FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Software Project Longevity – A Case Study on Open Source Software Development Projects

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Bernhard Kiselka
Matrikelnummer 0125881

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Mag. Mag.rer.soc.oec. Dr.techn. Stefan Biffl
Mitwirkung: Projektass. Dipl.-Ing. Dietmar Winkler

Wien, 30.11.2015

(Unterschrift Verfasser) (Unterschrift Betreuer)

Bernhard Kiselka

0125881, 066 926

Master Thesis

Software Project and Product Longevity
A Case Study on Open Source Software Development Projects

E-Mail: bernhard.kiselka@gmx.at

Phone: 0664/33 23 269

Date: 2015-11-30

For Raphaël

-

Für Raphaël

Contents

Erklärung zur Verfassung der Arbeit

Bernhard Kiselka, Friedrich Schiller-Straße 79b/7/3, 2340 Mödling

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

Mödling, 30.11.2015

Abstract

Metrics on Software Projects measure the past and the status quo. The question if an information system will still be usable in the future is not really answered. On the contrary longevity is seen as an important design issue. Optimally, we want to measure software product longevity right from the start of a project, not in the end when all effort is spent.

This work wants to find out what makes a (open source) software project so successful, that its software is used for a long time, and how to measure this longevity beforehand.

The methodical approach involves a systematic literature research (SLR) on project health indicators, which identifies successfully software projects, metrics and tools. A tool selection process finds the most suitable tools for building a framework for longevity evaluation based on quality attributes from ISO SQuaRE.

The work's result is a set of tools that forms an evaluation framework to estimate quality attributes, metrics, and key performance indicators with focus on software project longevity. Open source software projects from related work evaluate the framework. Main outcome is a short list of metrics measuring quality attributes important for longevity. A feasibility study shows these metrics can be used to measure aspects of longevity.

Kurzfassung

Metriken von Softwareprojekten messen die Vergangenheit und den Status Quo. Die Frage, ob ein Informationssystem in Zukunft immer noch verwendbar ist, wird nicht wirklich beantwortet. Im Gegensatz dazu wird Langlebigkeit als wichtiges Thema des Entwurfs gesehen. Denn optimalerweise wollen wir die Langlebigkeit eines Softwareprodukts bereits vom Start weg messen können, nicht erst am Ende wenn der ganze Aufwand bereits geleistet wurde.

Diese Arbeit will herausfinden, was ein (Open Source) Softwareprojekt so erfolgreich macht, dass dessen Software über lange Zeit verwendet wird, und wie diese Langlebigkeit im Vorhinein gemessen werden kann.

Das wissenschaftliche Vorgehen umfasst eine systematische Literaturrecherche über die Kennzeichen von gesunden Projekten, die erfolgreiche Softwareprojekte, Metriken und Tools identifizieren. Mit einem Auswahlverfahren werden die am Besten geeigneten Tools für ein Framework zur Beurteilung von Langlebigkeit basierend auf Qualitätsattributen von ISO SQuaRE ermittelt.

Das Ergebnis der Arbeit ist eine Menge an Tools die ein Beurteilungsframework bilden, um Qualitätsattribute, Metriken und Leistungskennzahlen mit dem Fokus auf der Langlebigkeit von Softwareprojekten zu beurteilen. Hauptresultat ist eine kurze Liste von Metriken, die Qualitätsattribute messen, welche für die Langlebigkeit wichtig sind. Wie eine Machbarkeitsstudie zeigt, können diese Metriken für Teilaspekte von Langlebigkeit verwendet werden.

Acknowledgements

First I want to thank my advisor, Ao.Univ.Prof. Dipl.-Ing. Mag. Mag.rer.soc.oec. Dr.techn. Stefan Biffl, for inspiring talks and uncomplicated communication. Furthermore I really like to express my gratitude to my main contact, Projektass. Dipl.-Ing. Dietmar Winkler, for the continuous cooperation, his good ideas and guidance.

I thank my parents for their parental care and inspiration. At least equally I thank my Grandma Ottilie Leskusek, who made my study possible due to her financial support.

I am deeply grateful to my girlfriend and partner Mag.pharm. Ines Raunicher. Ultimately due to her constant insistence I started my master thesis. Repeated encouragement and endless patience, when my work faltered, count as her strengths, which at least led to the end of my studies.

Danksagung

An erster Stelle möchte ich meinem Betreuer, Ao.Univ.Prof. Dipl.-Ing. Mag. Mag.rer.soc.oec. Dr.techn. Stefan Biffl, für die anregenden Gespräche und unkomplizierte Kommunikation danken. Weiters möchte ich meinem Hauptansprechpartner, Projektass. Dipl.-Ing. Dietmar Winkler, meinen besonderen Dank für die kontinuierliche Zusammenarbeit, seine guten Ideen und die gute Beratung aussprechen.

Meinen Eltern danke ich für deren elterliche Fürsorge und Inspiration. Zu mindestens ebenso großem Dank bin ich meiner Oma Ottilie Leskusek verpflichtet, durch deren finanzielle Unterstützung dieses Studium erst möglich wurde.

Ganz besonders großer Dank geht an meine Freundin und Partnerin Mag.pharm. Ines Raunicher. Durch ihr stetes Beharren habe ich diese Diplomarbeit letztlich begonnen. Wiederholtes Aufmuntern und endlose Geduld, wenn die Arbeit ins Stocken geriet, zählen zu ihren Stärken, die dazu geführt haben, dass ich mein Studium erfolgreich abschließen kann.

List of Tables

List of Figures

1 Introduction

This work presents a case study on the longevity of software projects. It contains a summary of all the necessary background and all steps that lead to the resulting framework.

Intended readers are quality manager, project managers, software developers and also decision makers. A practical understanding of and/or experience in software development helps reading.

1.1 Methodology

Detailed models of software evolution date back to the 1960's. As pointed out by *Scacchi* there has been a search for process models since the beginning of big software projects [35].

All Software life cycle models provide a scheme for managing the development of software systems. A software life cycle model is a description of how software should be developed [35].

Improving quality is either tried by improving process quality or product quality or both. *Kitchenham and Pfleeger* identify five different views of software quality: a transcendental view (sees quality as something that can be recognized but not defined), user view (sees quality as fitness for purpose), manufacturing view (sees quality as conformance to specification), product view (sees quality as tied to inherent characteristics of the project) and a value-based view (sees quality as dependent on the amount the customer is willing to pay for it). [36]

They also describe how to measure these views, but also state that "the way we measure quality depends on the viewpoint we take and the aspect of quality we want to capture." A technique like the Goal-Question-Metric can help to identify which metric out of the captured data from the software's measurement system is suitable for monitoring and improving quality. [36]

There exist different quality models, dividing quality into quality characteristics or using factors. They all try to be easy to understand and to be used for any kind of software. Section 2.2.1 describes details of three models.

Which quality characteristic is most important and which should be considered depends on the view port. [36]

1.2 Definitions

> *longevity*. 1a: a long duration of individual life. b: length of life. 2: long continuance : permanence, durability[1]

Though this definition also used by *Proenca et al*., they state that "e*ssentially, longevity can be seen as a non-functional quality attribute of an artifact that describes the degree to which the artifact continues to fulfil its purpose for a certain timespan or as long as a defined set of conditions holds.*" [9]

For the use in the context of software, we can find the following definitions

> *What is Software Longevity: The life expectancy of* ***software****, measured by various factors among which is its age.*[2]

Again *Proenca et al*. "*define* ***information longevity*** *as the objective that is met if information artifacts remain fulfilling their intended purpose across time for as long as needed. On the other hand,* ***systems longevity*** *for an information system can be defined as the objective that is met if it is possible to manage the system over time so that it remains fulfilling its intended purpose for as long as needed.*" [9]

The term **Free Software** was introduced by *Richard Stallman* in 1986 and means software that the end user is allowed to use (i.e. run), study, share (copy and distribute) and modify (change and improve). The term "free" is meant as in "free speech", not as "free of charge" (gratis) [37]. This definition known as the Free Software Definition is used by the Free Software Foundation (FSF).

The term Free Software is older than the term **Open Source Software**, which was introduced by *Bruce Perens* for the Open Source Initiative (OSI) stating which software license fulfils the need of the open-source certification of OSI. This definition was based on the Debian Free Software Guidelines of the FSF. So despite of fundamental philosophical differences between the FSF and the OSI there are not so many differences. Most software license fits both definitions. [38]

Additionally to the advantage of Open Source Software that you can look at the source code, typical Open Source Projects also provide more information. Usually the source code repository is accessible, providing not only the source code but also additionally meta data like author, time and comment. Then for most projects there also exist a bug

[1] http://www.merriam-webster.com/dictionary/longevity

[2] http://www.igi-global.com/dictionary/software-longevity/38782

tracking system, a mailing list, maybe an internet relay chat and last but not least the documentation. These data of the software system can be used to calculate metrics.

1.3 Motivation and Problem Definition

Successful software projects need requirements planning [12]. According to *Senyard and Michlmayr* successful Open Source projects are built by a small group with traditional software development and make a transition to a community based development [13]. For successful operation of a project they also define crucial activities, e.g. a prototype, modular design, available and working source code, attracting a community and communication plus other important activities [13]. So they name necessary activities but not success criteria.

Additionally project success criteria differ depending on the stakeholder [14]. The result of a successful software project does not necessarily be a good, used piece of software. A project can be financially successful, but the software is never used. This can be solved by balancing project attributes responsible for project success [15] – additional to activities found in [13]. So with attributes it is possible to meet the interests of both the customer and the company developing the software.

The **motivation** of this thesis is to find out how to measure the longevity of a software project/product. The measurements shall not only cover the produced code, but also the other aspects of a product (help and support, ease of use, features and reliability).

The thesis shall answer, if an information system will still be usable in the future. This aims at saving time at deciding whether to do a project or not and thus also saves resources of unsuccessful projects.

Longevity is a topic that is currently not fully addressed [8]. But it is seen as an important design issue [9].

The **problem area** is the evaluation of a software project: from static code analysis via interpreting support efforts to the reason for a wide usage of a certain software product. This includes all available information about a software project: specifications, documentation, communication and more – not only the source code itself. All these fields provided can be analysed for measuring longevity.

With that given problem area we can ask the following **key questions**:

> What makes a (open source) software project so successful, that its software is used for a long time? How can this longevity be measured beforehand?

Answering these questions should lead to results in compliance with the motivation of the work. The key questions will be used in the Research Issues in chapter 3.

Expected results are quality attributes and metrics, which are used for longevity measurement of a software project. These findings are used to define the requirements for metrics and tool selection. A selected set of tools will be included in an evaluation framework to estimate selected quality attributes, metrics, and key performance indicators with focus on software project longevity. We evaluate the framework by using selected projects based on related work and commonly used successful and less successful open source projects. Main outcome is the evaluation framework and a set of evaluated open source projects to estimate important longevity attributes.

1.4 Content of the Work

The structure of the work uses its key questions described above.

After this introduction including a detailed problem definition and an overview of the content of the work the next chapter 2 gives an overview of the Related Work useful for this thesis. Relevant topics are Process Models in section 2.1, Quality Models in section 2.2 and other Frameworks for Measuring OSS Projects in section 2.3.

Chapter 3 defines the Research Issues based on the already given key questions in section 1.3 in this chapter. How the research issues are handled is described with the Solution Approach in chapter 4, which lay out the concept of the work in detail.

In chapter 5 the first part of this work describes the Systematic Literature Review on Quality Attributes, Metrics and Tools. The Tool as second work part is documented in chapter 6. The third part of the work is the creation of the Framework for Longevity Evaluation, which is described in chapter 7. The evaluation of the work is split into chapter 8 containing the Feasibility Study based on Open Source Software and thus containing an introduction in chapter 8.1 and chapter 9 containing a Case Study of the proposed framework.

The Discussion of the results in chapter 10 contains also an outlook and potential next steps that leads to chapter 11 with the Conclusions summing up the complete work.

2 Related Work

Longevity is a topic that is currently not fully addressed [8], but it is seen as an important design issue [9]. The longevity framework presented in this work wants to measure longevity. This can be done by using project success criteria [14] and project attributes [15].

Ongoing work show that the quality attributes of ISO SQuaRE [1] can be brought in relation with the four topics of information longevity [8]. The longevity framework uses that relation.

Another way of measuring project success is found in the work of *Wahyudin* et al. who introduced "health indicators" to monitor the status of open source web engineering projects [10]. Based on this work *Sunindyo* et al. provide a framework for analysing OSS project health with heterogeneous data sources and evaluate the framework for a set of different OSS projects [11].

Important parts of the theoretical background are process models and quality models as they set the basis and environment for every software project.

The longevity framework is evaluated using related work on health indicators for measuring OSS projects Apache HTTPD and Apache Tomcat.

In related work also other frameworks for measuring OSS projects are discussed.

2.1 Process Models

A central tool in software product development is the software development process. Each company has its own practices, but usually the development follows a software process model.

Let us define the term process model first. To give a short definition from literature: a process models is "*a simplified representation of software process, presented from a specific perspective*." [31]

"*A process model determines the sequence of phases and milestones in a project. According to the sequence can be distinguished between sequential and iterative process models. In sequential process models is a phase (which is in such models usually the same as a production step) run through once. The start and end of each phase are defined by milestones. In iterative process models are phases run through several times, to gain a higher product maturity*." [25[3]]

[3] Translated from German by the author

Process models are important for quality and thus for longevity, as they provide guidelines, concrete instructions, sample documents and checklists helping the developer to produce high-quality and well-documented code.

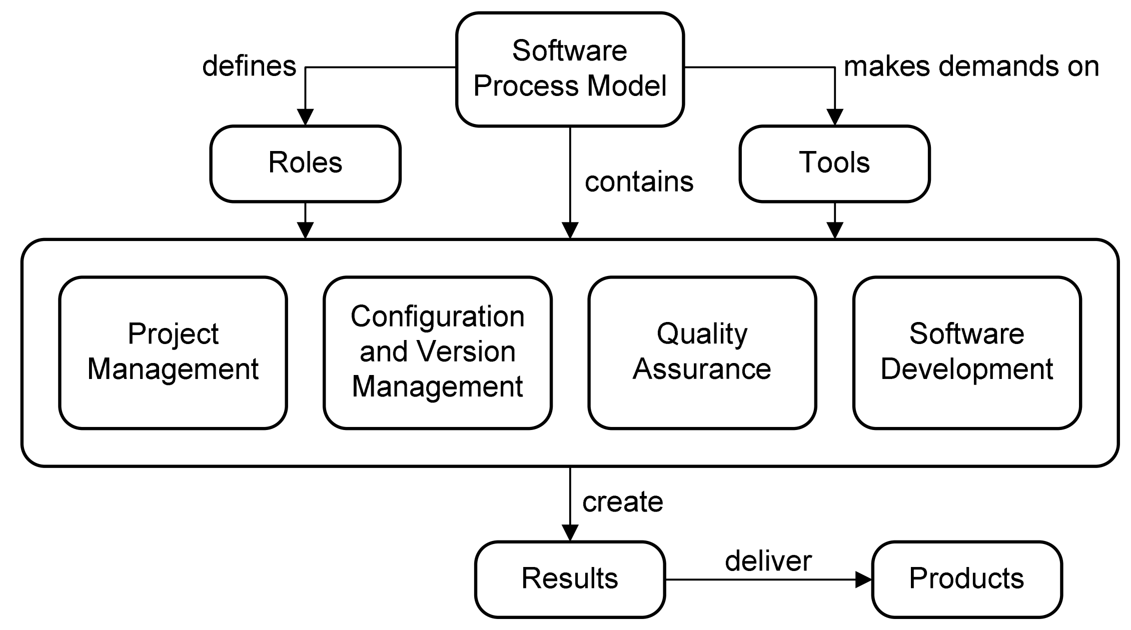
We notice a first classification into sequential (i.e. linear) and iterative (i.e. cyclic) process model types here. This classification is extended by light weight process models.

According to *Elting and Huber* software process models can be classified, how far they cover the full scope and how detailed they support the processes within the project. To distinguish they suggest the following questions:

- Supports the process model hints to adapt it to concrete projects (tailoring)?
- What is all part of the process model?
- What are the project results of the process model?
- Does the Process Model contain components (e.g. report templates) to document the results?
- How detailed is the support of the process model (e.g. estimation methods)?

Considering this questions process models cover a field that reaches on the one hand from a simple software development support to an universal model and on the other hand from a "*prescription of a rough behaviour pattern*" to "*exact rules*" [32]. The position and a possible content of a software process model are shown in Figure 1.

Figure 1 Position of Process Models in Software Development [32]

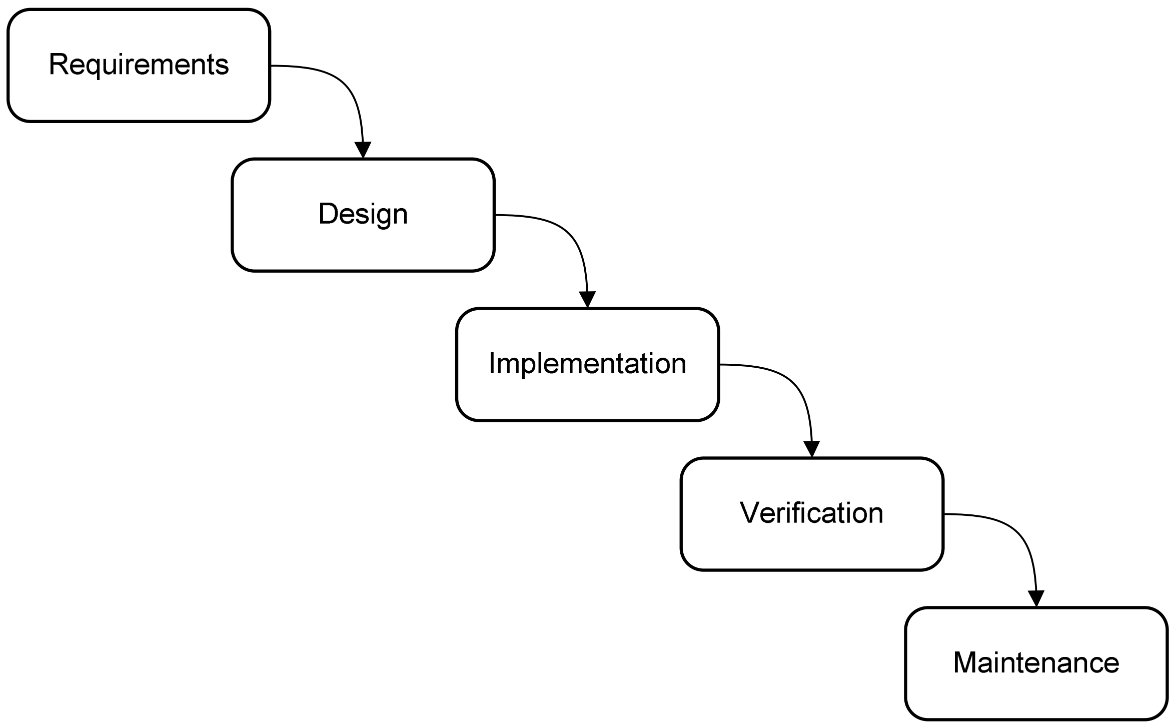


The reason why process models are of special interest is simple. They provide another form of knowledge and thus lead to improvement. While light weight process models (e.g. SCRUM) just define responsibilities (roles), other (e.g. the V-Model and V-Modell XT) define who has to do what and when. So a process model description also often contains guidelines, concrete instructions, sample documents and checklists. By continuous adaption of this assistance improvement is gained.

