

Migration of a piece of code

In this first step the code from the existing implementation is copied to the new platform. On the target system the dependencies may have changed, so it could happen that not all classes or methods are not available anymore and have to be replaced by newer ones. So refactoring and adaption is often necessary. Deprecated methods are often removed or new add-on constraints are added in the next major release version.

Adaption of the maven structure in the POM file

The POM[6] file is an XML (Extensible Markup Language)[7] representation of a Maven project and contains all relevant dependencies a project needs.

[6] `https://maven.apache.org/pom.html` (accessed 2017-05-09)
[7] `https://www.techopedia.com/definition/24387/extensible-markup-language-xml` (accessed 2017-05-09)

These dependencies have to be adapted to the new Jira version. It might happen that not all of the dependecies are compatible with the newest Jira version.

Adaption of the test cases

Each time the coding structure has been changed, the test cases must also be adapted. If new code is added, new tests have to be written. If code is removed tests can be deleted. The goal of this step is to ensure that all test cases work.

Verification if server accepts the add-on and starts correctly

With every change to the POM file, there is a risk that the server will no longer accept the add-on and refuse to run. Therefore, the local compilation of the add-on is not sufficient to make sure that the program works properly. It must also run on the server to ensure that it really works for 100 percent. This is because the compiler cannot detect all errors. For example, runtime errors or database errors can only be detected at runtime. Dependencies can also cause problems that will only show on runtime.

Repeat step 1 to 4 until all code slices are migrated

Now the first migration step has been completed. If everything runs smoothly, the next migration step can be started until all code pieces have been migrated.

6.2.4 Additional Problems

In this chapter, we discuss additional problems such as security vulnerabilities, compatibility and stability aspects, improvements and bug fixing.

6.2.5 Security Vulnerabilities

The Jira platform provides HTTPS[8] secured connections for all requests and responses between the server and the client. As an unauthorized user it is not possible to read or write any timesheet-specific data. The following statements are only valid if the user is logged in. The user needs a Jira account and is able to access the Timesheet add-on. He does not need an own timesheet to execute harmful attacks. First, a brief introduction about the HTTP[9] protocol is given. Afterwards two security gaps are discussed.

Structure

The client communicates with the server via AJAX[10] calls. A request is sent to the server, the server processes the request and sends back a response to the client. This can be trivial user data such as the user's status or when he was online the last time. But there are also sensitive data transfers which should not be viewed by anyone. A REST call cannot only be used to query data. It is also possible to modify or delete data. In the background the HTTP protocol defines different states how a client can communicate with the server. Table 6.2 shows the most common states which have also been used by our add-on.

[8] `https://www.techopedia.com/definition/5361/hypertext-transport-protocol-secure-https` (accessed 2017-04-14)

[9] `https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html` (accessed 2017-04-14)

[10] `https://www.techopedia.com/definition/24402/asynchronous-javascript-and-xml-ajax` (accessed 2017-04-14)

State	Description
POST	A resource is transmitted from the client to the server. A new resource is created or an existing one is modified. The POST method does not have the property “idempotence“. Therefore, it can happen that the same request leads to different behavior.
GET	The client requests a resource from the server. If the resource is available, the server responds with the requested resource.
PUT	A resource is sent to the server. If the resource does not exist, a new one is created, otherwise the existing one is modified. The PUT method has the property “idempotence“.
DELETE	An existing resource should be deleted.

Table 6.2: Most common States of the HTTP Protocol (W3, 2017)

Security gap number 1 – the self-developed REST call interfaces

REST calls must be secured from unauthorized usage. Jira provides a REST API[11] and those REST interfaces are protected. To execute some of them, administrator access is compulsory. However, the Timesheet add-on uses also self-defined REST interfaces to provide a broader communication base. Exactly those interfaces are vulnerable to security attacks. When the server receives a request from the client it only proves if the user exists. There are no further checks like is the user an administrator or a coordinator of a team. An attacker can simply crawl all data by entering the right rest call URL. He only has to know the correct URL address. He just needs to open a preferred web browser and open the console. For instance, in Google Chrome you can see all calls which were sending and receiving data by the client. Moreover you can see if the REST call was successful. If not, an error code is shown which helps to narrow down the problem.

But not only data can be read, it is also possible to manipulate data. This is possible because the server only checks if the user exists, but no permission

[11]`https://docs.atlassian.com/jira/REST/server/` (accessed 2017-04-14)

Permission	Description
Anonymous User	No Access
Jira User	Can view his timesheet.
Coordinator	Can see all timesheets of his team.
Read Only User	Can see all available timesheets. Only has read access.
Timesheet Administrator	Can read, modify and delete all timesheets.
Jira Administrator	Can only manage the Jira platform. Has no access to timesheet data.

Table 6.3: Permissions a user can have

checks are carried out. Thus an attacker can manipulate a timesheet - or in the worst delete it – if he only knows the specified timesheet ID. Due to the fact that the IDs are numbered consecutively from 1 to infinity it should not be a high effort to guess the correct ID. The timesheet ID is used very often and so many REST calls only need the ID as parameter. Because of all these facts, a capable user can cause enormous damage to the whole system.

To fix that problem, several permission checks were inserted. Table 6.3 shows the different permissions a user can have.

Some of the user roles already exist, but there were no permissions mapped with those. This step has been completed now. It should be mentioned that the Jira administrator also had full access to all timesheet data, but did not need them. Even not project specific persons can be a Jira administrator, so access rights were restricted for this role. Now they do not have access rights to view all timesheet data. The timesheet administrator role is introduced and has full access rights.