2.1.1 Linear Process Models

Linear Process Models are sequential process models. They have in common that it's not possible to step back from a later phase to a previous phase. A popular example is the **waterfall model** as shown in Figure 2.

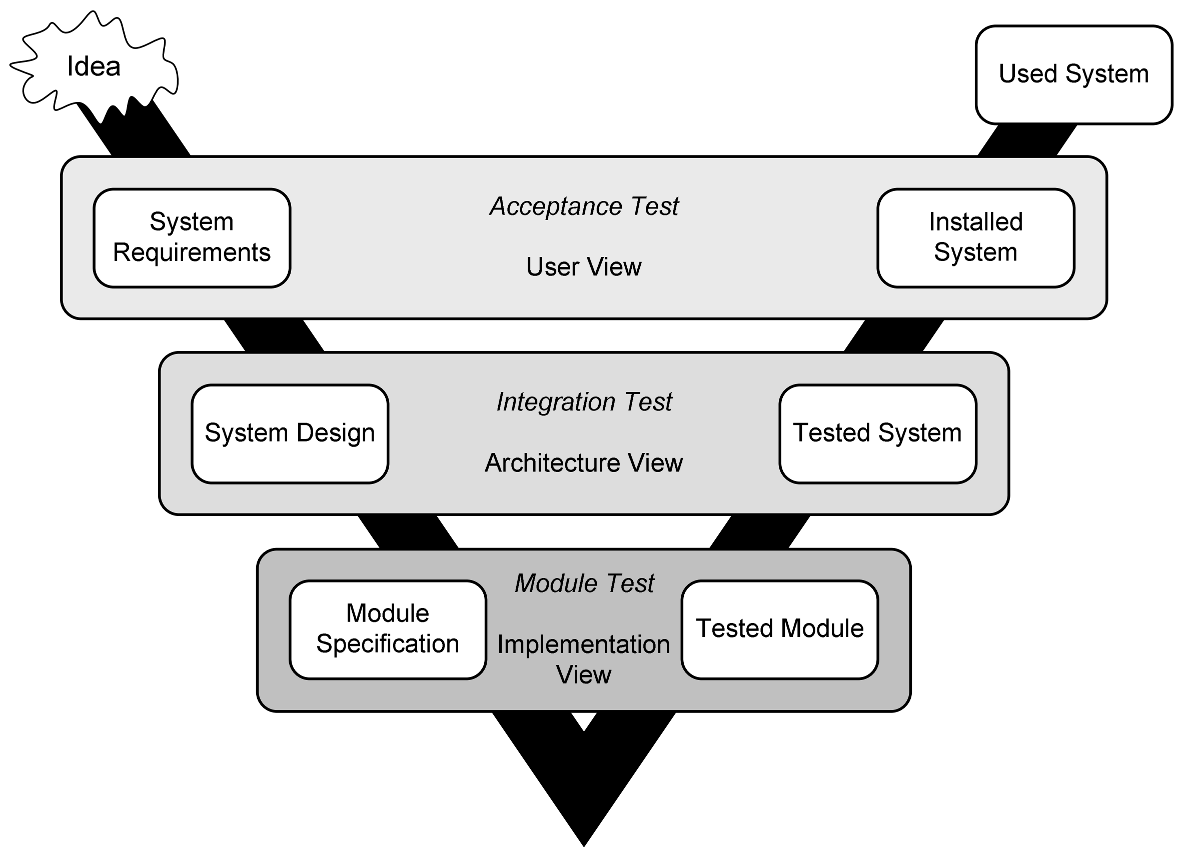
Figure 2 Waterfall model [24]



Both the **V-Model** and its successor the **V-Modell XT** are not strictly sequential process models, because they do not force an inspection and approval at the end of each phase and because a step back is possible (though not initially desired or even planned).

The V-Model can be seen as an extension to the waterfall model by testing after each phase as shown in Figure 3. The phases after the implementation are bent upwards to form a V shape. The tests can also be designed at the start of each phase.

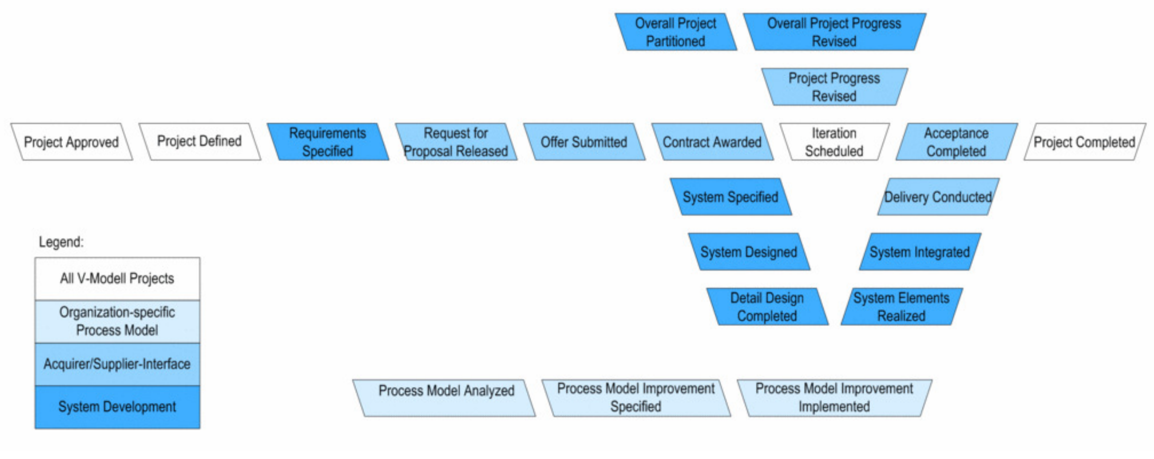
Figure 3 V-Model [25]



A special variant of the V-Model approach is the German V-Modell 97. The V-Modell 97 is the old installation standard of the V-Model concept. It was replaced in February 2005 with the first release of the V-Modell XT. Both versions are a development standard for IT system development of federal agencies of the Federal Republic of Germany. The V-Modell 97 is not only a software development model, but also accounts for the areas project management, quality management and configuration management. Therefore it contains the four sub models system development, quality assurance, configuration management and project management. Each of these sub models are designed in the V shape as shown in Figure 3. The real development activities are done in the sub model system development, which itself distinguishes between software and hardware development and is described from a functional point of view. The model describes who has to do what and when, but the focus lies on the activities and their completion.

In the V-Modell XT the completion of a work product (or formally: a reached project progress stage) is called a decision gate. Figure 4 shows possible decision gates in the known V shape coloured in project execution strategies. It also shows nicely the extension of the V-Modell 97 in all directions of the V shape. A project execution strategy suits the project type (one of system development project (acquirer), system development project (supplier), system development project (acquirer/supplier) - acquirer and supplier within the same organization (without contract) or introduction and maintenance of an organization-specific process model) and shows among the new integration of the acquirer also the possibility for tailoring in the V-Modell XT.

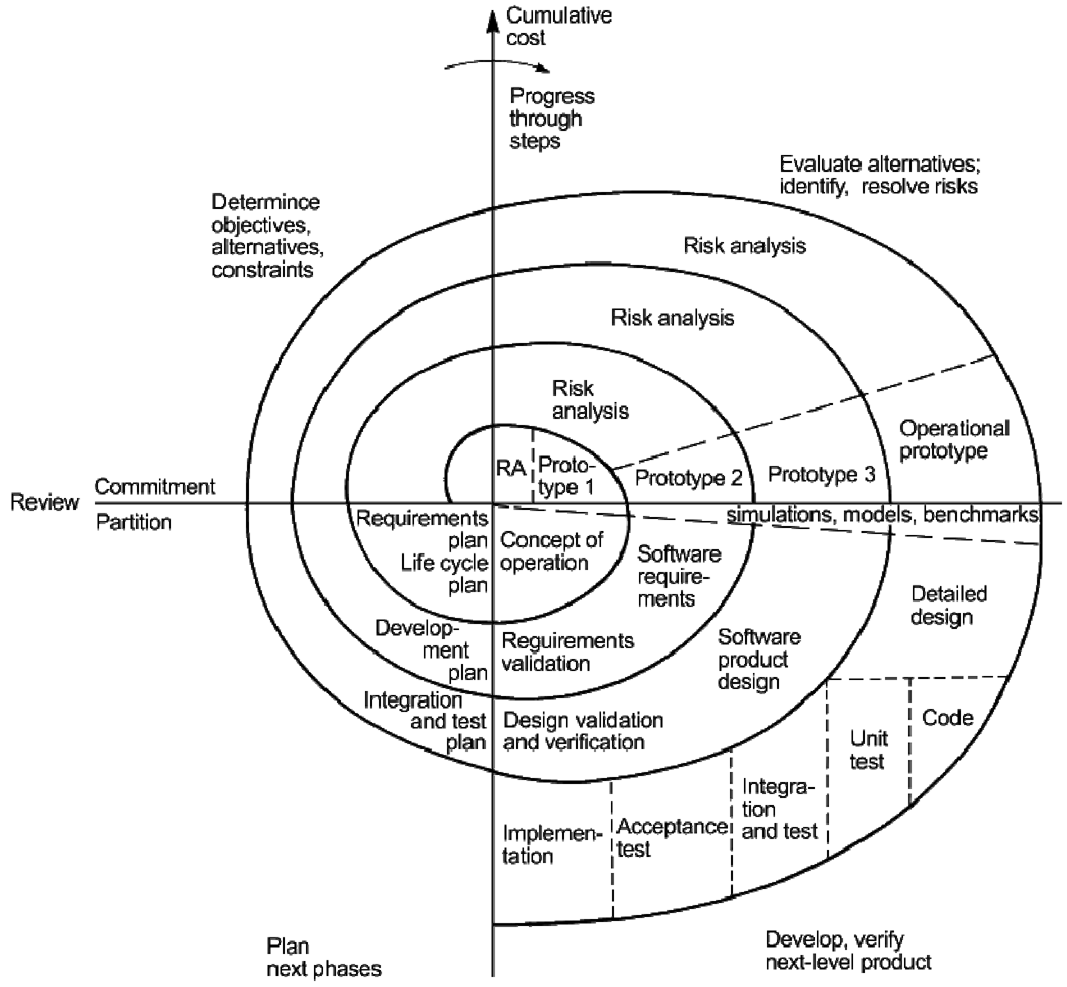
Figure 4 Decision gates of the V-Modell XT [26]



2.1.2 Cyclic Process Models

In response to the weaknesses of the waterfall model, iterative and incremental software development processes emerged. The work of a phase is split into iterations, which usually have the same structural design. The **spiral model** by *Boehm* [28] shown in figure 5 is such an iterative model.

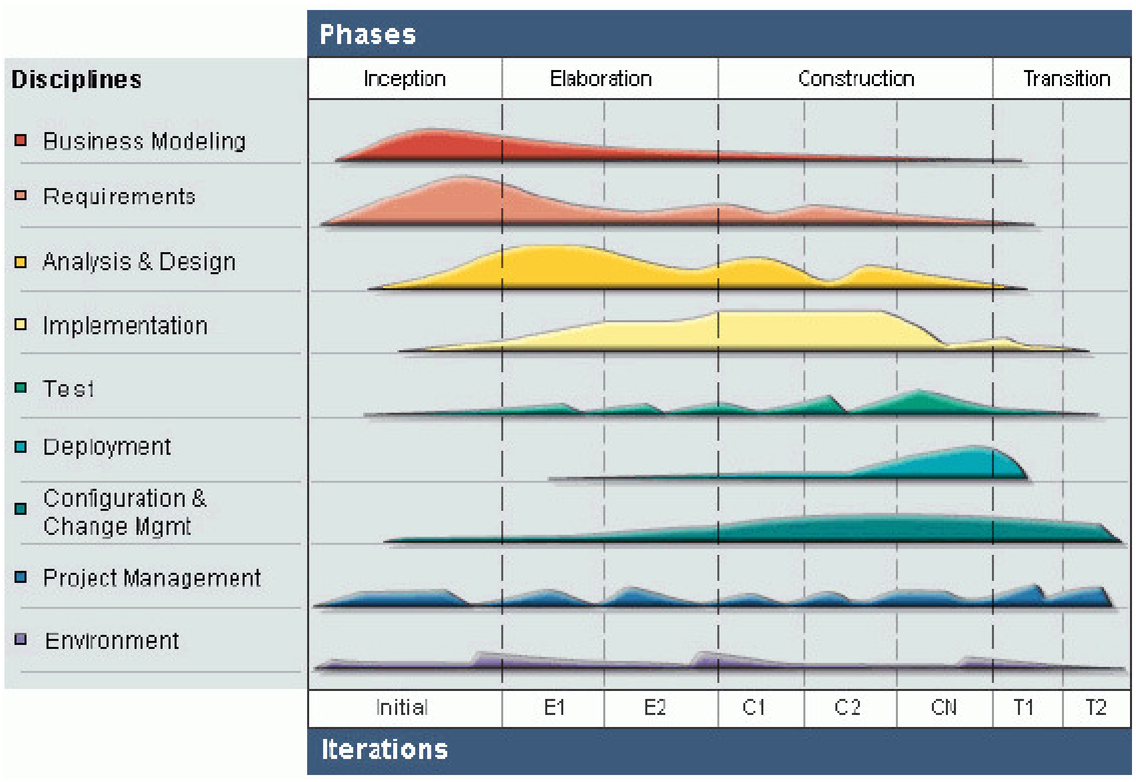
Figure 5 Spiral model [28]



This incremental development is also an essential part of the Rational Unified Process and in agile software development. Such process models for quick and not document orientated software development often used for smaller projects are discussed in the next section 2.1.3.

The spiral model was introduced in 1986 by Barry Boehm. It is not the first iterative approach, but the first to combine elements of waterfall model and prototyping. It was designed for big projects with an iteration lasting between 6 and 24 months. "*It incorporates many of the strengths of other models, while resolving many of their difficulties.*" [28] One major new feature is the risk analysis, but it lacks support for contracts, risk evaluation and a mechanism for a consistent work context. The number of iterations is also fixed, but the spiral model is very helpful to understand the Rational Unified Process.

Figure 6 Rational Unified Process [29]



The **Rational Unified Process** (short RUP) is a framework created by Rational Software, which is now a division of IBM. They evolved an iterative software development process framework of best practice approaches. Figure 6 shows the effort for the disciplines at any iteration. RUP provides a tailorable process that guides development with detailed description of activities and also sample artifacts. Additionally tools help to automate the application of the process and to adapt the process and tools to own needs.

A more detailed description is provided at the Rational Unified Process itself. It contains also a list of four "highlights". [29]

2.1.3 Light Weight Process Models

The representatives of linear and cyclic process models focus strongly on process and documentation. Compared to these heavy weight process models agile software development tries to reduce overhead like e.g. unnecessary analysis or documentation and focuses on the development and interaction. The critique of the heavy weight process models, described in previous sections 2.1.1 and 2.1.2, summoned in the "Manifesto for Agile Software Development":

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

Individuals and interactions *over processes and tools*

Working software *over comprehensive documentation*

Customer collaboration *over contract negotiation*

Responding to change *over following a plan*

That is, while there is value in the items on the right, we value the items on the left more." [33]

Agile approaches to software development are a later idea of handling software development and thus fairly new. The beginning of the agile paradigm marked the introduction of Extreme Programming by *Beck* [34]. With the following software development process models using an agile approach like Agile Software Development (including Agile Modeling and Agile Database Techniques), Crystal, Dynamic Systems Development Method (DSDM), Extreme Programming (XP), Feature Driven Development (FDD), Lean Software Development (LD), Rapid Application Development (RAD), Scrum and last but not least Test Driven Development (TDD) the agile paradigm is no longer a hype or new trend. This work discusses just a few of this plenty of available models, namely Extreme Programming and Test Driven Development.

Extreme Programming (XP) is an agile method for software development. It needs a great deal of discipline during execution and is used most often for small and medium sized projects. This approach to software developments uses stories defining the requirements for short iteration cycles. The code will be written using Pair Programming, (automated) tests performed all the time and with as few as necessary documentation. Another important distinction from linear process models is that architectural changes are welcomed and shall be performed using refactoring as adjustments are necessary for a new version [34].

Test Driven Development (TDD) is very similar to XP. It may also use stories and the technique Pair Programming; but the principle of TDD is to write an automated test first. This way or work does not conflict with XP at all. Focusing on tests, larger systems are designed from the start for testability by creating small, testable and preferable louse-coupled code units.

A good testable code, which should always be the result of TDD, has a higher quality code changes that produce errors should be impossible, if the tests covered all cases. A software containing such code and especially the knowledge of such completely test covered code raises the longevity of a software.

2.2 Quality Models

Process models are supported by quality models and organisational processes.

2.2.1 Quality Assurance Models

A lot of quality models for various purposes exist.

One important quality model is the **ISO 9000** family of quality management systems, which focuses on quality in organizations to meet the needs of all stakeholders while meeting regulatory requirements relating to a product [49]. It can be used quite universal.

The program **Capability Maturity Model Integration** (CMMI) is a standard focusing more on software development [47]. The CMMI defines the following maturity levels for development: Initial, Managed, Defined, Quantitatively Managed and Optimizing. Usually like with ISO 9000 a company has a certification for reaching a certain CMMI level [47].

Other quality models put more a focus on process improvement like ISO/IEC 15504, also known as **Software Process Improvement and Capability dEtermination** (**SPICE**). It is the reference model for maturity models similar to CMMI and aims on assessing an organization's capability [48].

For this thesis we are more interested in the quality of a software product during the whole engineering process.

Kitchenham and Pfleeger describe three quality models for software engineering: "One of the earliest quality models was suggested by Jim McCall and colleagues. […] The model defines software product qualities as a hierarchy of factors, criteria, and metrics." Factors are influenced by multiple criteria. "A quality factor represents a behavioral characteristic of the system. A quality criterion is an attribute of a quality factor that is related to software production and design. A quality metric is a measure that captures some aspect of a quality criterion. Thus, the 11 quality factors contribute to a complete picture of software quality. One or more quality metric should be associated with each criterion." [36]

So the **McCall quality model** was the first influential model [30]. We will find portions of this model further on in later published quality models. Also notable is the connection to metrics shown in Figure 7.