Security gap number 2 – administrator sides are open

The Timesheet add-on consists of two parts. In the user area users can manage their timesheets. Coordinators have an overview about all timesheets of a specific team. A coordinator can only see the timesheets of his team members. In the administrator area admins can create or delete a category or assign team members to a team. Normally if a user is not an administrator the links to the admin sections are hidden but still accessible. If the user knows the URL address of the admin area, he can access it because the server only checks if the user exists. Therefore the same prevention was applied as before. Moreover, it was possible to import or export all timesheets. In the absolute worst case scenario, a normal user was able to drop the database of the timesheet data. Now the server checks if the user has the permissions, otherwise he will be redirected to Jira login page.

6.2.6 Compatibility, Stability and Performance

The Timesheet add-on was tested on Google Chrome and Opera browser. All other browsers were not supported in the past.

In Firefox there was a nasty problem with the sending of timesheet data. The JS function click of the saveButton object had no event, but it was needed. So we defined an object event as a parameter for the event handler and it worked. In Google Chrome browser it worked without defining it because it tries to find the event variable in the global scope. Chrome always provides the event object in the global scope, hence no error occurred, but Firefox does not (Kling, 2017).

To solve the bug we used the following two statements:

```
form.saveButton.click(function (event) {
    event.preventDefault();

    ...
}
```

First, we pass the parameter event to the function. Afterwards we define that the execution of the standard behaviour should be ignored. When we click on the save button, no event is executed and the site is not reloaded anymore. We used the function preventDefault which prevents the execution of an event. The final solution can be found on Stackoverflow.com where this case is described in more detail (Kling, 2017).

Apple products had the problem that the Safari[12] engine was not able to understand the time format. The reason was that we used an unsupported date and time format. Firefox and Chrome support this time format, hence no error occurred. You can read more about it under (Stackoverflow, 2017). The add-on was also tested with iOS smartphones, Ipads and Mac Books.

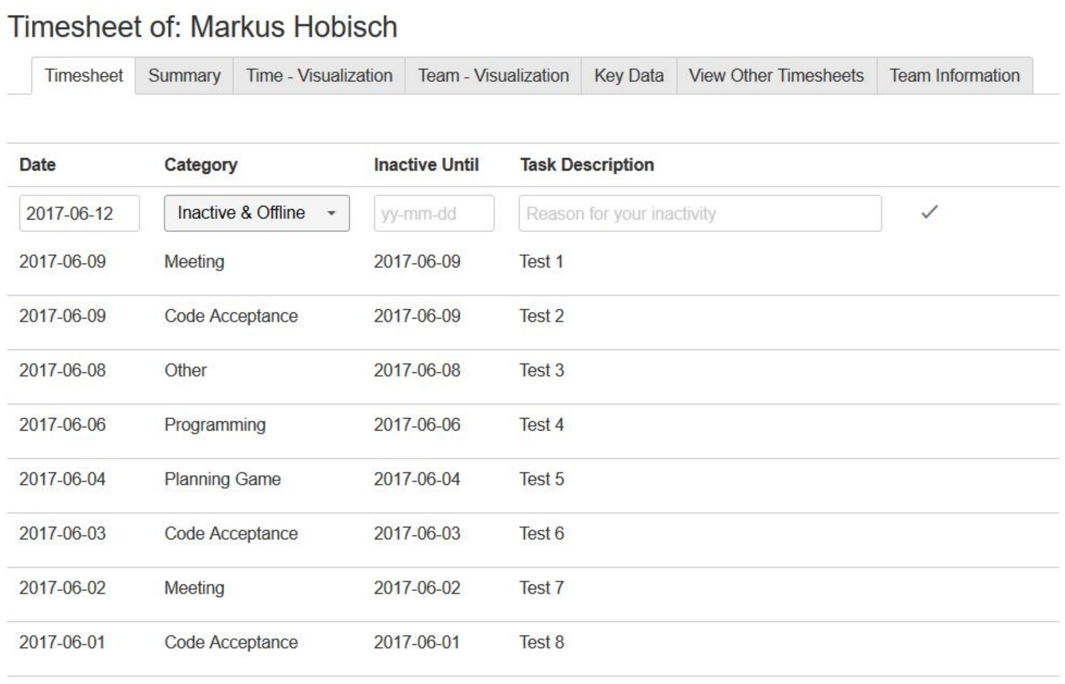
Microsoft's newest browser Edge[13] is now supported as well as Firefox and Safari. The add-on was tested on different operating systems such as Windows[14], Linux[15] and Mac OS[16].

After the migration, the timesheet was very slow and did not react very smoothly to user inputs. The reason was that in the JS source code many data were queried several times from the server and therefore the loading of the page was very long. Now the data are loaded only once from the server and cached locally. This step significantly improved the processing speed of the add-on. With a powerful code analysing tool which is integrated in IntelliJ IDEA from JetBrains[17] it was possible to detect buggy code e.g. code that always returns true or false independent from the input. Thus it was possible to increase the code quality and hence the system stability.