Figure 7 McCall quality model [30]

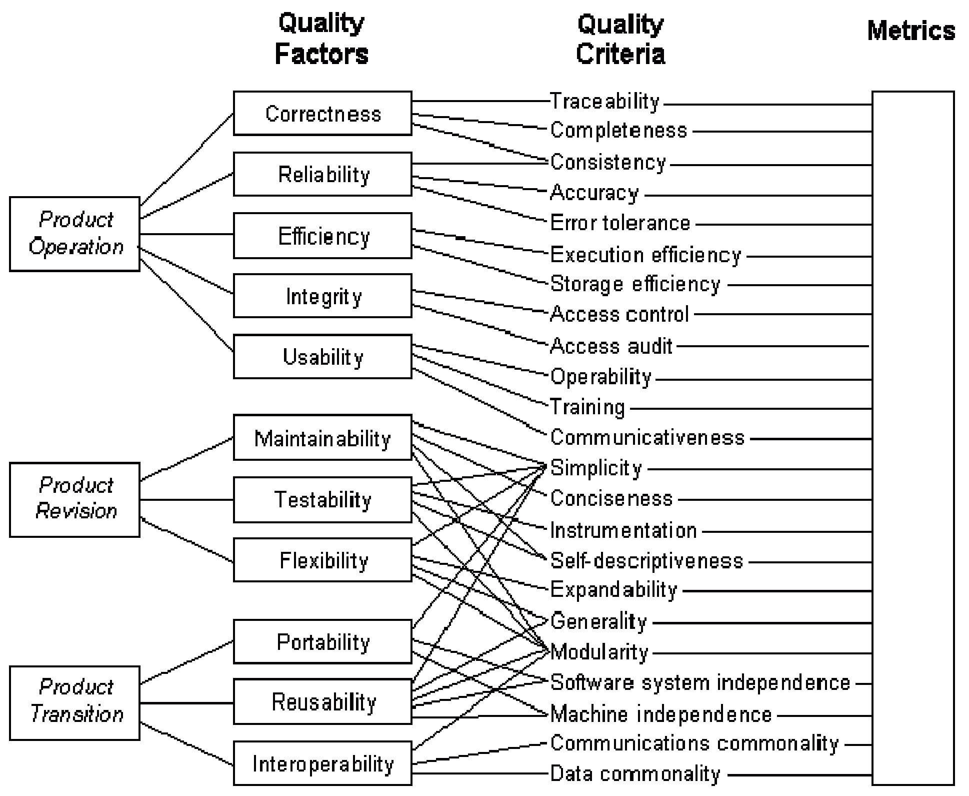


Figure 7 also lists nicely all quality factors and quality criteria of the McCall quality model.

The second quality model mentioned by Kitchenham and Pfleeger is the **ISO 9126 quality model**. It is special part of the ISO 9000 family addressing software quality. Instead of quality factor the term quality characteristic is used. "The standard recommends measuring the characteristics directly, but does not indicate clearly how to do it. Rather, the standard suggests that if the characteristic cannot be measured directly (particularly during development), some other related attribute should be measured as a surrogate to predict the required characteristic. However, no guidelines for establishing a good prediction system are provided." [36]

The ISO 9126 quality model is in the meantime out of date and was replaced by ISO/IEC 25000 SQuaRE, described in section 4.1 within the identification of quality attributes in the solution approach.

Besides all similarities of the McCall quality model and the ISO 9126 quality model there are several noteworthy differences: First the ISO 9126 model uses a different quality framework and terminology [36]. Second, as mentioned before, it uses the term "quality characteristics" instead of quality factor. Third, it also uses sub-characteristics to refine characteristics and most noteworthy the ISO 9126 model is completely hierarchical. So compared to the McCall quality model in Figure 7, which for example links the quality criteria Generality to both quality factors Flexibility and Reusability, for the ISO 9126 model an ambiguity of sub-characteristics to quality attributes is not possible. “Also, the sub-characteristics relate to quality aspects that are visible to the user, rather than to internal software properties. Thus, the IS0 model reflects more of a user view, while the McCall model reflects more of a product view.” [36]

Both the McCall and the ISO model share common problems: They lack a rationale for a choosing their factors / characteristics and also for the division of sub-characteristics to main-characteristics. This makes it impossible to check the model for completeness or consistent definition [36]. Additionally there is no description of the metrics (or indicators called in the ISO 9126 model). “In particular [...] there is no means for verifying that the chosen metrics affect the observed behavior of a factor. That is, there is no attempt to measure factors at the top of the hierarchy, so the model is untestable.” [36] The problem of main-characteristics not being testable still also accounts to ISO SQuaRE, but later research showed that metrics can be linked to quality attributes [46]. This is important for this work, because we need a linkage.

The third quality model mentioned by Kitchenham and Pfleeger is an approach by **Geoff Dromey**:

“Geoff Dromey has developed a model that addresses many of these problems. [...] Dromey believes that it is impossible to build highlevel quality attributes such as reliability or maintainability into products. Rather, software engineers must build components that exhibit a consistent, harmonious, and complete set of product properties that result in the manifestations of quality attributes. […] Dromey’s approach is important because it allows us to verify models. It establishes a criterion for including a particular software property in a model (that is, that a quality defect can be associated with the concept) and a means of establishing when the model is incomplete (that the model cannot classify a specific defect).” [36]

An overview of the quality factors / main characteristics used in the models (high level quality attributes) is given by Suman and Wadhwa [45], who also had a look at other quality models. Efficiency, Maintainability and Usability are present in almost all quality models. Not that their comparison does not contain all quality attributes present in a quality model, e.g. as sub-characteristics in ISO SQuaRE.

It is also important to look at the business value of quality: maybe a not fully working software product is also good enough.

We like to end the definition of software quality keeping in mind the non-definition of *Kitchenham and Pfleeger:*

> "Quality is a complex concept. Because it means different things to different people, it is highly context-dependent. Just as there is no automobile to satisfy everyone's needs, so too there is no universal definition of quality." [36]

2.2.2 Health Indicators

Originally the term health indicator comes from pulbic heath care. In the use of software development health indicators are special attributes of quality, that represent the state of a software project.

A way of measuring project success is found in the work of *Wahyudin* et al. who introduced "health indicators" to monitor the status of open source web engineering projects [10].

Based on this work *Sunindyo* et al. provide a framework for analysing OSS project health with heterogeneous data sources and evaluate the framework for a set of different OSS projects [11].

The SLR for creating the longevity evaluation framework uses this term to search for successful software projects.

The longevity framework is evaluated using the following related work on measuring OSS projects:

2.2.2.1 Apache HTTPD

The Apache HTTP Server project is an open source web server started in 1995. It is C-based and provides web pages, with plug-ins also pages written PHP and much more. The software is licensed under the Apache License and runs on almost all platforms.

As an open source project it is developed and supported by an open community under the lead of the Apache Software Foundation.

Apache can really be named a user-driven development, as the developers of the system are also its biggest users. The extensible design and modular API fits the needs of various users. [20]

All communication is available to the public. With a huge community most questions are answered by other users. If a bug is found in the software, a publically available patch to fix the problem is usually generated within a week. [20]

According to Open Hub[4] Apache HTTP Server is a very highly active, well established project with a very large development team (30 developers with commits in the last year).

Apache HTTPD is a good analysed healthy OSS project with different data sets from different authors of metrics by [2], [3], [4] and [18].

2.2.2.2 Apache Tomcat

The Apache Tomcat project is an open source web server started in 1999. It is a Java-based web application container that runs Java web applications: Servlet and JavaServer Pages (JSP). Originally Apache Tomcat was created as a subproject of Apache named Apache-Jakarta, but due to its popularity, it is now hosted as a separate Apache project. As an open source project it is supported and enhanced by volunteers from the open source community. [21]

The web server is very stable and has all features of a commercial web application container. It also has some extra features that extend servlet container (named Catalina), for example a management application [21].

Major versions on Apache Tomcat go along with versions of the Java Servlet specification (Java Servlet API). The actual version 8.0 supports the Servlet API 3.1 and the corresponding JSP 2.3 version.

According to Open Hub[5] Apache Tomcat is a very highly active, well established project with a large development team (10 developers with commits in the last year).

Apache Tomcat is a similarly good analysed OSS project [2], [6] and [7].

2.3 Frameworks for Measuring OSS Projects

A directly related work to the longevity framework measuring industry and OSS projects are the frameworks “evolizer” and “changedistiller” [5] as well as “Alitheia Core” [42], “OSSMeter” [43] and the Open Engineering Service Bus (OpenEngSB) [44]. They will be used for comparison with the proposed longevity framework and we want to give an overview here.

Evolizer [5] is an eclipse plug-in for software evolution analysis. The implementation uses Hibernate to build a meta-model of the analyzed software. The current implementation provides support for importing and representing data from the version-control systems CVS and Subversion, the bug tracking system Bugzilla, Java source code and integrates

[4] https://www.openhub.net/p/apache/

[5] https://www.openhub.net/p/tomcat

the meta-models of it. The development seems to have stopped and the project website is no longer online, but at least the source code of the project can be found on bitbucket[6].

For analysis the ChangeDistiller-algorithm seems to be used. It extracts fine-grained source code changes between subsequent revisions of Java classes. The algorithm does tree differencing and works on abstract syntax trees. Although the research project creating ChangeDistiller has stopped, the project can be found on bitbucket[7], but it does not show much activity.

Alitheia-Core [42] is a platform which aims at enabling software engineering research targeting OSS projects. Alitheia-Core provides support for processing source code repositories through an API. Alitheia-Core is designed to use OSGi. The platform has just a few metrics implemented and lacks metrics related to mailing list and bug tracking systems. It seems to be abandoned, as the website does not work and also the code found on GitHub[8] does not show activities.

OSSMeter [43] is a cloud-based platform that "extends the scope and effectiveness of OSS analysis and measurement with novel contributions on language-agnostic and language-specific methods for source code analysis, but also proposes using state-of-the-art Natural Language Processing (NLP) and text mining techniques such as question/answer extraction, sentiment analysis and thread clustering to analyse and integrate relevant information extracted from communication channels (newsgroups, forums, mailing lists), and bug tracking systems supporting OSS projects, in order to provide a more comprehensive picture of the quality indicators of OSS projects, and facilitate better evidence-based decision making and monitoring. OSSMETER also provides metamodels for capturing the meta-information relevant to OSS projects, and effective quality indicators, in a rigorous and consistent manner that enable direct comparison between OSS projects." [43] All these features sound very interesting, the website is running, but holds very few information about the project and announces a beta testing phase since February 2015. It looks like another EU-sponsored research program that is abandoned after the project end, as the code can be found on GitHub[9] and does not show activity either.

---

[6] https://bitbucket.org/sealuzh/tools-evolizer/

[7] https://bitbucket.org/sealuzh/tools-changedistiller/

[8] https://github.com/istlab/Alitheia-Core

[9] https://github.com/ossmeter/ossmeter

The **Open Engineering Service Bus** (OpenEngSB) [M38], [44] is a platform for tool integration. It puts more focus on a graphical workflow between the supported tools Mail, Twitter, Facebook, JIRA, Git, GitHub, Prom, Trac and maven. It is a well documented, modular system based on OSGi, that can be controlled via ssh console using Apache Karaf. In contrast to typical Enterprise Service Bus approaches existing workflows can easily be adapted to software changes due to the domain-based integration model. Thus, OpenEngSB enables transparency in dynamic development processes too and frees the engineers' time for productive work. Like with the other related projects, on the website openengsb.org, JIRA[10] and in the code[11] no activity is seen for more than a half year.

To sum it up: all similar frameworks for measuring OSS projects seem to be dead. The only active project is FLOSSmole[12] [M22], a tool only for data retrieval, seems hard to interact with. Though source code is available on GitHub[13], it does not look like an easily extendable framework.

[10] http://issues.openengsb.org/

[11] https://github.com/openengsb/openengsb

[12] http://flossmole.org/

[13] https://github.com/FLOSSmole

3 Research Issues

The key questions defined in section 1.3 of the introduction give the inspiration of this work: We are interested in success factors of software projects that have an influence on people using the software for a long time. Optimally, we want to measure software product longevity right from the start of a project, not in the end when all work is done.

That summarizes what this work should answer, but defined in more detail it specifies the research issues of this work:

RI.1: What is the longevity (project health) of a software project?

If one really knows about the health status of a software project, it would be easy to decide whether to use it or search for an alternative project. The longevity of a project also depends like the health status on the quality of the project, but it additionally focuses on the transition of project quality though changed perception of it in the future [9].

The knowledge about project health is highly critical for decision makers, because they have a bigger chance of coming to a decision. Also project managers should be interested in improving the longevity of a project, because it makes the product more attractive and thus valuable.

Without a detailed analysis a health status or even a longevity outlook will just be a rough estimation. The other research issues aim to reduce this uncertainty.

RI.2: What are the quality attributes (health indicators) of software?

The search for project success criteria [14] begins with identifying the relating project attributes [15]. To find out what makes a software project successful, it is necessary to identify what is different to other projects.

This step is necessary for quality assurance and academic researchers, as it provides the theoretical background for comparing software projects in a systematic way and also provides the link to a quality model.

RI.3: What is the quality of the code produced in a software project?

There exist multiple ways of assessing the correctness and quality of software: various types of tests, code reviews, inspections etc. [25]

The answer to this research issue is one of the main questions of quality assurance, if no static code analysis tool is used. Using a static code analysis tool this issue can be answered calculating metrics like the percentage of all code lines covered by well written tests.

Code quality itself will vary in the eye of the beholder, but using tools should give comparable results. The numbers calculated are interesting most of all for quality managers but also for developers, because they provide insight/feedback that would have to be guessed otherwise.

RI.4: How to measure important quality attributes (metrics) in context of longevity/project health?

We need metrics for measuring the longevity attributes.

Assigning numbers and symbols of project attributes rely on internal attributes (i.e. measurements) that can be computed directly [15]. These computed measurements are also called software metrics; in this work just metrics.

The resulting numbers of the metrics without context are just an intermediate result that is not much of use. But put into relation with previous runs on older versions of the software project or other software projects, a quality manager can make interesting findings.

RI.5: Which tools help measuring software longevity?

As the prediction of software longevity/project health is needed from the start, an objective opinion can be only calculated using tools.

The automatic and tool-supported use of metrics on a software project is of most interest for the quality assurance, but the numbers are even more helpful for software developers, as they get some reliable and even more important comparable numbers of their code.

RI.6: How can tools be combined to a usable framework?

Each tool has its strengths in calculating certain metrics. A combination of tools leads to a variety of independently calculated metrics and the comparison of the results helps to verify the metrics.

The framework for longevity is the goal of this work and should be interesting most of all for decision makers (investors), but also for project managers, quality managers or developers, because of all the advantages named in the above research issues and because a framework combines many metrics and tools making it even more fail-proof. To know an indication about the project health and thus the estimated longevity of the project makes planning and decision making easier.

The research gap is mainly the systematic focus on software longevity, resulting in a direct link of quality criteria via metrics to tools of a framework.

The research issues are covered by the results of the work.

4 Solution Approach

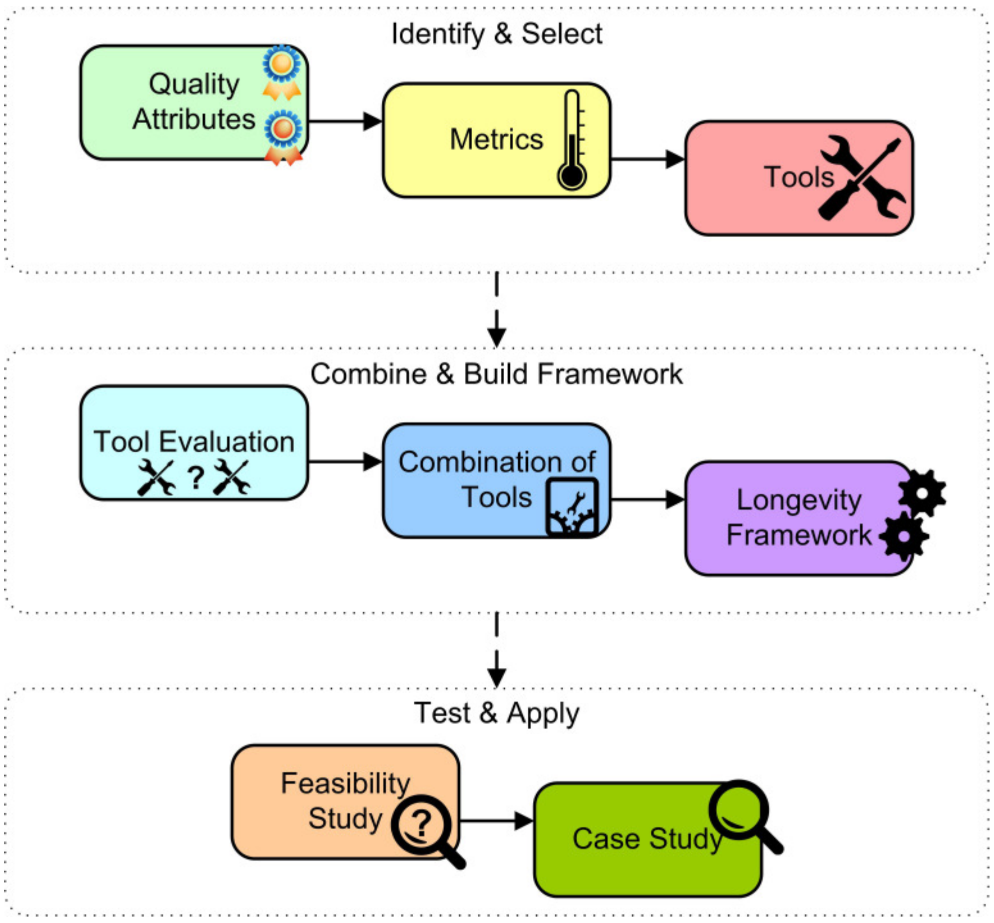
This chapter is about the profound methodological approach for measuring the longevity of a software project/product.

Based on the research issues (see previous chapter 3), the tasks for establishing a framework evaluating longevity of a software project are:

- Design a process for software project evaluation with respect to systems longevity
- Identify quality criteria (project attributes) to assess project and product longevity and help identify potential risks
- Identify metrics for measuring for these quality criteria
- Identify tools for calculating these metrics
- Aggregate these tools to a framework: integrate selected tools, i.e. making them work together)
- Test the framework
- Find well analyzed Open Source Software (OSS) projects, run the framework on them too and compare the results

These tasks represent the content of the work and are visualized in Figure 8.

Figure 8 Content of the Work



As shown in Figure 8 above, the tasks are grouped to three parts covering the research issues:

1. "Identify & Select": Analysis and identification of software quality attributes for longevity

   Based on software quality attributes defined by ISO SQuaRE [1] we want to know what makes a (open source) software project successful. We identify success criteria from literature by doing a systematic literature research (SLR). During that SLR we also identify metrics and tools and select those usable for longevity.

   So this part of the work addresses research issues 1, 2 and 3.

   The results of this part are a set of quality criteria that assess project and product longevity and a set of metrics and tools to measure the longevity using quality criteria.

2. "Combine & Build Framework": Aggregate and select the identified tools and build a framework for quality evaluation

Additional to the identified quality attributes we need a process supported by tools.

This part of the work designs a process for software project evaluation with respect to systems longevity. Based on the identified metrics for the quality attributes and tools for measuring these metrics, we integrate selected tools into a framework (i.e. making them work together).

The second part of the work addresses research issue 4 concerning the selection of tools and describes all steps of building the framework.

The result of this part is a framework (set of tools) that is capable of predicting the longevity or risk of a project.

3. “Test & Apply”: Evaluation of the quality evaluation framework

   We find out, if the built framework is valid and predicts correct results.

   Therefore we initially evaluate the framework (Feasibility Study) and use well analyzed Open Source Software (OSS) projects, run the framework on them too and compare the results (Case Study).

   The result of this part is a test of the framework by comparing with previous results from related work (“learning from the past”) and by applying to some software projects (“predicting the future”).

After describing the used quality model, this chapter describes the concept of the framework plus the concept of its evaluation.

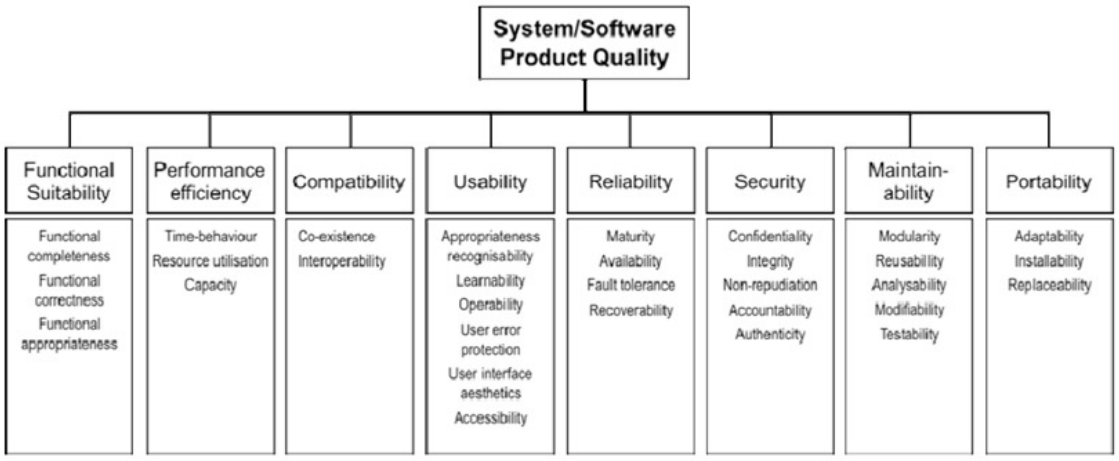
4.1 Identification of quality attributes

For estimating longevity of a software project we need to show which quality criteria are related. This fully answers **RI.2** asking for quality attributes of successful software [14], but it also sets the basis for answering **RI.1** about the longevity of a software project.

The quality criteria for designing the evaluation of the process are taken from ISO SQuaRE (the ISO/IEC 25010 Standard from 2011) [1]. The longevity framework uses the System/Software Product Quality. This is an overview of the concepts the framework for measuring project and product longevity is built on.

ISO Square is a standard for System and Software Quality Requirements. It defines software quality using eight main characteristics and 31 sub-characteristics shown in Figure 9:

Figure 9 ISO/IEC SQuaRE – System/Software Product Quality [1]



ISO SQuaRE is the successor of the ISO 9126 standard from 1992 [39], which did not contain the characteristics Compatibility and Security. Also some sub-characteristics changed; for longevity for example relevant is the new sub-characteristic Modularity that was added to Maintainability [45]. ISO 9126 itself derives of the McCall model [30]. Due to limitations of ISO 9126 it was revised and though ISO SQuaRE now also "defines a framework to specify and evaluate software quality" [22], direct measuring of the attributes is still not easy.

The quality attributes of ISO SQuaRE provide the division we use for linking metrics and tools to quality. Thus the main characteristics and their sub-characteristics are enumerated here. Instead of the term sub-characteristics we use the term quality attribute. For every quality attribute ISO SQuaRE gives a definition, like the cited definition of the main characteristics of System/Software product quality [1].

Functional Suitability is the "degree to which the software product or system provides functions that meet stated and implied needs when used under specified conditions" [1] and divides into: functional completeness, functional correctness and functional appropriateness. So this is mainly about functional requirements. A long-living software should comply with at least most if not all of the users' feature requests to it, resulting in the requirements of the software.

Performance efficiency is defined as "performance relative to the amount of resources used under stated conditions" [1] and contains the quality attributes time behaviour, resource utilization and capacity.

Compatibility according to ISO SQuaRE is defined as “degree to which a product, system or component can exchange information with other products, systems or components and/or perform its required functions, while sharing the same hardware and software environment” [1]. It is divided into co-existence and interoperability.

Usability, defined as “degree to which a product or system can be used by specified users to achieve goals with effectiveness, efficiency and satisfaction in a specified context of use” contains the quality attributes appropriateness recognisability, learnability, operability, user error protection, user interface aesthetics and accessibility. Especially the use interface aesthetics are not all easy to measure.

Reliability is sub-divided into maturity, availability, fault tolerance, software faults and recoverability.

Security lists confidentiality, integrity, non-repudiation, accountability and authenticity as its noteworthy quality attributes.

Maintainability, defined by ISO SQuaRE as “degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers” [1], is sub-divided into: modularity, reusability, analysability, modifiability and testability. These quality attributes are most relevant for software longevity. [8]

Portability is defined as “degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another” [1]. It is sub-divided into adaptability, installability and replaceability. Besides of installability these quality attributes are very important for longevity.

The comprehensive categorization of quality attributes into these main characteristics makes it easier to focus on a related issue and avoids overlooking other important issues [22].

4.2 Concept of the Framework for Measuring Project Longevity

The methodological approach for creating a creating a framework to evaluate project longevity consists of three parts:

1. SLR on project health
2. Tool selection process
3. Combination of selected tools

A **systematic literature research** (SLR) will find software projects that are successful and thus showed a good health. We plan to analyze the papers found in the SLR further to give an overview of the examined (open source) projects. During that scanning we also note the metrics and tools used.

The **tools selection process** takes all tools found in the SLR, adds commercially available tools for analyzing software or calculating metrics and finds the most suitable tools for building an evaluation framework.

The **evaluation framework** is planned as an aggregation of all available metrics via combining all selected tools into a framework. It combines the selected tools to a framework and uses metrics, which are available in at least two tools and have a relation to a quality attribute. The goal of the framework is simply to provide metrics associated to quality attributes from ISO SQuaRE.

We plan to find out the quality attribute addressed by a metric by comparison with the quality attribute definition of ISO SQuaRE. First by matching the metrics definition and the definition of its measures, second by matching the results to ISO SQuaRE using it’s distinction of main and sub characteristics. Moreover we can also match the metrics by the way the author of a paper mentioning the metrics sees it.

Each metrics used in the evaluation framework will list its linked quality attributes. This linking is based on identified quality attributes from ISO SQuaRE in literature [46].

As we know from the SLR which metric is used to describe healthy/longevity projects, we can make a list of quality attributes affecting that metric and thus can answer **RI.2** that wants to know which quality criteria are related to the longevity of a software project. How to find out which quality attributes are most important for longevity shows an in-depth analysis of Information System Longevity aspects [46].

4.3 Concept for Evaluation of the Framework

The methodological approach for testing the framework consists of a feasibility study and a case study.

The **Feasibility Study** shall include:

1. Testing/Training the framework: Apply the metrics of the framework to OSS projects
    a. an active (healthy) OSS project
    b. an inactive (dead) OSS project

    The aim of this first step is to see if the tools and metrics work out in practice with a (OSS) project.
2. Comparing the testing/training results with findings from the related work

The aim of this second step is to verify the results of the framework with findings from related work. Of most interest are papers that also give the values of the metrics they calculated, because the feasibility study shall show if and how good it is possible to compare these values with values by the framework.

The result of this part is a test of the framework by comparing with previous results from related work ("learning from the past").

The **Case Study** shall include testing/Training the framework: Apply the metrics of the framework to

a. other active OSS projects
b. industry projects

The aim of this first step is to see if the tools and metrics work out in practice with a (OSS) project.

The result of this part is a test of the framework by applying to some software projects ("predicting the future").

5 Systematic Literature Review on Quality Attributes, Metrics and Tools

A systematic literature research (SLR) on health indicators for OSS projects is the first part of the work:

1. Identifying metrics from SLR
    a. Match metrics to different quality attributes from ISO SQuaRE [1]
2. Identifying tools from SLR

This chapter describes the results of the SLR.

5.1 Systematic Literature Research

The goal of this systematic literature research (SLR) is to find software projects that are successful and thus showed a good health. We are interested in related work, because in most cases the authors give arguments for a good or bad health status of a project. These health indicators found in literature answer **RI.1** by showing the project health of analyzed projects. Additionally, the indicators found can be related to quality criteria answering **RI.2**. Additionally the authors also often name the metrics and sometimes the tools they used for interpreting a health indicator. We use their findings to address **RI.4** and **RI.5**.

Based on previous work on Health Indicators [10] and [11], we executed a SLR in the bibliographic database Scopus[14] by using the following search string

Search string for Health Indicators: < TITLE-ABS-KEY("PROJECT HEALTH" AND "SOFTWARE") AND SUBJAREA(COMP OR ENGI) >

This advanced search only returned 23 results, which is actually not a high number.

First we checked the results for relevance, i.e. if the results were out of scope and e.g. not concerning software development. A first check based on the title and abstract; a second in detailed analysis of the content.

During reading the resulting literature, we checked if each work described (i.e. named used) metrics and projects analyzed. Of special interest were open source software (OSS) projects, because it is fairly easy to get its data and to verify those results. For all relevant

[14] https://www.scopus.com/

papers we looked for the mentioned data sources, the approach of the paper, the described metrics and tools as well as the OSS projects under investigation.

Eight out of the 23 resulting papers were out of scope and six were also not relevant, because they mentioned neither metrics nor OSS projects. Including two papers without full text available and two duplicates, that are 18 out of all results, little more than 78% of the results were not usable.

Thus in a second search iteration (second level search) we scanned the references of the relevant papers, that had a promising title, recursively and added them to the publication list. This way we got additional 59 papers, of which only 10 are not relevant.

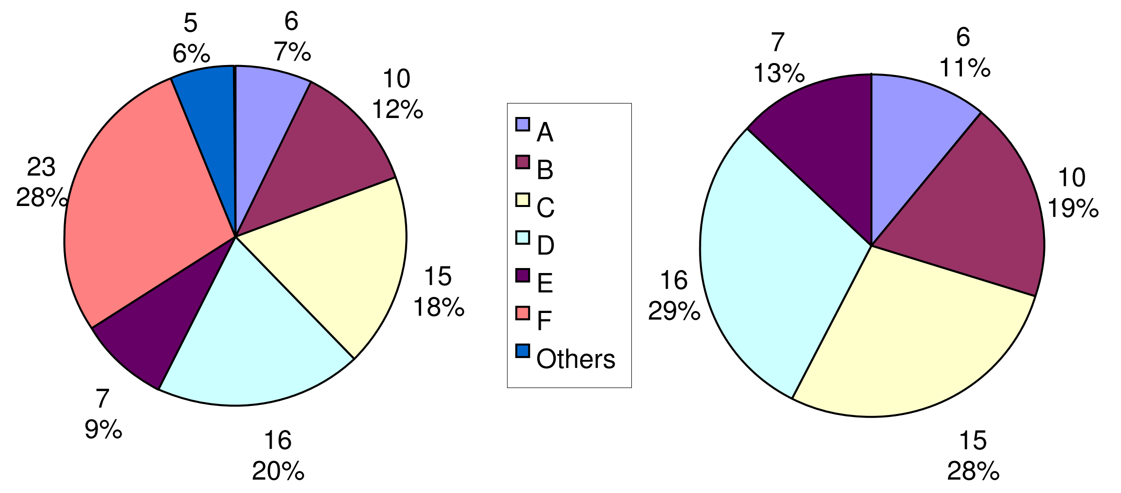
Based on this information we rated the papers using the scoring shown in following Table 1 with the given results:

Table 1 Ranking definition and results

Ranking	**Meaning**	**# of Papers**	**%**
A	Pick: Relevant well written paper with well documented measurements of OSS projects	6	7,32%
B	Relevant paper with well documented measurements of OSS projects	10	12,20%
C	Relevant paper with a sound approach analyzing some OSS projects	15	18,29%
D	Relevant paper analyzing some OSS projects (no measurements!)	16	19,51%
E	Relevant good written paper with well documented measurements, but without OSS project links (no trace possible!)	7	8,54%
F	Out of scope	23	28,05%
Others	Duplicates or no full text available	5	6,10%
Sum		**82**	**100%**

Figure 10 visualizes the ranking of the papers found in the SLR:

Figure 10 Ranking all and relevant results



The left shows the distribution of all 82 papers found, while the right shows the ranking of only the 54 papers relevant for further research. The difference and distribution is interesting, because it has an impact on the retrieval of metrics and tools.

Altogether we identified 59 different OSS projects. A summary is given in Table 2, while the full list is given in the appendix.

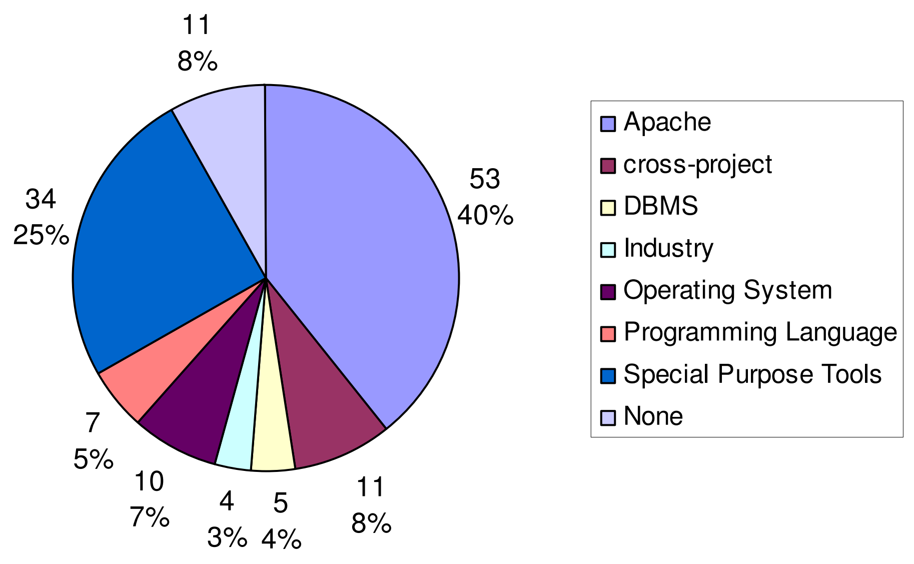
Table 2 Summary of OSS projects analyzed

Project Type	Projects	# of Papers	%
Apache	Ant, Cocoon, Excalibur, HTTPD, Jakarta, Lenya, Log4J, Lucene, MyFaces, Ode, OJB, OpenJPA, POI, Roller, Slide, Struts, Tomcat, Woden and Xindice	53	39,26%
cross-project		11	8,15%
DBMS	HSQLDB, MySQL and PostGreSQL	5	3,70%
Industry		4	2,96%
Operating System	Fedora Linux, FreeBSD, Linux, OpenBSD, RedHat Enterprise Linux and SUSE Linux	10	7,41%
Programming	Per and Python	7	5,19%

Language			
Special Purpose Tools	ArgoUML, Azureus, BIND, CMS, CodeCrawler, Cultivate, Eclipse, Evolution, gcc, Ghostscript, Gnome, Gutenprint, JBoss AS, jEdit, Jfreechart, Jgit, JUnit, KDE, Mozilla, NetBeans, OpenOffice, Sendmail, Squirrel, StarOffice, Xdoclet and XFree86	34	25,19%
None		11	8,15%
Sum		**135**	**100,00%**

Figure 11 shows which OSS projects of the SLR are best analyzed:

Figure 11 OSS projects types



This shows that in literature the OSS projects are not equally good analyzed. Some projects receive a lot of attention, which might also be interpreted as an indicator of good health of that software project.

5.2 Metrics

As defined in **RI.2**, which sees the need for metrics to calculate quality attributes, the search for metrics is a goal of the work, as enables automation.

Based on the SLR on health indicators from section 5.1 we also analyzed the metrics identified in the papers. There are plenty of suggested metrics (altogether 115 different, see Table 15 in the appendix for a full list) in the 54 relevant papers, but it was possible to group the found metrics. Table 3 shows the classification to different kind of metrics and some examples of metrics identified by the SLR.

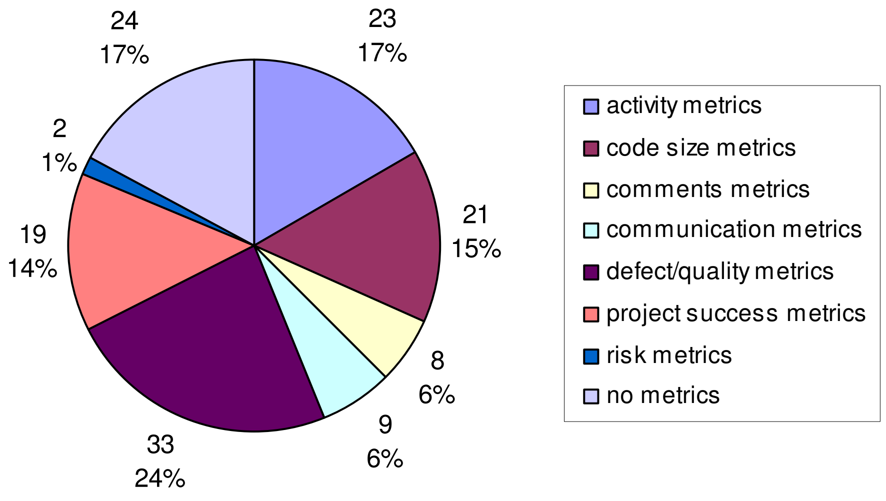
Table 3 Metrics Groups with examples

Classification	Examples of suggested metrics	# of Papers	%
activity metrics	Developer Contribution Pattern (Number of SCM Commits/Number of Email Conversation, Number of Defect Status Changes) [M1], Number of Active Contributors [M32], Responding Speed to Bug Reports [M21], Service Delays on Open Issues [M34]	23	16,55%
code size metrics	KLOC/time period added [M32], Number of Administrators [M36], Number of CVS commits [M36], Number of Developers [M36]	21	15,11%
comments metrics	Comment Frequency [M37], Proportion of Code to Comments [M16]	8	5,76%
communication metrics	Communication and Use Intensity (number of downloads compared to mailing list activity) [M2], Total Communication Metric (Number of communication artefacts / time) [M28], User Coupling Metric (Communication graph based on mailing list) [M28]	9	6,47%
defect/quality metrics	Bug History Metric (bug activities during a certain period of time) [M28], Defect Density (Post-release Defects/KLOCA) [M5], Defect Removal Time: Defect Confirmed [M1], Defect Reported Timestamp - Defect Confirmed	33	23,74%

	Timestamp [M1], Defect Service Delay (Defect Response Time: Time to resolve problem reports [M5]		
project success metrics	Age of Project versa Number of Developers [M23], Number of Downloads vs. Number of Developers [M23], Number of weekly Downloads [M17], Page Views vs. Number of Developers [M23]	19	13,67%
risk metrics	Risk Factor Analysis and Classification [M38]	2	1,44%
None	Paper describes no metrics or no full text available	24	17,27%
Sum		**139**	**100,00%**

Figure 12 visualizes the classification of the 139 different metrics found in the SLR. We see that the different metrics groups distribute quite fine.