6.2.7 Improvements

The user interface of the timesheet has been redesigned. Due to the lack of screen space some timesheet columns were hidden by default. They will

[12]https://www.apple.com/safari/ (accessed 2017-04-14)
[13]https://www.microsoft.com/de-at/windows/microsoft-edge (accessed 2017-04-14)
[14]https://www.microsoft.com/de-at/windows/get-windows-10 (accessed 2017-04-14)
[15]https://www.ubuntu.com/ (accessed 2017-04-14)
[16]https://www.apple.com/at/macos/sierra/ (accessed 2017-04-14)
[17]https://www.jetbrains.com/idea/ (accessed 2017-04-28)


Timesheet of: Markus Hobisch

Date	Category	Inactive Until	Task Description	
2017-06-12	Inactive & Offline	yy-mm-dd	Reason for your inactivity	✓
2017-06-09	Meeting	2017-06-09	Test 1	
2017-06-09	Code Acceptance	2017-06-09	Test 2	
2017-06-08	Other	2017-06-08	Test 3	
2017-06-06	Programming	2017-06-06	Test 4	
2017-06-04	Planning Game	2017-06-04	Test 5	
2017-06-03	Code Acceptance	2017-06-03	Test 6	
2017-06-02	Meeting	2017-06-02	Test 7	
2017-06-01	Code Acceptance	2017-06-01	Test 8	

Figure 6.1: Inactive category is selected

only be shown if a special category is selected in the drop down menu. These special categories are:

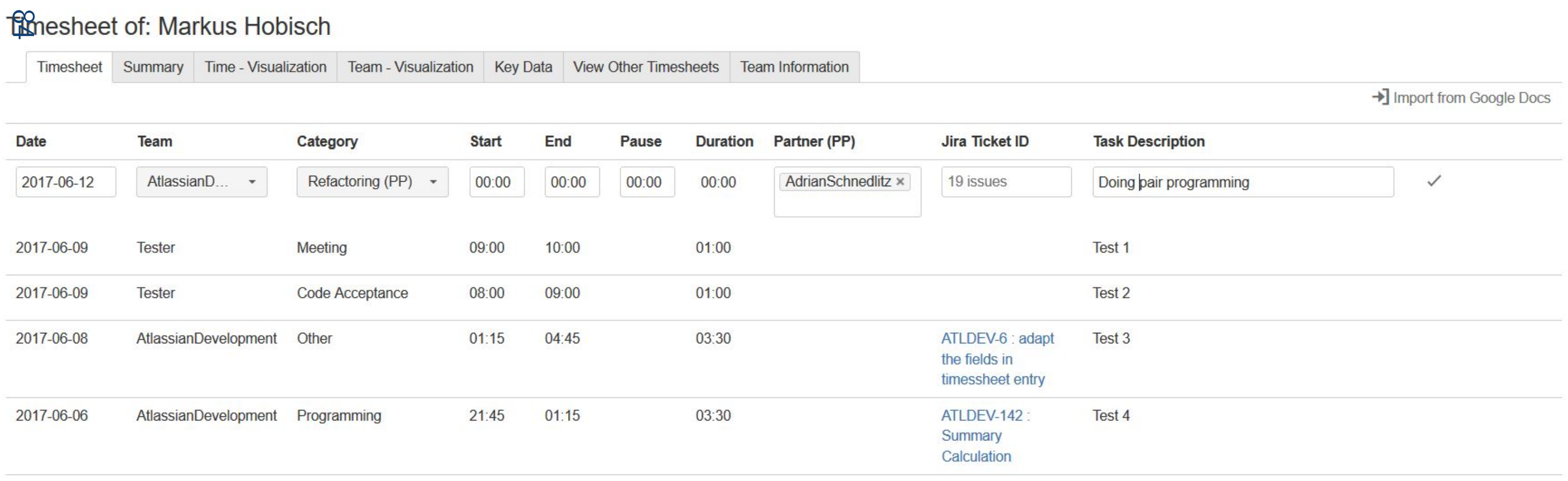
- Inactive
- Inactive & Offline
- all categories which contain the string "PP"

Inactive, Inactive & Offline category

If one of these categories is selected, only columns Date, Category, Inactive Until and Task Description are visible. In Figure 6.1 the inactive category is selected.

Refactoring (PP)

If a category contains the string “PP”, it is clear that it is a pair programming category. For this purpose the pair programming field previously called partner is shown. Figure 6.2 shows the selected pair programming category.



Date	Team	Category	Start	End	Pause	Duration	Partner (PP)	Jira Ticket ID	Task Description	
2017-06-12	AtlassianD...	Refactoring (PP)	00:00	00:00	00:00	00:00	AdrianSchnedlitz ×	19 issues	Doing pair programming	✓
2017-06-09	Tester	Meeting	09:00	10:00		01:00			Test 1	
2017-06-09	Tester	Code Acceptance	08:00	09:00		01:00			Test 2	
2017-06-08	AtlassianDevelopment	Other	01:15	04:45		03:30		ATLDEV-6 : adapt the fields in timesheet entry	Test 3	
2017-06-06	AtlassianDevelopment	Programming	21:45	01:15		03:30		ATLDEV-142 : Summary Calculation	Test 4	

Figure 6.2: Pair Programming category is selected

6.3 Software Testing

After the migration process, it had to be ensured that the requirements were met. To validate the correctness of the program all tests were started. Many of them had already been written by Adrian Schnedlitz but were also adapted to the new platform version. Because the tests did not cover the new functionalities, new test cases were implemented. Our testing phase consists of three parts:

- Test Environment
- Test Framework
- Additional Software Testing Group

Software tests are very powerful to find bugs in code. Every time you change a code piece, you can run all test cases and see if the implementation still fulfils the requirements. If not, you can easily find where the problem is. But test cases cannot test all kinds of problems. For instance, performance issues can only be tested on the physical server. The reaction time depends on the network connection to the server. If the connection is very fast, you will not see any difference if the data are requested from the server several times or just once. But if data is not cached, this can lead to huge performance degradation.