Figure 12 Metrics Groups identified



The 16 A- and B-ranked papers (see Table 1), which we classified as papers with well documented measurements of OSS projects in the SLR, present 48 (34 different) OSS projects and 27 different metrics. Except project success metrics and risk metrics, for all other metrics groups at least three metrics are described in detail in these papers and only

the defect/quality metrics group stands out with ten metrics. This means that we found enough well documented metrics belonging to each group, as risk metrics were not in scope of the SLR.

Note that the number metrics of relevant papers is much more than the number of relevant papers found, as 22 papers of the 54 papers in the SLR named more than one metric. The most metrics are mentioned by seven papers, naming more than five papers, together 68 metrics. It is obvious that some papers name metrics that belong to more than one classification.

Some of the papers just name a metric and do not give a definition. That is why we tried to compare the findings of SLR with other information about OSS projects. We found calculated metrics at Open Hub[15]. The metrics group communication, activity, code size and comments exist also at Open Hub (former Ohloh). For example the Apache HTTP Server project page[16] and factoids[17] list similar metrics.

5.3 Tools

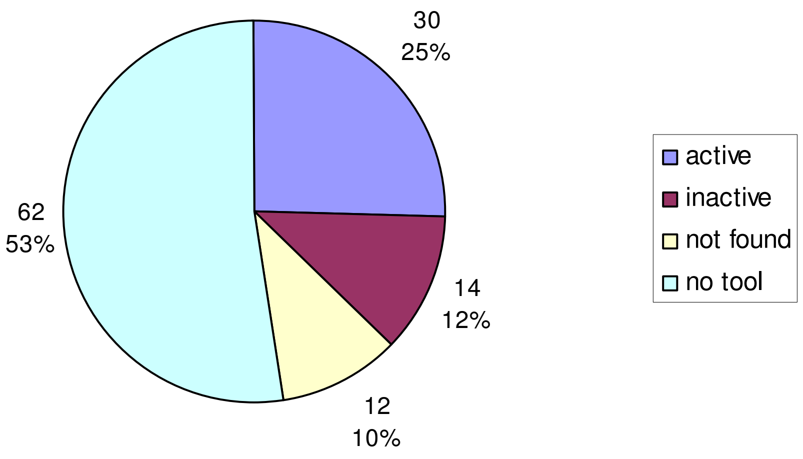
Like at the search for metrics in the section above, we also analyzed the tools identified in the papers of the SLR. The number of tools mentioned was not so big compared to the number of metrics: 56 tools were named in the papers. We checked, if the mentioned tools exists and, if found, classified them as active or inactive tool having no activity during the last two years. Figure 12 shows these numbers:

[15] https://www.openhub.net/

[16] https://www.openhub.net/p/apache

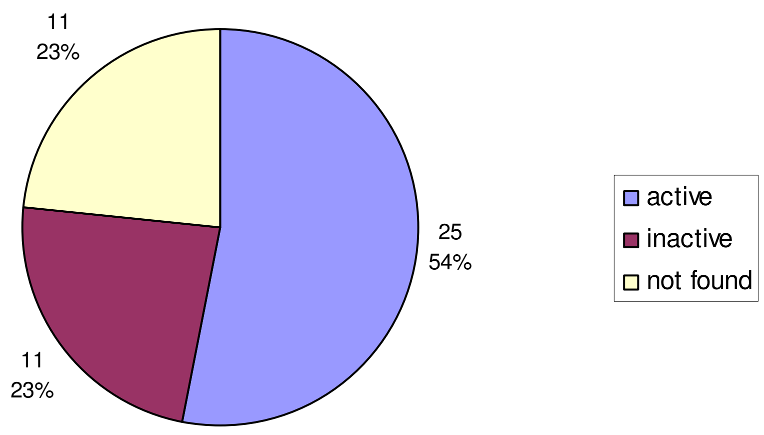
[17] https://www.openhub.net/p/apache/factoids

Figure 13 All tool/paper combinations of the SLR



62 papers named no tools, but not many papers used the same tools. Mostly papers of the same author used the same tool for more than one paper. See Table 16 in the appendix for a full list. Leaving out the duplicates, we still found 47 different tools, which is presented in Figure 14 again with the classification of not found, active and inactive tools.

Figure 14 Different tools identified in the SLR



These 36 available tools and the 25 active tools among them (i.e. almost 70%) are not really a high number; but compared with the 54 relevant papers, on average nearly every second paper had a tool given. Also the metrics of the 21 A- and B-ranked papers of the SLR were calculated with ten active tools shown in Table 4 with a short description of the tool.

Table 4 Active tools of A- and B-ranked papers

No.	Tool	Description	License
1	Bugzilla Query Commands [M1]	use Command-line Bugzilla Queries, see https://www.bugzilla.org/docs/2.16/html/cmdline.html	MPL
2	c_count [M47]	c_count counts lines, statements, other simple measures of C/C++ source programs, see http://invisible-island.net/c_count/c_count.html	custom
3	Eclipse Checkstyle Plugin [M1], [M33]	The Eclipse Checkstyle Plugin (aka eclipse-cs) integrates the static source code analyzer Checkstyle into the Eclipse IDE, see http://eclipse-cs.sourceforge.net/	LGPLv2
4	JIRA Query Commands [M1]	use the query language JQL of Atlassian JIRA, see https://developer.atlassian.com/jiradev/jira-apis/jira-rest-apis/jira-rest-api-tutorials/jira-rest-api-example-query-issues	commercial
5	Logiscope [M37]	Automatic Code Analysis with Logiscope, see http://www.kalimetrix.com/logiscope	commercial
6	Resource Standard Metrics (RSM) [M47]	RSM by M Squared Technologies provides a standard method for analyzing C, ANSI C++, C# and Java source code across operating systems, see http://msquaredtechnologies.com/	commercial
7	SLOCCount [M8], [M31]	set of tools for counting physical Source Lines of Code (SLOC), see http://www.dwheeler.com/sloccount/	GPLv2
8	Source Monitor [M47]	SourceMonitor counts lines, comments and calculates metrics (e.g. complexity), see	Freeware

		http://www.campwoodsw.com/	
9	SPSS [M28], [M37]	IM SPSS Statistics is a software package used for statistical analysis, see http://www.ibm.com/software/analytics/spss/	commercial
10	Understand [M47]	This Static Code Analysis Tool is an IDE built from the ground up to help you fully comprehend your source code, see https://scitools.com/	commercial

Most of the identified active tools are commercial ones. The open source tools were either limited in functionality or focused on a special topic (language or metric) or not active any more.

5.4 Summary

The SLR on health indicators resulted in 54 relevant papers out of 82 found. These papers were classified by usefulness and further analyzed looking for metrics, tools and projects as sources of its findings. All papers named 115 different metrics and 47 different tools. This work presented the results of the metrics and tools search in the SLR, trying to find some anomalies and/or commonalities. Those are the limitations found:

Though it was a systematic literature research, it is sure it did not contain all papers relevant to this topic due to its search string “health indicators”. The search string “longevity” is even worse, but it is clear that metrics and tools are also described in papers on other topics related to open source projects.

Another limitation, likely of the search string, is that the search returned many papers without metrics and even more papers without tools. Including the term “metrics” or “tool” to the search string would reduce the number of papers even further leaving out all papers that do not use this specific term.

A grave limitation is the return quality of the papers. Not all papers give a definition of the metrics they use. Also risk metrics are rarely mentioned, but this may also be downside of the search term “project health” – or simply sign for risk metrics not being present in literature and tools.

Surprisingly project success metrics are only identified for papers without referenced data or OSS projects. This leads to two possible reasons: first, project success metrics may not be easy to measure reliable or second, they may be added to other groups.

Interestingly the tools written for academic purposes can't be found or are discontinued and are thus listed as inactive. One reason might be that they are not valuable enough, meaning they are useless to continue as an open source or commercial project. So this is an example for a not long-living project. The license cannot be the reason for abundance: if the software had some unique features, it would live on as an OSS project.

Further comparison with alternative other open source review sites like Open Hub or static code analysis tools like SonarQube[18] or Kiuwan[19] should lead to an understanding of commonly used metrics.

[18] http://www.sonarqube.org/

[19] https://www.kiuwan.com/

6 Tool Selection Process

Given all the metrics found in the SLR in chapter 5 and also plenty of tools that measure these metrics, we want to know which tools are most suitable for calculating metrics. These tools will be used to answer **RI.5**, as they calculate longevity metrics.

All the 25 active tools found in section 5.3 of the previous SLR plus additional tools found by a web-based research on tools calculating metrics must be evaluated to select them for usage in the framework for longevity evaluation.

To make this selection reproducible and well documented, the systematic tool evaluation process of *Poston* and *Sexton* [23] is used. Additional to their presented systematic tool evaluation process, we also use their set of forms for evaluation. This should lead to more well-founded results when doing a tool evaluation. Their proposed evaluation process is structured in four steps:

1. identifying user needs (including the definition of mandatory features of a tool),
2. defining the tool selection criteria and prioritizing them,
3. finding available tools and classify them and
4. evaluating candidate tools and selecting the best fitting [23].

We follow these steps doing the first three all during the set up of the tool study.

6.1 Tool Study Setting

The requirements listed in Table 5 must be fulfilled by all tools, otherwise no easy integration into the longevity framework or test would be possible.

Table 5 Mandatory tool requirements

No.	Requirement Category	Mandatory Requirement
1	General Requirements	Availability for testing purpose
2	Metrics Calculation	Ability of calculating multiple metrics of at least one metrics group of Table 3 from the SLR
3	Metrics Calculation	Support of the programming language Java
4	Export Functionality	Export calculated metrics as XML or another interoperable format via API

The tool needs to be available for testing, because a well-founded selection and also further use would not be possible. This means to exclude commercial tools without a test version available from the current evaluation.

A tool must calculate a metric of a relevant type, i.e. the metric must be clearly assignable to a metrics group of Table 3. That way we make sure to have a link to quality attributes.

Furthermore the tool must be able to export the metrics calculated in an easily readable format. The metrics calculation and export must be also easy to run, read and to integrate. We need this export functionality for the integration to a framework making the tools work together.

The mandatory requirements are also used for a web-based research[20] on tools calculating metrics. Table 6 shows the results of that search.

Table 6 Additional tools found

No.	**Tool**	**Description**
1	Alitheia-Core	platform for software analytics and software engineering research, see https://github.com/istlab/Alitheia-Core
2	BugzillaMetrics	Runs self-defined metrics on nearly any attribute and event stored in Bugzilla, see http://www.bugzillametrics.org/
3	CLOC	Count Lines of Code (CLOC) counts blank lines, comment lines, and physical lines of source code in many programming languages, see https://github.com/AlDanial/cloc
4	JArchitect	Static code analysis tool for java that offers a wide range of features: calculates 82 code quality metrics, run own queries and integrates own plug-ins etc. See http://jarchitect.com/
5	Kiuwan	Software Analytics in the Cloud: Static code analysis using metrics, see https://www.kiuwan.com/
6	nDepend	Static code analysis tool for .NET with the same features as JArchtitect, see http://www.ndepend.com/
7	Open Hub	Retrieves data from open source repositories and provides statistics about the longevity of projects, their licenses and

[20] mainly looking at tools from https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

		software metrics and commit statistics, see http://www.openhub.net
8	ProjectCodeMeter	Estimates Software Development Cost & Time, measures Code Quality and Team Productivity using code analysis, see http://www.projectcodemeter.com/cost_estimation/index.html
9	Sonargraph	Sonargraph is a static code analyzer that computes 100's of metrics, finds code duplications, visualizes dependencies and allows scripting own metrics or code checkers, see https://www.hello2morrow.com/products/sonargraph
10	SonarQube	Static code analysis tool, see http://www.sonarqube.org/
11	SourceMeter	Static source code analysis solution for Java, C/C++, Python and RPG, see https://www.sourcemeter.com/

All of these tools are available, it would be impossible to find them otherwise. Together with the 25 active tools from the SLR we run a pre-selection. The check, if a tool fulfils all mandatory requirements, resulted in a list of remaining seven tools shown in Table 7. The full reasoned result of this check is listed in Table 17 in the appendix.

Table 7 Remaining tools for detailed evaluation

No.	**Tool**
1	BugzillaMetrics
2	CLOC
3	Eclipse Checkstyle Plugin
4	Open Hub
5	Resource Standard Metrics (RSM)
6	Source Monitor
7	Understand

For each of the requirement category multiple additional selection criteria are defined as listed in Table 8 and for prioritisation weighted by a factor [23].

Table 8 Tool Selection Criteria

No.	**Selection Criteria**	**Priority**	**Weight**
A	General Requirements		
1	Availability for Testing Purpose	C (Critical)	10,0
2	Platform independency: tool runs on Windows	H (High)	7,0
3	Platform independency: tool runs on Linux	M (Medium)	3,5
4	Platform independency: tool runs on Mac OS	L (Low)	1,0
5	Time of Installation not longer than 30 minutes	H (High)	7,0
6	Simple Installation using script(s) or installer	M (Medium)	3,5
B	Metrics Calculation		
7	Ability to calculate multiple metrics of at least one metrics group of Table 3	C (Critical)	10,0
8	Support of the programming language Java	H (High)	7,0
9	Ability to calculate metrics of both metric groups “code size metrics” and “defect/quality metrics”	H (High)	7,0
10	Ability to calculate metrics of at least another metrics groups (e.g. “activity metrics” or “communication metrics”)	M (Medium)	3,5
11	Support of additional sources (e.g. bug tracker, mailings lists)	M (Medium)	3,5
12	Support of evaluating and checking design documents	M (Medium)	3,5
C	Export Functionality		
13	Export calculated metrics as XML or another interoperable format via API	C (Critical)	10,0
14	Usable User / Developer Guide for the Export / API exists	H (High)	7,0
15	Support for a client to use the export	M (Medium)	3,5
16	Examples available	M (Medium)	3,5
17	Coding examples available	L (Low)	1,0

With that selection criteria defined we can start the tool study.

6.2 Tool Study

Sticking to the tool evaluation approach proposed by Poston and Sexton, every selection criterion is rated with a value between 0 and 1 standing for the percentage of fulfilling the criterion [22]. We use the following scales:

- 0,0 if the selection criterion is not fulfilled
- 0,35 if the selection criterion is to a small amount fulfilled
- 0,7 if the selection criterion is mainly fulfilled
- 1,0 if the selection criterion is completely fulfilled

The above rating is replaced by a simpler one, if a distinction between four cases is far too complicated to answer because the selection criterion is just a yes or no question:

- 0,0 if the selection criterion is not fulfilled
- 0,5 if the selection criterion is partially fulfilled
- 1,0 if the selection criterion is completely fulfilled

For all pre-selected tools we calculate the numbers for each selection criterion. Additionally to the scale the value is also weighted and all numbers are summarized.

Some criteria were not so easy to scale, as we had to search the documentation for this features.

The result of the scoring is shown in Table 9 (see Table 18 and Table 19 in the appendix for detailed results):

Table 9 Tool Study Results

Rank	Tool	Score
1	Understand	77,5
2	Open Hub	75,575
3	Resource Standard Metrics (RSM)	70,9
4	Source Monitor	69,725
5	CLOC	69,125
6	BugzillaMetrics	66,175
7	Eclipse Checkstyle plug-in	53,18

The commercial tool Understand and the web-based commercial Open Hub (free available for Open Source Projects) took the first two places. Understands looks like a real powerful tool providing not only metrics but also dependency analysis. Like all commercial client tools it satisfies with a nice installer. Like Open Hub they both provide a well-documented API including examples and a wide range of metrics, though Open Hub stands here clearly out in covering also activity metrics by analysing the check-in frequency of the source code repository.

The commercial tool Resource Standard Metrics (RSM), the freeware source monitor and the GPL tool CLOC cover most of the requirements sufficiently, but not outstanding.

BugzillaMetrics does fine, considered that it can't analyze Java Code. If you are interesting in metrics of a bug tracker like Bugzilla or Mantis you should definitely take look at this solution. The Eclipse Checkstyle plug-in does have enough features to rank better; it mostly does poorly because its main intention is not to calculate metrics.

6.3 Results and Tool Selection

Looking at the ranking we see two clear choices for a framework: Understand and Open Hub. CLOC and Source Monitor should be integrated as well, because they are can be run and integrated via command line. Always to keep in mind is the license of a tool, especially for CLOC, as it is GPL based. RSM will not be included to the framework, because in the trial version it supports only the evaluation of 20 files. We recommend taking a deeper insight and integrating only those tools that provide interesting metrics.

All in all seem the results of the pre-selection and the final scoring valid. All tools of the pre-selection are usable and achieve quite good results in the tests during the scoring. And all results of the scoring realise between 85 and 58 percent of the maximum reachable score. We have to consider that we scored only tools that fulfilled the mandatory requirements, which alone account for a third of the scoring.

Like every study also this tools study, though based on the academically found method by Poston and Sexton [22], has **limitations**.

Firstly, requirements used for the tool (pre-) selection and scenarios used for the selection criteria are based on real-world settings. These settings also have an impact on the weight factor used.

Secondly, the tool selection focuses on available tools (i.e. open source software, test versions) excluding commercial tools from the current evaluation; if no test version is available.

Thirdly, data collection is based on subjective assessment that needs to be revisited to increase evidence. A possible solution would be to let another person do the scoring independently and compare the results.

And last but not least more a design issue: we left out requirements or selection criteria that can't be measured precisely. That way it is hard to include non functional requirements like usability into the study.

Generally speaking, for a plug-in it is complicated to perform as good as a complete tool, because the installation takes a big part of the score. Furthermore a plug-in might not even make it into the pre-selection, as it might go unnoticed, especially if the containing tool does not advertises the features of its plug-in.

7 Framework for Longevity Evaluation

Based on the SLR, the tools selection and related work we build the framework:

1. combining selected tools that measure metrics from the SLR
2. compare with alternative frameworks from related work

This chapter describes the work done for building the framework. It answers **RI.6**, how a framework can combine the selected tools.

The **requirements** of the framework for longevity evaluation can be defined straight ahead: It needs tools that calculate comparable metrics, which are relevant for software longevity.

Additionally these **principle considerations** should be taken into account. We do not want to create another framework that will be abandoned in the future. A much better idea is to enhance existing tools with a metrics section about project health / longevity. Such a section for example would fit very well to the tool ProjectCodeMeter because this software has a quite similar way of presenting metrics in a manageable view. May be it is even possible to sell that feature.

That is why we build only a prototype of the framework. This means that we calculate all metrics using the selected tools manually and sum up the result using a spreadsheet.

As we use only tools approved by the tool selection process, it is clear that all these tools have an export or an API for their calculated metrics. Integrating this proof of concept into a framework from related work or a standalone software tool would be the next step done in future work.

7.1 Combination of Tools

To combine tools, all metrics of all tools get listed. As stated in section 6.3, we recommend taking a deeper insight and integrating only those tools that provide interesting metrics. So we discard all metrics that are not used for longevity.

We do not list the resulting big list, as they can be extracted from Table 10.

It became more and more obvious during evaluation that every tool has other names for the same metrics. Some tools have a good documentation, which defines precisely what a metric means and often also how it is calculated. Resource Standard Metrics (RSM) however calculates a lot of metrics, but has no documentation. So we were not able to list these metrics simply we did not know, what they mean. The same accounts for CLOC, in its very short documentation it does not even mention the word “metrics” – and “measure” also only once. Open Hub has interesting values, but it does not provides enough details about the values giving often just a classification within a range. This is good for a human readable overview, but is not necessarily correct.

7.2 Metrics Selection

Table 10 answers **RI.3**; as it is possible to read from it which selected tool calculates a metric important for longevity.

Table 10 Selected metrics and tools

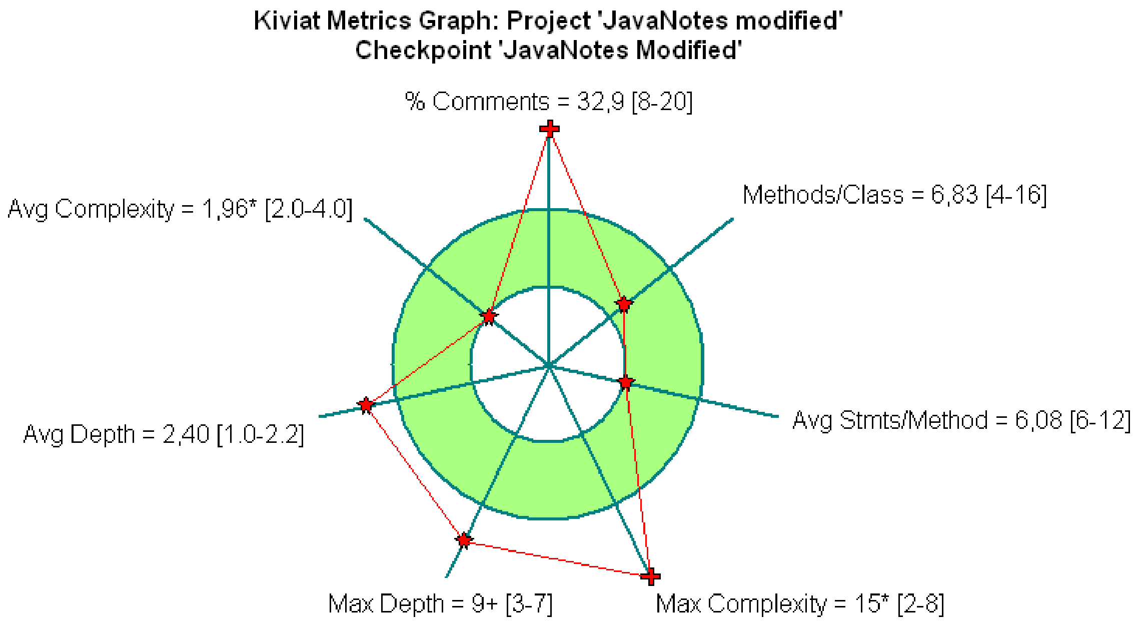
No.	Metric	Tool(s)	Quality Attribute
1	Percent Lines with Comments / Ratio of comment lines to code lines	SourceMonitor, Understand, RSM, (Open Hub)	Maintainability [1]
2	Methods per Class	SourceMonitor, Understand, RSM	Modularity [1]
3	Average Statements per Method	SourceMonitor, Understand	Modularity [1]
4	**Maximum Complexity**	**SourceMonitor**	**Testability [46]**
5	**Average Block Depth / Depth of Inheritance Tree**	**SourceMonitor, (Understand)**	**Reusability [46]**
6	**Average Complexity**	**SourceMonitor, Understand, RSM**	**Testability [46]**
7	Number of lines containing source code (aka LOC)	SourceMonitor, Understand, RSM, Open Hub	Maintainability [1]
8	**Average Essential Complexity**	**Understand**	**Testability [46]**
9	**Lack of Cohesion in Methods**	**Understand**	**Modularity, Reusability [46]**
10	Number of commits	Open Hub	Modifiability [1]
11	Number of contributors	Open Hub	Analysability [1]
12	Commits per year	Open Hub	Maintainability [1]
13	COCOMO	Open Hub	Efficiency [1]
14	Project Activity Index	Open Hub	Satisfaction [1]

We marked the metrics relevant for software longevity by printing the line bold. Actually this applies to all identified metrics, which have a quality attribute that is relevant for longevity [46].

The bold marked tools Understand, RSM and SourceMonitor in Table 10 help calculate metrics for quality attributes that influence software longevity. This is the answer to **RI.5**.

We assume that, if all bold marked metrics do not exceed a certain threshold like shown in Figure 15 for an example student Java project, the software project has a good longevity answering **RI.1**.

Figure 15 Example Kiviat Metrics Graph by SourceMonitor



Alternatively, if more than four metrics out of the eight metrics in Table 10, which are capable of setting an acceptance range, have a metric value calculated above of an acceptable value and are relevant for longevity (i.e. have one value of a bold marked metric out of range), the project exceeds the fixed threshold of 50 percent and thus is relevant for longevity. That result answers **RI.1** about the longevity of a software project with “Yes”. The same idea is used by the commercial cloud-based software Kiuwan (obviously the Kiviat diagram is where they got their name from).

Each metric used in the evaluation framework lists its linked quality attributes. This linking is based on identified quality attributes from ISO SQuaRE in literature [46].

As we know from the SLR which metric is used to describe healthy/longevity projects, we can make a list of quality attributes affecting that metric and thus can answer **RI.2** that wants to know which quality criteria are related to the longevity of a software project. How to find out which quality attributes are most important for longevity shows an in-depth analysis of Information System Longevity aspects [46].

8 Feasibility Study

We test the framework learning form the past:

1. Testing/Training the framework: Apply the metrics of the framework to OSS projects
2. Compare the testing/training results with findings from the related work

This chapter describes the work done for evaluating the framework with a feasibility study, i.e. comparing the framework results with findings of related work.

As this work uses open source projects for analysis, this chapter starts with an overview of free software and open source software.

8.1 Open Source Software

Strictly speaking there is a difference between free software and open source software, as already mentioned in the introduction in chapter 1.2 Definitions. The author likes the term Open Source Software more, that is why this term is used throughout the thesis - for both Free Software and Open Source Software.

Actually the idea behind Open Source Software is that all users and every developer can participate. They all shall have any possibilities to do so – without limitations. In comparison to closed source and / or privately owned software, also transparency and collaboration are principles of Open Source Software.

With the start of computers in the 1950s software was mainly scholarly work of academic researchers. Computer manufactures put focus on the hardware and added the software for free. User groups sharing tips and software with each other started. In the late 1960s writing software became increasingly more expensive, the era of legal protection of software including software patents started [16].

Nevertheless, the culture of sharing software never really stopped.

8.1.1 Free Software

In 1984 *Richard Stallman* launched the GNU project[21]. For this project Stallman also created the GNU General Public License (GPL). With its several different versions it had a deep impact on software producers, as Stallman acted as an outspoken but also controversial advocate for free software - especially in contrast to proprietary software.

[21] www.gnu.org

The term **Free Software** was introduced by Stallman in 1986 and means software that the end user is allowed to use (i.e. run), study, share (copy and distribute) and modify (change and improve). The term “free” is meant as in “free speech”, not as “free of charge” (gratis) [37]. This definition known as the Free Software Definition is used by the Free Software Foundation (FSF).

The aim of the FSF is to promote Free Software politically, legally and socially. This charity wants to help users to get Free Software as defined in The Free Software Definition[22].

8.1.2 Open Source

Despite of GNU’s success not everyone used the GPL. Plenty of slightly different license agreements exist.

The term Free Software is older than the term **Open Source Software**, which was introduced by *Bruce Perens* for the Open Source Initiative (OSI) stating which software license fulfils the need of the open-source certification of OSI. This definition was based on the Debian Free Software Guidelines of the FSF. So despite of fundamental philosophical differences between the FSF and the OSI there are not so many differences. Most software license fits both definitions. [38]

The differences between the various licenses require detailed study, but they all share a common ground. If a piece of software is not declared public domain, copyright law is protecting it. All open source software licences give users the right to access the source code. If the licence is one of the FSF (e.g. GPL) it additionally requires that any software that is build with this GPL based software must be distributed under the GPL itself. This part of the GPL is also called the “viral” or “copyleft” provision. [16]

Each license has its own purpose. According to *Krishnamurthy* there are four significant differences [19]:

1. Does the license allow mixing its open source code with non open source code?
2. Is it possible to make modifications and not to return them to the original author?
3. Can the open source code be relicensed by anyone?
4. Are there special privileges for the original copyright holder (i.e. the author)?

The result of this comparison is shown in Table .

[22] https://www.gnu.org/philosophy/free-sw.html

Table 11 Comparison of open source licenses [19]

License	Can be mixed with non-free software	Modifications can be made privately and not returned to author	Can be relicensed by anyone	Contains special privileges for the original copyright holder over user's modifications
General Public License (GPL)	No	No	No	No
GNU Library General Public License	Yes	No	No	No
Berkeley System Definition (BSD)	Yes	Yes	No	No
Netscape Public License	Yes	Yes	No	Yes
Mozilla Public License (MPL)	Yes	Yes	No	No
Public Domain	Yes	Yes	Yes	No

The chosen license has an important impact on the project prospects, as it defines the possibilities of further usage. The GPL for example forbids the bundling and selling of an open source project with other code. [19]

According to *Miller* et al. "some open source advocates consider copyleft optional. Thus, OSI accepts FSF's GPL license, but it also endorses some licenses that FSF rejects." [16]

Miller et al. state that "FOSS and commercial software need each other", because "the competition between the two keeps the software marketplace in check with added diversity and innovation". [16] The author thinks that mostly the need for innovation leads to new software and just this incentive is the key to both open source and commercial software development. There is surely more competition, if an open source and a commercial version of a software for a certain purpose exist, but mostly the features, price or support count. If a commercial version of open source software is developed, someone sees a market for it. This leads to more competition of course, but does not necessarily keep the software marketplace in check.

8.1.2.1 MIT, BSD, Apache and Academic Free Licences

The licenses of the Massachusetts Institute of Technology (MIT) and the Berkeley Software Distribution (BSD) are among the oldest open source licenses [17]. They both originate from an academic background, as both were started by a university in the United States of America. These licenses read easily and contain all the basic principles of a typical open source license. Together with the Apache License and the Academic Free License they have in common that these projects do not require projects using such licensed code to distribute the source code. So these licenses are easy to handle, as it is not complicated to follow its clauses.

The MIT License[23] contains the author and the year of the release, followed by a clause granting permission free of charge. A precondition for usage is to preserve the copyright notice and include them to all copies of the software. The license ends with a warranty disclaimer. A warranty disclaimer alone does not take away all the risk: additional agreements during sale or other laws can nullify the disclaimer [17].

The BSD License exists in multiple forms. Until 1999 the BSD had a clause forcing users to acknowledge that the software include BSD code in advertisements. Though this clause might still remain, it has no longer a legal effect [17]. The only key difference to the MIT License is an additional clause that tries to protect the reputation of the creator.

The Apache 1.1 License is very similar to the BSD License. It does not have the advertising clause, but after the warranty disclaimer it additionally names some original contributors. The Apache 2.0 License was rewritten in 2004.

The Apache 2.0 License is a complete revision of the previous version 1.1. Its text is longer and more complex, defining the rights granted in detail. New provisions are that patent use is granted and other licenses are allowed for derivative works. “By making a Contribution, a licensee is agreeing to have that addition to the Work licensed under the same, open, terms applicable to the original Work. (...) But there is no obligation to make a Contribution: licensees are free to take their Derivative Work and license it under a different license.” [17]

The Academic Free License is essentially similar to the Apache 1.1 License. The license additionally clarifies patent law and adds two provisions concerning the choice of law and shifting of attorneys’ fees.

[23] https://opensource.org/licenses/MIT

8.1.2.2 GPL, LGPL and Mozilla Licences

The licenses described in this chapter are very different from the previous description (see chapter 8.1.2.1). They impose substantial limitations on the creation and usage of derivative works. The GNU General Public License[24] (GPL) demands that all changes by own work need to be distributed (i.e. made public) under the GPL and does not allow relicensing.

The GPL is one of the foundation open source licenses. The license was created by the Free Software Foundation (FSF) and is the preferred license for FSF projects.

The license is prefaced with a preamble, which defines the 3 main purposes of the GPL: Firstly to make sure that software is and stays free. Secondly to point out that software is distributed “as is” and without warranty. And thirdly to prohibit software patents. Compared with previous discussed licenses, the GPL is written more specific and more detailed.

The GNU Lesser General Public License[25] (LGPL) is an alternative license to GPL. This license allows the combination of proprietary and open source code. Originally this license was designed to use libraries with the GPL (hence the first name Library General Public License), but it is a license with just fewer rights (only for guaranteed freedom to modify the components licensed under LGPL). As with the GPL, multiple versions of the LGPL exist.

The Mozilla Public License[26] (MPL) is a hybrid out of the ideas of the GPL and the BSD license. While the BSD license allows relicensing, the GPL enforces new work to be published under the GPL. The MPL allows code that stands under its license to be mixed with code of other licenses, even proprietary code. However, code under the MPL must remain under the MPL and made freely available. So the MPL makes it possible to keep modules proprietary and in this way motivates open source enthusiast as well as companies to use it and help to improve the code modules. As written by Netscape, the MPL reads more like a corporate contract and it has also a focus on patent rights and its limited grant to an extend that is still consistent with an open source model. [17]

8.1.2.3 Qt, Artistic and Creative Commons Licences

Additional to the classical open source licenses presented above, there are also some other notable open source licenses mostly associated with particular programs: the Q Public License (of the Qt Toolkit) and the Artistic License (of Perl). Both of them have

[24] http://www.gnu.org/licenses/gpl.html

[25] https://www.gnu.org/copyleft/lesser.html

[26] https://www.mozilla.org/en-US/MPL/

special features and are frequently used, but mostly just applied to that kind of software for which the licenses were originally written. To sum up the license introduction, we also have a look at the Creative Commons license, an effort to bring the open source model of developing software also to literature and the arts.

The Q Public License Version 1.0[27] (QPL) was written by Trolltech to let the Qt system (the underlying system of the KDE linux desktop system) work legally together with the free idea of linux. It permits the distribution of modified source code as patches under less restrictive terms than modifications compiled into the original code, and it permits some rights to the code just to the initial developer. With the cross-licensing of the Qt Toolkit 4.0 as both GPL and QPL, KDE switched to the GPL and the QPL may become less important. With version 4.5 the Qt Toolkit changed its license to the LGPL.

Like the QPL the Artistic License[28] (or Perl Artistic License) is non-copyleft license. The Perl License adds an option for commercial usage to the Artistic License, whose intention is to "maintain 'artistic' control over the licensed software and derivative works created from it" [17]. Perl is dual licensed under both the Artistic License and the GPL, so it is likely to stumble across this license, if you use open source software. Two versions of the Artistic License exist: version 1.0 is vague and confusing and thus just an approved open source license; the extensively rewritten version 2.0 is also an approved free software (i.e. GPL compatible) license. Like QPL and MPL, this license is designed for centralized projects. It has some limitations as it is ambiguous about key terms concerning modification and distribution. However it is easy to comply with the spirit of the license and the license itself. [17]

Actually the Creative Commons licenses[29] (CC) are not licenses for open source software. The non-profit Creative Commons Corporation and supported by the Stanford University Law School created – inspired by the GPL – licenses to encourage creators of texts, music, web sites and film to make their work open source.

The Creative Commons Licenses are solidly constructed and well-written and build on modules [17]. Note that the "copyleft" idea from the GPL is called "share alike" in the CC, a module of the CC.

[27] https://opensource.org/licenses/QPL-1.0

[28] http://www.perlfoundation.org/artistic_license_1_0

[29] https://creativecommons.org/licenses/

8.1.2.4 License Usage

Although the MIT license is used most often according to GitHub[30] or BlackDuck[31], the GPL and MPL (especially as it influenced a lot of similar licenses) are successful too.

But the author agrees with *Laurent*, who does not only see the success but also the impact of a license:

> "As can be seen from the examples of the GPL and the MPL, the success of licenses is a factor less of the terms or the wording of those licenses than of the ideas that they represent. Powerful, meaningful ideas draw minds, and the success of open source and free software licensing is the result of the minds that such ideas can draw." [17]

8.2 Project Selection

Based on the SLR in chapter 5.1 containing 82 papers we sorted out the most frequently analyzed OSS projects.

Table 12 Most frequently used projects in context of health indicators

Rank	Project	# Papers	# different Authors	# data sets
1	**Apache HTTPD**	14 : [M1], [M2], [M3], [M4], [M5], [M6], [M7], [M8], [M9], [M10], [M11], [M12], [M13], [M14]	10	**3: [M1], [M5], [M13]**
2	Cross projects (top x sourceforge projects, etc.)	12: [M15], [M16], [M17], [M18], [M19], [M20], [M21], [M22], [M23], [M24], [M25], [M26]	12	3: [M16], [M18], [M21]
3	**Apache Tomcat**	9: [M1], [M2], [M3], [M8], [M27], [M28], [M29], [M30], [M31]	5	**4: [M1], [M28], [M30], [M31]**
4	Python	5: [M4], [M8], [M11], [M13], [M32]	4	1: [M13]

[30] https://github.com/blog/1964-license-usage-on-github-com

[31] https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses

5	Gnome	3: [M8], [M14], [M32]	3	0
6	Linux (Kernel)	3: [M4], [M7], [M8]	3	0
7	Mozilla	3: [M4], [M8], [M14]	3	0
8	PostGreSQL	3: [M11], [M12], [M13]	2	1: [M13]
9	Apache MyFaces	3: [M1], [M29], [M33]	1	1: [M1]
10	Apache Slide	3: [M1], [M2], [M3]	1	1: [M1]
11	Apache Xindice	3: [M1], [M2], [M3]	1	1: [M1]
12	**ArgoUML**	2: [M16], [M31]	2	**2: [M16], [M31]**
13	Eclipse	2: [M14], [M31]	2	1: [M31]
14	NetBeans	2: [M8], [M14]	2	0
15	Perl	2: [M4], [M8]	2	0
16	Apache Cocoon	2: [M27], [M28]	1	1: [M28]
17	Apache Lenya	2: [M28], [M34]	1	1: [M28]
18	Apache Log4J	2: [M28], [M34]	1	1: [M28]
19	jEdit	2: [M16], [M35]	1	1: [M16]
20	JFreeChart	2: [M16], [M35]	1	1: [M16]

As some authors used the same OSS projects for multiple papers, the number of different authors is a hint for the number of distinguished data set of OSS projects. We select only projects with well documented measures (i.e. ranked A or B) analyzed by at least two different authors marked bold in Table 12: Apache HTTPD, Apache Tomcat and ArgoUML. The papers analyzing cross projects do not cover identical or other projects of the SLR, so we can't use them.

Selecting these OSS projects might enable a comparison of different analysis approaches and results (derived from at least two different authors) as they are analyzing similar projects with (maybe) different data sources. Thus, we can use these projects/results for verification/validation and for justification purposes of the framework.

There are some OSS projects containing data from the papers analyzing OSS projects: flossmole.org, openhub.net (former ohloh.net), flossmetrics.org to name a few. Just the first two are fully active; the tool website of flossmetrics.org is no longer continued. Besides of analysis data (e.g. flossmole[32]), some OSS projects also contain tools for calculating measurements (e.g. ohcount[33]) or accessing source code (e.g.ohloh-scm[34] or sourcechange[35]).