The previous example is very similar to the next one. Security issues are very hard to handle because they are not so obvious. The add-on is protected over HTTPS which prevents the recording of data. But nobody has considered that all data were automatically decoded by the client. You only need a Jira account to get access to the necessary page. In our case the weakest link in the chain was not the encryption but the server which did not check if a user had the permissions to get the requested data.

6.3.1 Test Environment

We developed the add-on in a virtual machine locally. Due to the fact that the behavior of a virtual server running on a virtual machine could be different compared to a physical one we also used a test server for testing. We used a test server with a live demo version. The test server had an old Jira backup version from our productive live instance and was updated to Jira 7. There we tried out all the different states the add-on can have. Several users tried to add or delete timesheet entries simultaneously. Bugs were reported in Jira.

As mentioned earlier, security and performance issues were identified during the test phase. The stability was increased by eliminating bugs in rarely used functions. Database errors such as data inconsistency were also fixed. We used the timesheet importer and exporter to save and restore timesheet data like configuration settings or timesheet data.

6.3.2 Test Framework

As test framework we used PowerMock[18] combined with normal JUnit[19] tests. PowerMock is a framework that uses bytecode manipulation to enable mocking of static methods, constructors, final classes and methods, private methods and many more. It uses a custom class loader. It extends many other mocking libraries such as EasyMock[20].

6.3.3 Additional Software Testing Group

The add-on was tested by the Catrobat team. Users reported many bugs which finally helped us to improve the stability of the add-on. After the add-on had been well tested it was rolled out to all students. The current Google timesheet was replaced by our own one.

[18] http://powermock.github.io/ (accessed 2017-04-28)
[19] http://junit.org/junit4/ (accessed 2017-04-30)
[20] http://easymock.org/ (accessed 2017-04-30)

6.3.4 Updating Jira Software Server & Backups

The Jira test server and the productive instance were updated to the latest version (currently 7.2). The upgrading process was first applied to the test instance and on success also to the productive instance. This ensured the failure safety.

6.4 Results

The Timesheet add-on was successfully updated from version 6 to 7. Many additional requirements were implemented to complete the add-on. A test run was done with Catrobat team members to ensure that all functions meet the requirements. The add-on was well tested and rolled out for all Catrobat team members. It should be mentioned that the current add-on is only compatible as long as the main version is not changed. The Atlassian SDK is neither upwards nor downwards compatible. Due to the fact that Atlassian makes big changes in each major version it cannot be guaranteed that it will still work with future releases.

7 Summary and Future Work

In the final chapter we want to recapitulate the most important information and give an outlook for future work.

7.1 Summary

The Timesheet add-on is an online timetracking tool for education and helps students to manage their time better. It is used by the Catrobat organisation because it completely fulfils the requirements of the Catrobat project. The add-on was upgraded to version 7, otherwise the add-on could not have been used in our project. By switching to the new major version, many problems arose which could be solved successfully. The add-on had to be migrated to the new platform. We chose the Chicken Little methodology as migration strategy. The pros and cons of this approach were discussed.

According to the Chicken Little approach 2.2.3, the source code was split into pieces and then migrated step by step. The Cold Turkey approach would have resulted in a new development and would have been too laborious. In principle, the migration would have been possible in one step but it would have been very difficult. The complexity was very high and perhaps only an expert could have done it. Therefore, a very large problem was divided into many smaller parts because they were easier to handle.

After the migration process stability, safety, compatibility and speed were emphasized. Security features were implemented that check if a user has the permission to request a resource. Many web browsers are supported now. Also the add-on is more stable and can react faster on user inputs through further improvements.

Finally, a test run was started with the Catrobat team members to ensure the correctness of the program. Bugs reported by members were fixed and the availability was increased. This guarantees the future usability of the Timesheet add-on. The add-on is already used in Catrobat.

7.2 Future Work

Future Work can include requirements like ones in Table 7.1.

The main points relate to the email notifications. The settings could be extended so that it is possible to decide who is informed and how often the person is informed during a period of time. Currently users only get email notifications if they are inactive but not when they are disabled. Also it would be better for coordinators and timesheet administrators to get summarized reports about all user activities at once. This would reduce the number of notifications and would help to improve the overview. These reports could be sent in specific time intervals, e.g. once a month.

The team visualization works only for one team. If a user is in more than one team, the diagrams are not drawn correctly. The reason for this problem is currently unknown. It might be great if it also worked for many teams.

A great feature would be the automatic backup of all timesheets on the server or in a cloud. Not every user would have to save his timesheet locally from time to time. Configuration settings could also be saved. In the case of a total failure, the data would not be lost. This would significantly reduce the damage and would help to recover the system much faster.

Last but not least, it would be nice to reimport a timesheet in the same way as it was exported before. Currently when a timesheet is exported, a new file is created but it cannot be uploaded to the server again. Instead, the file must be opened and all entries must be copied manually. This is very cumbersome. It would be better if the file could be uploaded directly to the server.