It is notable to mention that the majority of papers analysed several projects on a higher level to investigate communication issues, bug data or community activities. Furthermore Apache projects were used for detailed investigations, e.g. to investigate health indicators. Other available open source projects have not been considered for applying metrics and measurements.

Additional projects to learn from the past

As defined in the concept for evaluation of the Solution Approach in section 4.3, we want to compare with both active and inactive projects from related work.

Thus additional to the selected active projects Apache HTTPD, Apache Tomcat and ArgoUML (marked bold in Table 12) we also need some inactive projects. We chose Apache Xindice and Apache Lenya.

This lead to our final list of OSS projects, which are used to compare the testing/training results with findings from the related work

1. Apache HTTPD as active OSS project [2], [3], [4] and [18]
2. Apache Tomcat as active OSS project [2], [6] and [7]
3. Apache Xindice as inactive OSS project [2]
4. Apache Lenya as inactive OSS project [6]

We chose these projects because they were named as active / inactive in the SLR. Additionally they are good documented projects, as related work provided some calculated values, making a comparison easy.

8.3 Feasibility Study

With the selected OSS projects we can the show results of the evaluation concept out of chapter 4.3:

We started with **Apache HTTPD** Server 2.4.17, using the actual stable version.

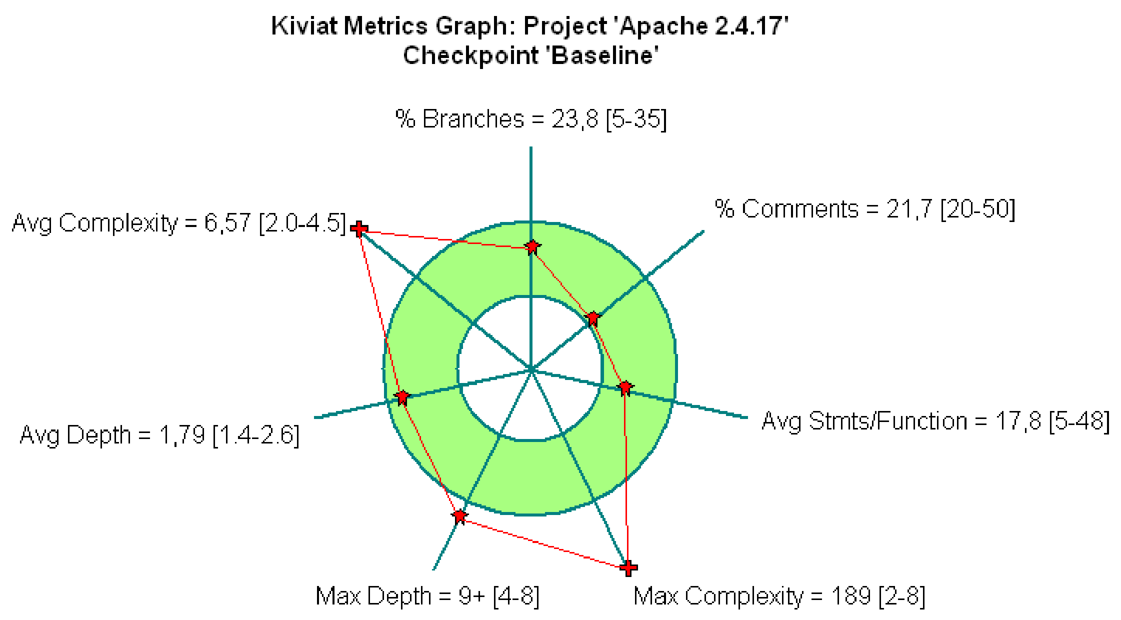
[32] Flossmole: http://flossmole.org/content/getting-data

[33] Ohcount: https://github.com/blackducksw/ohcount

[34] Ohloh-scm: https://github.com/blackducksw/ohloh_scm

[35] Sourcechange: http://sourceforge.net/projects/sourcechange

Figure 16 Apache HTTPD 2.4.17 Kiviat Metrics Graph by SourceMonitor



The Kiviat Metrics Graph show in Figure 16 shows a nice rating of some results. Note that only averaged values can be compared reasonably between OSS projects. Thus the metrics Maximum Complexity and Maximum Block Depth can not be compared. They are related to Average Complexity and Average Block Depth anyway. The possible range is suggested by SourceMonitor and it fits quite well. As Apache HTTPD is a large project, this may reason the high Maximum Block Depth. The metrics also show a high complexity. Due to the large code base, the percentage of comments in all lines of code shows that Apache HTTPD is not badly documented and so usable. Also the average statements per method point out a well written code. So a huge code base does not necessarily mean that the methods must be huge as well.

These were only values visualized by the first tool. The other tools show similar results in Table 13. As not all tools cover the calculation of all metrics, not all columns are filled. Yellow values must be checked, they seem to be wrong. The results of the feasibility study given for Apache HTTPD are just the numbers. We do not state here how which metric of a tool are exported and interpreted. This can be found in the technical report. Sometimes even small calculations are necessary to get a matching result. For matching the metrics of the different tools a good documentation is necessary to avoid guessing.

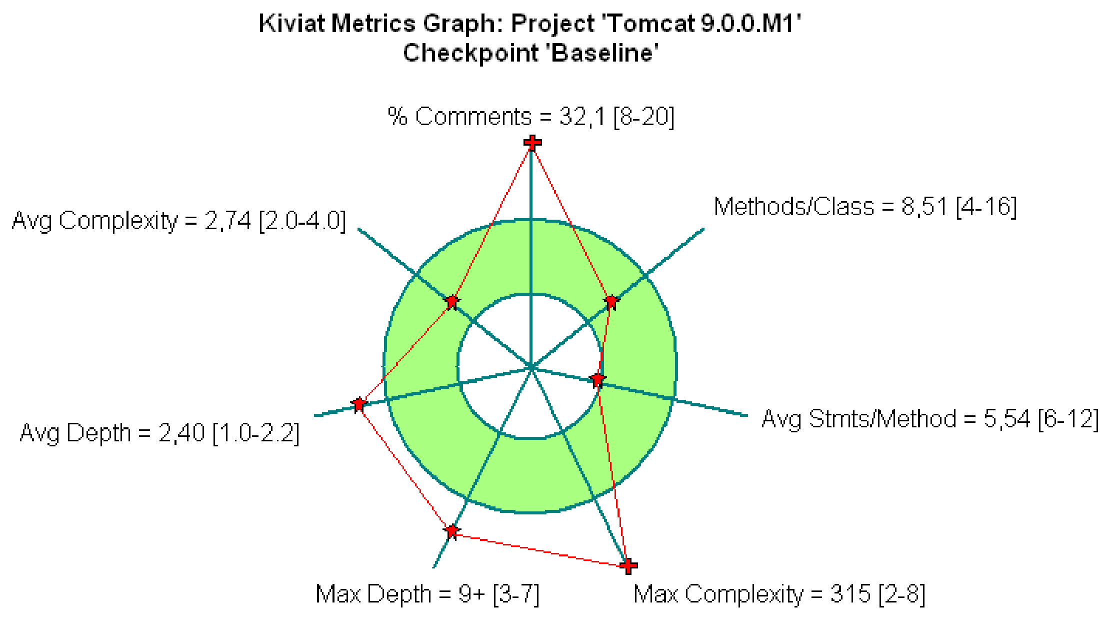
Table 13 Metrics calculated for Apache HTTPD 2.4.17

No.	Metric	SourceMonitor	Understand	Open Hub
1	Percent Lines with Comments / Ratio of comment lines to code lines	21,7	37	10,45
2	Methods per Class			
3	Average Statements per Method	17,8		
4	Maximum Complexity	189	165	
5	Average Block Depth / Depth of Inheritance Tree	1,79	0	
6	Average Complexity	6,57	6,15	
7	Number of lines containing source code (aka LOC)	250625	251898	1774749
8	Average Essential Complexity		3,14	
9	Lack of Cohesion in Methods		3,55	
10	Number of commits			65981
11	Number of contributors			118
12	Commits per year			2416
13	COCOMO			934040
14	Project Activity Index			70
15	Number of Files	416	423	

Problematic values seem to be the completely differing lines of code (LOC) metric. This is an issue we find in almost every project, so it can’t be related to a language. Of course, the number of LOC depends on the parts of the source code selected. Sometimes libraries are included in the source code and very often also tests. At Open Hub it is possible to define more than one source code repository. All these factors and also the programming languages supported by the tool lead to a difference.

The second project in the feasibility study is **Apache Tomcat** 9.0.0.M1.

Figure 17 Apache Tomcat 9.0.0.M1 Kiviat Metrics Graph by SourceMonitor

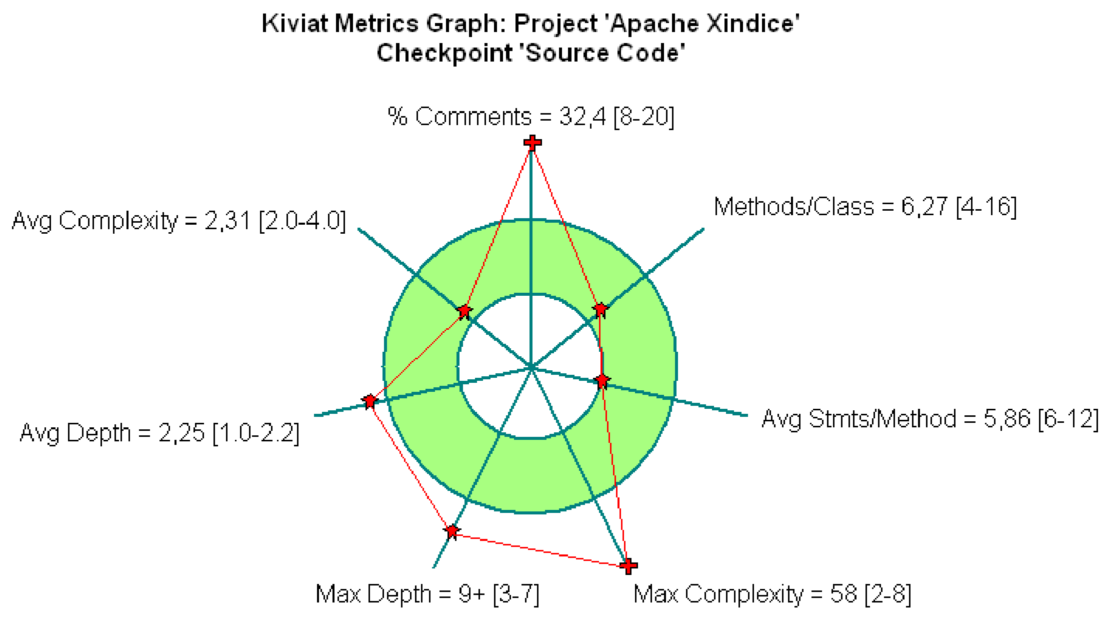


Compared to Apache HTTPD, Apache Tomcat shows significant lower value of the metric Average Complexity. Obviously is Apache Tomcat has really many comments, but the reason for this may be simple: The usage of Javadoc leads to much more comments, because the comments get bigger.

The metrics calculated do not differ much from Apache HTTPD. Both are highly active projects. The values calculated can be found in the technical report.

With **Apache Xindice** 1.2m1 we chose an inactive project.

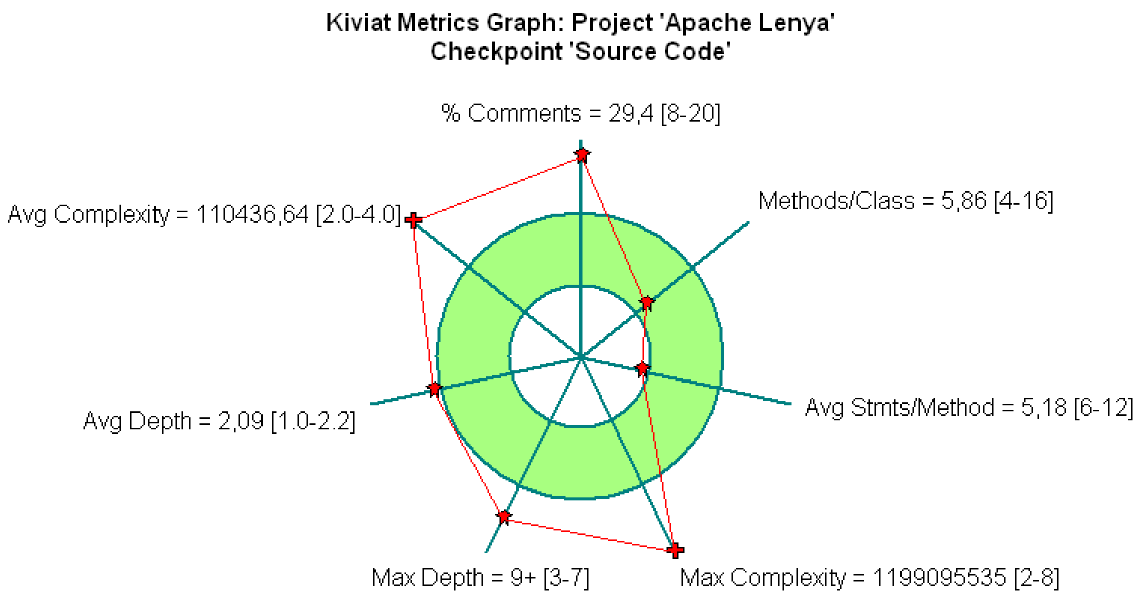
Figure 18 Apache Xindice 1.2m1 Kiviat Metrics Graph by SourceMonitor



Just by comparing this kiviat graph in Figure 18 with the kiviat graph of Apache Tomcat in Figure 17, we do not see many differences. Just some metrics change a little. This means that obviously the metrics Maximum Complexity, Depth of Inheritance Tree and Average Complexity tough measuring relevant quality attributes for longevity do not predict a long healthy software project life. Just Open Hub provides metrics that analyze the activity by measuring commits of contributors.

The second inactive project **Apache Lenya** 2.0.4 shows a similar kiviat graph in Figure 19 like the tow projects before. The project has a huge code base containing a little more than 3500 files resulting in much too high numbers for Maximum Complexity and Average Complexity. Both Apache Xindice and Apache Lenya are retired projects. We used the last available source code for calculating metrics.

Figure 19 Apache Lenya 2.0.4 Kiviat Metrics Graph by SourceMonitor



8.4 Results

The metrics Maximum Complexity, Depth of Inheritance Tree (aka Average Block Depth), Average Complexity, Average Essential Complexity and Lack of Cohesion in Methods measure quality attributes relevant to longevity. The quality attributes relevant for longevity are Testability, Reusability and Modularity [46]. SourceMonitor and Understand appeared to be the best tools for analyzing longevity.

SourceMonitor shows its user with just a few steps relevant metrics for longevity plus shows a kiviat graph interpreting the metrics within a range. It is not possible to specify in detail which source code to analyze. A user could do delete not necessary code before analyzing, for machine interaction separate calls for each subfolder are a way to do the same.

Understand is an even more powerful tool with a very fine documentation. It nominally calculates more metrics, but they can not be interpreted easily. This needs some extra work summing up metric results and calculating average or selecting a maximum value.

Open Hub calculates very interesting values, but only for open source projects and none of them are relevant for longevity.

It is a very good idea to combine several tools, as not all tools calculate all metrics. Also the metric return by different tools may very, depending on the type of metric. This is most likely caused by the different approach of all tools, which and how many source files to analyze.

Feasibility Study Results

The feasibility study shows that the metrics calculated are not wrong at all. But they miss aspects of longevity. The metrics do not cover the whole field of software development and especially usage.

The metrics do show the status quo of a software project. The quality attributes relevant to longevity only measure the longevity related to the source code. Other factors like project activity, feature completeness, documentation or multiple other reasons for choosing a rival project are not covered by metrics.

The feasibility study also showed that answering **RI.1** about the longevity of a software project precisely is not possible, because the frameworks misses metrics.

Limitations of the OSS project selection

It is notable to mention that the majority of papers in the SLR analysed several projects just on a higher level. They investigate communication issues, bug data, or community activities. As a source for these metrics is not only the source code of the project used, but data from bug trackers and mailing lists.

Interestingly only Apache projects are used for detailed investigations, e.g., to investigate health indicators. The reason for this selection is that except ArgoUML no other OSS projects were analyzed by a least two different authors that wrote a paper we found in the SLR.

A **threat to validity** of the framework for longevity evaluation is the number of used metrics. With the selected tools we chose just 14 interesting metrics. This number might be much too small to cover all relevant quality attributes. ISO SQuaRE alone names 55 characteristics. Another threat is that metrics cover only a parts of software.

9 Case Study

In the original concept for evaluation of the framework we also wanted to test, if the framework is capable of predicting the future. This should be done by calculating the metrics of other open source projects. It is also possible to run the tools of the framework for longevity evaluation to all kind of project, also industry projects.

But as the feasibility study already showed that the metrics of the framework fail to predict usage in the future, we did not do a case study.

Generally speaking a case study actually involves the same steps as the feasibility study: Run the tools of the framework on the source code and compare the results.

10 Discussion

It is complicated to link OSS projects, metrics and tools found in the SLR (all of them are linked via a SLR paper) to a quality attribute. The linkage of a quality attribute is established via a metric as its measures (called success criteria [14]) can be connected to a quality attribute (the same as a project attribute [15]). But this link is weak and not easy to recognise.

Like in any another software project, there are plenty of **open issues** and ideas for further improvements.

The first idea addresses the link of quality attributes to metrics. It would be a very interesting and wide field of study with a potential huge impact on the usage of ISO SQuaRE in practice to use the Metrics Guidebook of *Wilbur et al.* [40], who suggest metrics and measures for typical system engineering quality improvement goals, to enhance ISO SQuaRE with an application guidebook.

Another idea for further work on matching metrics to quality attributes is to do an SLR on the quality attributes of ISO SQuaRE using the definition of the attributes and match the metrics that was found in one paper of a quality attribute search to this quality attribute. This will end in multiple connections of one metrics to many quality attributes similar to the McCall quality model, that connects quality factors with quality criteria [30] show in Figure 7, but between quality criteria (quality attributes in ISO SQuaRE) and metrics will be a connection line drawn, i.e. the metrics clearly named.

We definitely see potential in a better traceability of metrics to quality attributes using our approach of evaluating longevity as just one part of the framework. This would lead to a better understanding of the meaning of a metric calculated for a software project. It is also the possibility for tools of being a bigger help to developers showing a complete stack of quality attributes and not only a few selected (like maintainability, reliability, portability, efficiency and security in the tool Kiuwan).

Further Improvement of the framework for longevity evaluation is a consequential next issue. Integrating the proof of concept of this work into a framework from related work or a standalone software tool would be the next step done in future work. And as mentioned in section 5.4: If the software lives on (either as an OSS or commercial project), the quality attributes of longevity are fulfilled fur this software.

Further improvement based on research can to be done on interpreting the selected metrics in Table 10: score the relevance of each quality attribute for longevity by a factor and replace the longevity relevance threshold calculation. A profound method for determining the factor for each quality attribute of ISO SQuaRE needs to be found. This would also be extension of [46].