Requirement	Description
Notification	It should be possible to specify who receives notifications
Notification	The email notifications should have a description
Notification	Users should be informed when they were disabled
Notification	Inactive users should receive emails periodically
E-Mail	Coordinators should receive summary reports about all inactive and disabled users
Visualization	Team visualizations should work for more than one team
Database Management	Changes in database should be logged
Database Management	All timesheets should be periodically saved by the system
Language Support	German language should also be supported

Table 7.1: Future Requirements

Appendix

The project was renamed from TimePunch in Timesheet for legal reasons.
The complete code of the **Timesheet** can be found on **Github** -
`https://github.com/Catrobat/Timesheet`

Bibliography

Allman, Eric (2012). "Managing Technical Debt." In: *Commun. ACM* 55.5, pp. 50–55. ISSN: 0001-0782. DOI: 10.1145/2160718.2160733. URL: http://doi.acm.org/10.1145/2160718.2160733 (cit. on p. 39).

Apache (2017). *Apache Homepage*. visited on 2017-06-05. URL: http://velocity.apache.org (cit. on p. 7).

Arnold, M. and T. Braithwaite (2017). *Homepage Financial Times*. visited on 2017-05-21. URL: https://www.ft.com/content/90360dbe-15cb-11e5-a58d-00144feabdc0 (cit. on p. 34).

Atlassian (2017a). *Active Objects*. visited on 2017-06-05. URL: https://developer.atlassian.com/docs/atlassian-platform-common-components/active-objects (cit. on p. 46).

Atlassian (2017b). *Atlassian Support End of Life Policy*. visited on 2017-04-07. URL: https://confluence.atlassian.com/support/atlassian-support-end-of-life-policy-201851003.html (cit. on p. 48).

Atlassian (2017c). *Getting Started*. visited on 2017-04-07. URL: https://developer.atlassian.com/docs/getting-started (cit. on p. 48).

Atlassian (2017d). *JIRA 7 removes non-deprecated methods in UserProjectHistoryManager*. visited on 2017-04-12. URL: https://jira.atlassian.com/browse/JRASERVER-43526 (cit. on p. 70).

Bateman, A. and J. Murphy (1994). "Migration of Legacy Systems." In: *School of Computer Applications. Dublin: Dublin City University: Dublin* (cit. on pp. 25, 27).

Bennett, K. (1995). "Legacy systems coping with success." In: *IEEE Software* 12.1, pp. 19–23. ISSN: 0740-7459. DOI: 10.1109/52.363157 (cit. on p. 10).

Bennett, K. H., M. Ramage, and M. Munro (1999). "Decision model for legacy systems." In: *IEE Proceedings - Software* 146.3, pp. 153–159. ISSN: 1462-5970. DOI: 10.1049/ip-sen:19990617 (cit. on p. 10).

Bianchi, A. et al. (2003). "Iterative reengineering of legacy systems." In: *IEEE Transactions on Software Engineering* 29.3, pp. 225–241. ISSN: 0098-5589. DOI: 10.1109/TSE.2003.1183932 (cit. on pp. 24, 28).

Bisbal, J., D. Lawless, Bing Wu, and J. Grimson (1999). "Legacy information systems: issues and directions." In: *IEEE Software* 16.5, pp. 103–111. ISSN: 0740-7459. DOI: 10.1109/52.795108 (cit. on pp. xvi, 10, 13, 14, 17, 20, 21, 23, 24, 28–30).

Bisbal, J., D. Lawless, Bing Wu, J. Grimson, et al. (1997). "An overview of legacy information system migration." In: *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, pp. 529–530. DOI: 10.1109/APSEC.1997.640219 (cit. on p. 25).

Boehm, Barry W. (1981). *Software Engineering Economics*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0138221227 (cit. on pp. xvi, 36, 37).

Brodie, Michael L. and Michael Stonebraker (1995). *Migrating legacy systems: Gateways, interfaces and the incremental approach*. The Morgan Kaufmann series in data management systems. San Francisco, Calif: Kaufmann Publ. ISBN: 1558603301 (cit. on pp. 23, 24).

Brodie, Michael L., Michael Stonebraker, and Se Ai (1993). *DARWIN: On the Incremental Migration of Legacy Information Systems* (cit. on pp. 23–25, 27).

Burwinklepple, Christine (2017). *End of support for JIRA 6.4: FAQs about upgrading*. visited on 2017-04-07. URL: https://www.atlassian.com/blog/jira-software/end-of-support-for-jira-6-4 (cit. on p. 48).

Carr, D. and R. J. Kizior (2000). "The case for continued Cobol education." In: *IEEE Software* 17.2, pp. 33–36. ISSN: 0740-7459. DOI: 10.1109/52.841603 (cit. on p. 34).

Carver, J. C. et al. (2017). "GitHub, Technical Debt, Code Formatting, and More." In: *IEEE Software* 34.2, pp. 105–107. ISSN: 0740-7459. DOI: 10.1109/MS.2017.51 (cit. on p. 41).

Catrobat (2017). *PocketCode Homepage*. visited on 2017-04-07. URL: https://share.catrob.at/pocketcode/ (cit. on p. 55).

Chen, H. et al. (2014). "Device driver generation targeting multiple operating systems using a model-driven methodology." In: *2014 25nd IEEE International Symposium on Rapid System Prototyping*, pp. 30–36. DOI: 10.1109/RSP.2014.6966689 (cit. on p. 15).

Cimitile, A., H. Muller, and R. R. Klosch (eds.) (1997). "Pulling Together." In: *Proceedings of the ICSE-97 on Software Engineering. Workshop on Migration Strategies for Legacy Systems. Available as Technical Report TUV-1841-97-06 from Technical University of Vienna, A-1040 Vienna, Austria* (cit. on p. 10).

Colosimo, Massimo et al. (2009). "Evaluating legacy system migration technologies through empirical studies." In: *Information and Software Technology* 51.2, pp. 433–447. ISSN: 0950-5849. DOI: `http://dx.doi.org/10.1016/j.infsof.2008.05.012`. URL: `http://www.sciencedirect.com/science/article/pii/S0950584908000694` (cit. on p. 11).

Comella-Dorda, S. et al. (2000). "A survey of black-box modernization approaches for information systems." In: *Proceedings 2000 International Conference on Software Maintenance*, pp. 173–183. DOI: `10.1109/ICSM.2000.883039` (cit. on pp. xviii, 12, 17–19).

Confluence (2017). *Confluence Homepage*. visited on 2017-06-17. URL: `https://www.atlassian.com/software/confluence` (cit. on p. 5).

Crotty, James and Ivan Horrocks (2016). "Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company." In: *Applied Computing and Informatics*, pp. -. ISSN: 2210-8327. DOI: `https://doi.org/10.1016/j.aci.2016.12.001`. URL: `http://www.sciencedirect.com/science/article/pii/S2210832716301260` (cit. on p. 34).

Cunningham, Ward (1992). "The WyCash Portfolio Management System." In: *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*. OOPSLA '92. Vancouver, British Columbia, Canada: ACM, pp. 29–30. ISBN: 0-89791-610-7. DOI: `10.1145/157709.157715`. URL: `http://doi.acm.org/10.1145/157709.157715` (cit. on p. 38).

Eilebrecht, Karl and Gernot Starke (2010a). "Einleitung." In: *Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung*. Heidelberg: Spektrum Akademischer Verlag, pp. 1–4. ISBN: 978-3-8274-2526-3. DOI: `10.1007/978-3-8274-2526-3_1`. URL: `http://dx.doi.org/10.1007/978-3-8274-2526-3_1` (cit. on p. 42).

Eilebrecht, Karl and Gernot Starke (2010b). "Grundlagen des Software-Entwurfs." In: *Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung*. Heidelberg: Spektrum Akademischer Verlag, pp. 5–18. ISBN: 978-3-8274-2526-3. DOI: `10.1007/978-3-8274-2526-3_2`. URL: `http://dx.doi.org/10.1007/978-3-8274-2526-3_2` (cit. on pp. 42, 43).

Erdle, Christoph (2005). “Management von Softwaresystemen.” In: URL: http://www4.in.tum.de/lehre/seminare/hs/WS0506/mvs/files/Ausarbeitung_Erdle.pdf (cit. on pp. 28, 29).

Fontana, F. A., V. Ferme, and S. Spinelli (2012). “Investigating the impact of code smells debt on quality code evaluation.” In: *2012 Third International Workshop on Managing Technical Debt (MTD)*, pp. 15–22. DOI: 10.1109/MTD.2012.6225993 (cit. on p. 41).

Fowler, Martin et al. (1999). *Refactoring: Improving the Design of Existing Code.* Boston, MA, USA: Addison-Wesley. ISBN: 0-201-48567-2 (cit. on pp. 39, 41).

Freeman, Eric and Elisabeth Freeman (2006). *Entwurfsmuster von Kopf bis Fuß.* Köln: O’Reilly. ISBN: 3-89721-421-0 (cit. on pp. 43, 45).

Gangadharan, G. R. et al. (2013). “IT Innovation Squeeze: Propositions and a Methodology for Deciding to Continue or Decommission Legacy Systems.” In: *Grand Successes and Failures in IT. Public and Private Sectors: IFIP WG 8.6 International Working Conference on Transfer and Diffusion of IT, TDIT 2013, Bangalore, India, June 27-29, 2013. Proceedings.* Ed. by Yogesh K. Dwivedi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 481–494. ISBN: 978-3-642-38862-0. DOI: 10.1007/978-3-642-38862-0_30. URL: http://dx.doi.org/10.1007/978-3-642-38862-0_30 (cit. on p. 34).

Git (2017). *Git Homepage.* visited on 2017-06-17. URL: https://git-scm.com/ (cit. on p. 5).

Google (2017). *Google Homepage.* visited on 2017-06-05. URL: https://developers.google.com/closure/templates/ (cit. on p. 46).

Goyla, F. P. (2000). “Legacy integration-changing perspectives [Cobol].” In: *IEEE Software* 17.2, pp. 37–41. ISSN: 0740-7459. DOI: 10.1109/52.841604 (cit. on p. 17).

H2database (2017). *H2database Homepage.* visited on 2017-04-12. URL: http://www.h2database.com/html/performance.html#performance_comparison (cit. on p. 70).

Ismail, Nick (2017). *Legacy systems: the next financial crisis?* visited on 2017-05-21. URL: http://www.information-age.com/legacy-systems-next-financial-crisis-123465888/ (cit. on p. 34).

Karthikeyan, T. and T. Nandhini (2016). “Dependent component cost model of legacy application for hybrid cloud.” In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–5. DOI: 10.1109/ICCPCT.2016.7530154 (cit. on p. 35).

KCOM (2017). *KCOM Homepage*. visited on 2017-05-11. URL: http://www.kcom.com/connected-thinking/opinion/calculating-the-true-cost-of-legacy-it-systems/ (cit. on p. 32).

Kent, J. and M. Dewey (2016). "Legacy test systems Replace or maintain." In: *2016 IEEE AUTOTESTCON*, pp. 1–5. DOI: 10.1109/AUTEST.2016.7589645 (cit. on p. 15).

Kling, Felix (2017). *Ajax post working in Chrome, but not in Firefox*. visited on 2017-05-08. URL: http://stackoverflow.com/questions/18274383/ajax-post-working-in-chrome-but-not-in-firefox (cit. on pp. 80, 81).

Koitz, Roxane and Wolfgang Slany (2014). "Empirical Comparison of Visual to Hybrid Formula Manipulation in Educational Programming Languages for Teenagers." In: *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools*. PLATEAU '14. Portland, Oregon, USA: ACM, pp. 21–30. ISBN: 978-1-4503-2277-5. DOI: 10.1145/2688204.2688209. URL: http://doi.acm.org/10.1145/2688204.2688209 (cit. on p. 55).

Li, X. (2010). "A multi-Agent based legacy information system integration strategy." In: *2010 International Conference on Networking and Digital Society*. Vol. 2, pp. 72–75. DOI: 10.1109/ICNDS.2010.5479398 (cit. on pp. 10, 14).

Li, Z., P. Avgeriou, and P. Liang (2015). "A systematic mapping study on technical debt and its management." In: *Journal of Systems and Software* 101. PT: J; NR: 33; TC: 5; J9: J SYST SOFTWARE; PG: 28; GA: CB3CY; UT: WOS:000349507000015, pp. 193–220. ISSN: 0164-1212. DOI: 10.1016/j.jss.2014.12.027 (cit. on p. 39).

Littlejohn, K., M. V. DelPrincipe, and J. D. Preston (2000). "Embedded information system re-engineering." In: *IEEE Aerospace and Electronic Systems Magazine* 15.11, pp. 3–7. ISSN: 0885-8985. DOI: 10.1109/62.888319 (cit. on pp. 12, 14, 15).

Makki, S. K. (2006). "The Integration and Interoperability Issues of Legacy and Distributed Systems." In: *2006 Seventh International Conference on Web-Age Information Management Workshops*, pp. 21–21. DOI: 10.1109/WAIMW.2006.30 (cit. on pp. xvi, 11).

Martini, A., T. Besker, and J. Bosch (2016). "The Introduction of Technical Debt Tracking in Large Companies." In: *2016 23rd Asia-Pacific Software*

Engineering Conference (APSEC), pp. 161–168. DOI: 10.1109/APSEC.2016.032 (cit. on pp. 40, 41).

Menychtas, A., K. Konstanteli, et al. (2014). “Software modernization and cloudification using the artist migration methodology and framework.” English. In: *Scalable Computing* 15.2, pp. 131–152. URL: www.scopus.com (cit. on p. 12).

Menychtas, A., C. Santzaridou, et al. (2013). “ARTIST Methodology and Framework: A Novel Approach for the Migration of Legacy Software on the Cloud.” In: *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 424–431. DOI: 10.1109/SYNASC.2013.62 (cit. on p. 12).

Mukhija, Arun (2003). *Estimating Software Maintenance*. Requirements Engineering Research Group Institut fuer Informatik Universitaet Zuerich (cit. on pp. xvi, 37).

Paulson, L. D. (2001). “Mainframes, Cobol still popular.” In: *IT Professional* 3.5, pp. 12–14. ISSN: 1520-9202. DOI: 10.1109/6294.952975 (cit. on pp. 33, 34).

Power, K. (2013). “Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options.” In: *2013 4th International Workshop on Managing Technical Debt (MTD)*, pp. 28–31. DOI: 10.1109/MTD.2013.6608675 (cit. on p. 39).

Sarrab, Mohamed, Mahmoud Elbasir, and Laila Elgamel (2013). “The Technical, Non-technical Issues and the Challenges of Migration to Free and Open Source Software.” In: *IJCSI International Journal of Computer Science Issues* Vol. 10, Issue 2, No 3.ISSN (Print): 1694-0814 — ISSN (Online): 1694-0784, pp. 464–469 (cit. on p. 22).

Schnedlitz, Adrian (2016). “TimePunch - An Online Timetracking Tool for Education.” MA thesis. Graz University of Technology (cit. on pp. 3, 7, 48, 66, 68).

Schneidewind, N. F. and C. Ebert (1998). “Preserve or Redesign Legacy Systems [Guest Editor’s Introduction].” In: *IEEE Software* 15.4, pp. 14–17. ISSN: 0740-7459. DOI: 10.1109/MS.1998.687937 (cit. on p. 35).

Scratch (2017). *Scratch Homepage*. visited on 2017-04-07. URL: https://scratch.mit.edu/ (cit. on p. 57).

Slany, W. (2012). “A mobile visual programming system for Android smartphones and tablets.” In: *2012 IEEE Symposium on Visual Languages and*

Human-Centric Computing (VL/HCC), pp. 265–266. DOI: 10.1109/VLHCC.2012.6344546 (cit. on pp. 55, 57).

Slany, Wolfgang (2014). "Pocket Code: A Scratch-like Integrated Development Environment for Your Phone." In: *Proceedings of the Companion Publication of the 2014 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity*. SPLASH '14. Portland, Oregon, USA: ACM, pp. 35–36. ISBN: 978-1-4503-3208-8. DOI: 10.1145/2660252.2664662. URL: http://doi.acm.org/10.1145/2660252.2664662 (cit. on pp. 53, 55).

Sneed, H. M. (2001). "Extracting business logic from existing COBOL programs as a basis for redevelopment." In: *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*, pp. 167–175. DOI: 10.1109/WPC.2001.921728 (cit. on pp. 16, 33).

Souiou, Wafa and Nora Bounour (2013). "Migration of legacy systems to service oriented architecture." In: *The Second International Conference on Digital Enterprise and Information Systems (DEIS2013)*. The Society of Digital Information and Wireless Communication, pp. 166–173 (cit. on pp. 11, 12).

Stackoverflow (2017). *Stackoverflow Homepage*. visited on 2017-04-09. URL: http://stackoverflow.com/questions/3085937/safari-js-cannot-parse-yyyy-mm-dd-date-format (cit. on p. 81).

Stangaroneifi20, Joe (2017). *What is the true cost of your legacy applications?* visited on 2017-05-21. URL: http://www.mrc-productivity.com/blog/2015/03/whats-the-true-cost-of-your-legacy-applications/ (cit. on pp. 32, 33).

Store, Android (2017). *Android Store Homepage*. visited on 2017-04-07. URL: https://play.google.com/store/apps/details?id=org.catrobat.catroid&hl=de (cit. on p. 57).

Technopedia (2017a). *Technopedia Hoempage*. visited on 2017-06-17. URL: https://www.techopedia.com/definition/24022/remote-procedure-call-rpc (cit. on p. 7).

Technopedia (2017b). *Technopedia Homepage*. visited on 2017-06-17. URL: https://www.techopedia.com/definition/24407/application-programming-interface-api (cit. on p. 4).

Technopedia (2017c). *Technopedia Homepage*. visited on 2017-04-14. URL: https://www.techopedia.com/definition/24402/asynchronous-javascript-and-xml-ajax (cit. on p. 4).

Technopedia (2017d). *Technopedia Homepage*. visited on 2017-07-10. URL: https://www.techopedia.com/definition/24308/fourth-generation-programming-language-4gl (cit. on p. 5).

Technopedia (2017e). *Technopedia Homepage*. visited on 2017-04-14. URL: https://www.techopedia.com/definition/5361/hypertext-transport-protocol-secure-https (cit. on p. 5).

Technopedia (2017f). *Technopedia Homepage*. visited on 2017-06-17. URL: https://www.techopedia.com/definition/25326/legacy-code (cit. on p. 6).

Technopedia (2017g). *Technopedia Homepage*. visited on 2017-06-17. URL: https://www.techopedia.com/definition/635/legacy-system (cit. on p. 6).

Technopedia (2017h). *Technopedia Homepage*. visited on 2017-04-28. URL: https://www.techopedia.com/definition/1352/uniform-resource-locator-url (cit. on p. 7).

Tripathy, Priyadarshi and Kshirasagar Naik (2014a). "Basic Concepts and Preliminaries." In: *Software Evolution and Maintenance*. John Wiley & Sons, Inc., pp. 1–24. ISBN: 9781118964637. DOI: 10.1002/9781118964637.ch1. URL: http://dx.doi.org/10.1002/9781118964637.ch1 (cit. on p. 10).

Tripathy, Priyadarshi and Kshirasagar Naik (2014b). "Legacy Information Systems." In: *Software Evolution and Maintenance*. John Wiley & Sons, Inc., pp. 187–222. ISBN: 9781118964637. DOI: 10.1002/9781118964637.ch5. URL: http://dx.doi.org/10.1002/9781118964637.ch5 (cit. on pp. xvi, 17, 23, 25–28).

UECD (2010). *Handbook on Deriving Capital Measures of Intellectual Property Products*. G - Reference, Information and Interdisciplinary Subjects Series. OECD Publishing. ISBN: 9789264072909. URL: https://books.google.at/books?id=UyB1CWnftQ4C (cit. on pp. xviii, 36).

W3 (2017). *W3 Homepage*. visited on 2017-04-14. URL: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html (cit. on pp. xviii, 78).

Webopedia (2017). *webopedia homepage*. visited on 2017-06-17. URL: http://www.webopedia.com/TERM/C/comma_delimited.html (cit. on p. 4).

Wu, Bing, D. Lawless, J. Bisbal, J. Grimson, et al. (1997). "Legacy systems migration-a method and its tool-kit framework." In: *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, pp. 312–320. DOI: 10.1109/APSEC.1997.640188 (cit. on p. 28).

Wu, Bing, D. Lawless, J. Bisbal, R. Richardson, et al. (1997). “The Butterfly Methodology: a gateway-free approach for migrating legacy information systems.” In: *Proceedings. Third IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.97TB100168)*, pp. 200–205. DOI: 10.1109/ICECCS.1997.622311 (cit. on p. 28).

Yli-Huumo, Jesse, Andrey Maglyas, and Kari Smolander (2016). “How do software development teams manage technical debt? An empirical study.” In: *Journal of Systems and Software* 120, pp. 195–218. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2016.05.018. URL: http://www.sciencedirect.com/science/article/pii/S016412121630053X (cit. on pp. 38–40).

Zapata, F. et al. (2015). “How to speed up software migration and modernization: Successful strategies developed by precisiating expert knowledge.” In: *2015 Annual Conference of the North American Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC)*, pp. 1–6. DOI: 10.1109/NAFIPS-WConSC.2015.7284166 (cit. on p. 12).