A research on how the metrics of the tools of the frameworks are calculated would really help to do a better integration of the calculated metrics. So the framework could weight the input metric of each tool, giving a more precise combined metric result.

The intuitive next work package for this framework is to enlarge the tool set of the framework to integrate more metrics and evaluate them. Then, if these enhancements show good results, the following step would be the development a software tool. A big challenge will be to the use of commercial tools then, so it could be a better idea to calculate the metrics with the new developed tool, if possibly using existing tools.

11 Conclusions

As *Spinellis* put it: “Quality, time, and cost are the three central factors determining the success or failure of any software project, and quality is the only one of those factors that can not be changed on the spot by management fiat.” [41] In this work we had a look at the quality as the factor for project health or even longevity. But, with the quote of Spinellis in mind, do not forget that without enough time or too high development costs, the project with the best quality is doomed!

This thesis gave a well-founded view on project health using a Systematic Literature Review (SLR). Also possible metrics and especially tools were evaluated thoroughly and tried to link with quality attributes to make it easier to focus on metrics relevant for longevity.

The selection process for tools is based on a systematic tool evaluation [23]. The most suitable tools were integrated into a framework. For making them work together the provided metrics are interpreted and recalculated if necessary. The result of the framework is a combined list of metrics of all tools.

The feasibility study shows that the metrics calculated return correct values. But they miss aspects of longevity. Right now it is only possible to measure longevity based on the source code. Other factors like project activity, feature completeness, documentation or multiple other reasons for choosing a rival project are not covered by metrics.

One or more metrics for project health or longevity would be an interesting additional feature for every analyzed tool.

12 Appendix

12.1 Literature References

[1] ISO/IEC 25010:2011. “Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - System and software quality models”. International Standards Organisation, 2011.

[2] Wahyudin, Dindin. "Quality Prediction and Evaluation Models for Products and Processes in Distributed Software Development". PhD thesis, Vienna University of Technology, Vienna, 2008.

[3] Mockus, Audris, Roy T. Fielding, and James Herbsleb. "A case study of open source software development: the Apache server." In *Proceedings of the 22nd international conference on Software engineering*, pp. 263-272. Acm, 2000.

[4] Bird, Christian, Alex Gourley, and Prem Devanbu. "Detecting patch submission and acceptance in OSS projects" In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, p. 26. IEEE Computer Society, 2007.

[5] Gall, Harald C., Beat Fluri, and Martin Pinzger. "Change analysis with evolizer and changedistiller." *IEEE Software* 26, no. 1 (2009): 26-33. 2009.

[6] Biffl, Stefan, Wikan Danar Sunindyo, and Thomas Moser. "A Project Monitoring Cockpit Based On Integrating Data Sources in Open Source Software Development." In *SEKE*, pp. 620-627. 2010.

[7] Weissgerber, Peter, Mathias Pohl, and Michael Burch. "Visual data mining in software archives to detect how developers work together." In *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*, pp. 9-9. IEEE, 2007.

[8] Winkler, Dietmar, and Artur Kulmukhametov: ”Systematic Literature Review on Information Systems Longevity“, Technical Report No.: IFS-CDL-13-04, Vienna University of Technology, Vienna, 2013. http://qse.ifs.tuwien.ac.at/publication/IFS-CDL-13-04.pdf (retrieved 2015-11-22)

[9] Proenca, Diogo, Goncalo Antunes, José Luis Borbinha, Artur Caetano, Stefan Biffl, Dietmar Winkler, and Christoph Becker. "Longevity as an Information Systems Design Concern." In *CAiSE Forum*, pp. 73-80. 2013.

[10] Wahyudin, Dindin, Khabib Mustofa, Alexander Schatten, Stefan Biffl, and A. Min Tjoa. "Monitoring the “health” status of open source web-engineering projects." *International Journal of Web Information Systems* 3, no. 1/2 (2007): 116-139. 2007.

[11] Sunindyo, Wikan Danar, Thomas Moser, Dietmar Winkler, and Stefan Biffl. "Analyzing OSS Project Health with Heterogeneous Data Sources." *International Journal of Open Source Software and Processes (IJOSSP)* 3.4 (2011): 1-23. 2011.

[12] Anton, Annie I. "Successful software projects need requirements planning." *Software, IEEE* 20.3 (2003): 44-46. 2003.

[13] Senyard, Anthony, and Martin Michlmayr. "How to have a successful free software project." *Software Engineering Conference, 2004. 11th Asia-Pacific*. IEEE, 2004.

[14] Agarwal, Nitin, and Urvashi Rathod. "Defining 'success' for software projects: An exploratory revelation." *International journal of project management* 24.4 (2006): 358-370. 2006.

[15] Fenton, Norman. "Software measurement: A necessary scientific basis." *Software Engineering, IEEE Transactions on* 20.3 (1994): 199-206. 1994.

[16] Miller, Keith W., Jeffrey Voas, and Tom Costello. "Free and open source software." *It Professional* 12.6 (2010): 14-16. 2010. http://www.upstreme.com/pdf/2010ITPro-OpenSourceEditorial.pdf (retrieved 2015-05-14)

[17] Laurent, Andrew M. St. "Understanding open source and free software licensing." *O'Reilly Media, Inc.*, 2004.

[18] Bachmann, Adrian, and Abraham Bernstein. "Software process data quality and characteristics: a historical view on open and closed source projects." *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. ACM, 2009.

[19] Krishnamurthy, Sandeep. "A managerial overview of open source software." *Business Horizons* 46.5 (2003): 47-56. 2003.

[20] Fielding, Roy T., and Gail Kaiser. "The Apache HTTP server project." *Internet Computing, IEEE* 1.4 (1997): 88-90. 1997.

[21] Vukotic, Aleksa, and James Goodwill. *Apache Tomcat 7*. Apress, 2011.

[22] Alves, Tiago L., Pedro Silva, and Miguel Sales Dias. "Applying ISO/IEC 25010 Standard to prioritize and solve quality issues of automatic ETL processes." *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014.

[23] Poston, Robert M., and Michael P. Sexton. "Evaluating and selecting testing tools." *Software, IEEE* 9.3 (1992): 33-42. 1992.

[24] Royce, Winston W. "Managing the development of large software systems." *proceedings of IEEE WESCON*. Vol. 26. No. 8. 1970. http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf (27.11.09)

[25] Zuser, Wolfgang, Thomas Grechenig, and Monika Köhle. “Software-Engineering: Mit UML und dem Unified Process“. München: Pearson Studium, 2001.

[26] V-Modell XT Authors: *Das V-Modell XT*, 2009. http://www.vmodellxt.de/ (13.12.2009)
In English: http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/1.2.1/Documentation/ (13.12.2009)

[28] Boehm, Barry W.: "A spiral model of software development and enhancement." *ACM SIGSOFT Software Engineering Notes* 11.4 (1986): 14-24. 1986.

[29] IBM Corporation: RUP data sheet, 2007. ftp://ftp.software.ibm.com/software/rational/web/datasheets/RUP_DS.pdf (25.12.2009)

[30] General Electric Company; McCall, Jim A.; Richards, P. K.; Walters, G. F. *Factors in Software Quality: Final Report*. Information Systems Programs, General Electric Company, 1977.

[31] Sommerville, Ian: *Software Engineering;* Reading: Addison-Wesley, 2000.

[32] Elting, Andreas, and Walter Huber. "Immer im Plan? Programmieren zwischen Chaos und Planwirtschaft." *Magazin für Computertechnik (c’t)., Heise Verlag* 2 (2001), pp. 184ff. 2001.

[33] Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas: ”Manifesto for Agile Software Development”, 2001. http://www.agilemanifesto.org (10.01.2010).

[34] Beck, Kent. "Embracing change with extreme programming." *Computer* 32.10 (1999): 70-77. 1999.

[35] Scacchi, Walt. "Process models in software engineering." *Encyclopedia of software engineering*, 2001.

[36] Kitchenham, Barbara, and Shari Lawrence Pfleeger. "Software quality: The elusive target." *IEEE software* 1 (1996): 12-21, 1996.

[37] Stallman, Richard. *Free software, free society: Selected essays of Richard M. Stallman*. Lulu.com, 2002.

[38] Perens, Bruce. "The open source definition." *Open sources: voices from the open source revolution* (1999): 171-188, 1999.

[39] IS0/IEC 9126. “Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use” International Organisation for Standardization, Geneva, 1992.

[40] Wilbur, Ann, Towers, Gayle, Sherman, Tom, Yasukawa, Dan, and Shreve, Sue. *Metrics Guidebook For Integrated Systems And Product Development, International Council on Systems Engineering*. INCOSE-TP-1995-002-01, 1995. http://www.whalen.ws/index_files/MetricsGuidebook_1995-0725.pdf (17.11.2015)

[41] Spinellis, Diomidis. *Code quality: the open source perspective*. Addison Wesley, 2006. ISBN 0-321-16607-8. http://www.spinellis.gr/codequality/intro.html (19.11.2015)

[42] Gousios, Georgios, and Diomidis Spinellis. "Alitheia core: An extensible software quality monitoring platform." *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009.

[43] Almeida, Bruno, et al. "OSSMETER: Automated Measurement and Analysis of Open Source Software." *Projects Showcase@ STAF’15* (2015): 36. 2015.

[44] Pieber, Andreas. “Flexible Engineering Environment Integration for (Software+) Development Teams”, Master Thesis, Vienna University of Technology, Vienna, 2011.

[45] Suman, Manoj Wadhwa. "A Comparative Study of Software Quality Models." *International Journal of Computer Science and Information Technologies* 5.4. 2014.

[46] Winkler, Dietmar, Nathaniel Boisgard, and Bernhard Kiselka: ”BenchmarkDP“, Technical Report, Vienna University of Technology, Vienna, 2015.

[47] Chrissis, Mary Beth, Mike Konrad, and Sandra Shrum. *CMMI for Development: Guidelines for Process Integration and Product Improvement*, 3rd Edition, SEI Series in Software Engineering. Addison-Wesley Professional / Pearson Education, 2011.

[48] Van Loon, Han. *Process Assessment and ISO/IEC 15504: a reference book*. Springer Science & Business Media, 2004.

[49] Poksinska, Bozena, Jens Jörn Dahlgaard, and Marc Antoni. "The state of ISO 9000 certification: a study of Swedish organizations." *The TQM Magazine* 14.5 (2002): 297-306.

12.2 SLR References

References containing metrics:

[M1] Wahyudin, Dindin. "Quality Prediction and Evaluation Models for Products and Processes in Distributed Software Development". 2008.

[M2] Wahyudin, Dindin, Khabib Mustofa, Alexander Schatten, Stefan Biffl, and A. Min Tjoa. "Monitoring the "health" status of open source web-engineering projects." International Journal of Web Information Systems 3, no. 1/2 (2007): 116-139. 2007.

[M3] Wahyudin, Dindin, Alexander Schatten, Khabib Mustofa, Stefan Biffl, and A. Min Tjoa. "Introducing" HEALTH" Perspective in Open Source Web-Enginerring Software Projects Based on Project Data Analysis." In iiWAS, pp. 269-278. 2006.

[M4] Mishra, Birendra, Ashutosh Prasad, and Srinivasan Raghunathan. "Quality and profits under open source versus closed source." ICIS 2002 Proceedings (2002): 32. 2002.

[M5] Mockus, Audris, Roy T. Fielding, and James Herbsleb. "A case study of open source software development: the Apache server." In Proceedings of the 22nd international conference on Software engineering, pp. 263-272. Acm, 2000.

[M6] Mockus, Audris, Roy T. Fielding, and James D. Herbsleb. "Two case studies of open source software development: Apache and Mozilla." ACM Transactions on Software Engineering and Methodology (TOSEM) 11, no. 3 (2002): 309-346. 2002.

[M7] Aberdour, Mark. "Achieving quality in open-source software." Software, IEEE 24, no. 1 (2007): 58-64. 2007.

[M8] Halloran, Timothy J., and William L. Scherlis. "High quality and open source software practices." In Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering. 2002.

[M9] Roberts, Jeffrey A., Il-Horn Hann, and Sandra A. Slaughter. "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects." Management science 52, no. 7 (2006): 984-999. 2006.

[M10] Rigby, Peter C., and Ahmed E. Hassan. "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list." In Proceedings of the Fourth International Workshop on Mining Software Repositories, p. 23. IEEE Computer Society, 2007.

[M11] Bird, Christian, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. "Open borders? immigration in open source projects." In Mining Software

Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on, pp. 6-6. IEEE, 2007.

[M12] Ogawa, Michael, Kwan-Liu Ma, Christian Bird, Premkumar Devanbu, and Alex Gourley. "Visualizing social interaction in open source software projects." In Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on, pp. 25-32. IEEE, 2007.

[M13] Bird, Christian, Alex Gourley, and Prem Devanbu. "Detecting patch submission and acceptance in oss projects." In Proceedings of the Fourth International Workshop on Mining Software Repositories, p. 26. IEEE Computer Society, 2007.

[M14] Bachmann, Adrian, and Abraham Bernstein. "When process data quality affects the number of bugs: Correlations in software engineering datasets." In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, pp. 62-71. IEEE, 2010.

[M15] Linstead, Erik, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. "Sourcerer: mining and searching internet-scale software repositories." Data Mining and Knowledge Discovery 18, no. 2 (2009): 300-336. 2009.

[M16] Gall, Harald C., Beat Fluri, and Martin Pinzger. "Change analysis with evolizer and changedistiller." IEEE Software 26, no. 1 (2009): 26-33. 2009.

[M17] Choi, Namjoo, Indushobha Chengalur-Smith, and Andrew Whitmore. "Managing first impressions of new open source software projects." Software, IEEE 27, no. 6 (2010): 73-77. 2010.

[M18] Wahyudin, Dindin, Rudolf Ramler, and Stefan Biffl. "A framework for defect prediction in specific software project contexts." In Software Engineering Techniques, pp. 261-274. Springer Berlin Heidelberg, 2011.

[M19] Sharma, Vibhu Saujanya, and Vikrant Kaulgud. "Adoption and use of new metrics in a large organization: A case study." In Emerging Trends in Software Metrics (WETSoM), 2013 4th International Workshop on, pp. 21-27. IEEE, 2013.

[M20] Gary, Kevin, and Harry Koehnemann. "The Benefits of Transparency in Managing Software Engineering Capstone Projects." In American Society for Engineering Education. American Society for Engineering Education, 2010.

[M21] Crowston, Kevin, James Howison, and Hala Annabi. "Information systems success in free and open source software development: Theory and measures." Software Process: Improvement and Practice 11, no. 2 (2006): 123-148. 2006.

[M22] Howison, James, Megan Conklin, and Kevin Crowston. "FLOSSmole: A collaborative repository for FLOSS research data and analyses." International Journal of Information Technology and Web Engineering (IJITWE) 1, no. 3 (2006): 17-26. 2006.

[M23] Krishnamurthy, Sandeep. "Cave or community?: An empirical examination of 100 mature open source projects." First Monday (2002). 2002.

[M24] Stewart, Katherine, and Tony Ammeter. "An exploratory study of factors influencing the level of vitality and popularity of open source projects." ICIS 2002 Proceedings (2002): 88. 2002.

[M25] Chow, Tsun, and Dac-Buu Cao. "A survey study of critical success factors in agile software projects." Journal of Systems and Software 81, no. 6 (2008): 961-971. 2008.

[M26] Ossher, Joel, Sushil Bajracharya, Erik Linstead, Pierre Baldi, and Cristina Lopes. "Sourcererdb: An aggregated repository of statically analyzed and cross-linked open source java projects." In Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on, pp. 183-186. IEEE, 2009.

[M27] Biffl, Stefan, Wikan Danar Sunindyo, and Thomas Moser. "Semantic integration of heterogeneous data sources for monitoring frequent-release software projects." In Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on, pp. 360-367. IEEE, 2010.

[M28] Biffl, Stefan, Wikan Danar Sunindyo, and Thomas Moser. "A Project Monitoring Cockpit Based On Integrating Data Sources in Open Source Software Development." In SEKE, pp. 620-627. 2010.

[M29] Wahyudin, Dindin, Alexander Schatten, Dietmar Winkler, and Stefan Biffl. "Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project." In Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on, pp. 229-236. IEEE, 2007.

[M30] Weissgerber, Peter, Mathias Pohl, and Michael Burch. "Visual data mining in software archives to detect how developers work together." In Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on, pp. 9-9. IEEE, 2007.

[M31] McIntosh, Shane, Bram Adams, and Ahmed E. Hassan. "The evolution of ANT build systems." In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, pp. 42-51. IEEE, 2010.

[M32] Jansen, Slinger. "Measuring the health of open source software ecosystems: Beyond the scope of project health." Information and Software Technology 56, no. 11 (2014): 1508-1519. 2014.

[M33 Wahyudin, Dindin, Alexander Schatten, Dietmar Winkler, A. Min Tjoa, and Stefan Biffl. "Defect Prediction using Combined Product and Project Metrics-A Case Study from the Open Source" Apache" MyFaces Project Family." In Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference, pp. 207-215. IEEE, 2008.

[M34] Winkler, Dietmar, Wikan Danar Sunindyo, Stefan Biffl, and Thomas Moser. "Analyzing OSS Project Health with Heterogeneous Data Sources." Open Source Software Dynamics, Processes, and Applications (2013): 207. 2013.

[M35] Fluri, Beat, Emanuel Giger, and Harald C. Gall. "Discovering patterns of change types." In Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on, pp. 463-466. IEEE, 2008.

[M36] Chawla, Sanjay, Bavani Arunasalam, and Joseph Davis. Mining open source software (oss) data using association rules network. Springer Berlin Heidelberg, 2003.

[M37] Stamelos, Ioannis, Lefteris Angelis, Apostolos Oikonomou, and Georgios L. Bleris. "Code quality analysis in open source software development." Information Systems Journal 12, no. 1 (2002): 43-60. 2002.

[M38] Sunindyo, Wikan, Thomas Moser, Dietmar Winkler, and Richard Mordinyi. "Project Progress and Risk Monitoring in Automation Systems Engineering." In Software Quality. Increasing Value in Software and Systems Development, pp. 30-54. Springer Berlin Heidelberg, 2013.

[M39] Wahyudin, Dindin. "Event-based monitoring of open source software projects." *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*. IEEE, 2007.

[M40] Kaltenegger, Andreas. "Deriving Project Health Indicators of Open Source Software Projects using Social Network Analysis." Master Thesis, Vienna University of Technology, Vienna, 2010.

[M41] Nonnen, Jan, and Paul Imhoff. "Identifying knowledge divergence by vocabulary monitoring in software projects." *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. IEEE, 2012.

[M42] Williams, Rose, et al. "Predicting Project Health Prior to Inception." *SEKE*. 2010.

[M43] Shenhar, Aaron J., et al. "Project success: a multidimensional strategic concept." *Long range planning* 34.6 (2001): 699-725. 2001.

[M44] Valverde, Sergi, et al. "Self-organization patterns in wasp and open source communities." *Intelligent Systems, IEEE* 21.2 (2006): 36-40. 2006.

[M45] Li, Paul Luo, Jim Herbsleb, and Mary Shaw. "Forecasting field defect rates using a combined time-based and metrics-based approach: a case study of OpenBSD." *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*. IEEE, 2005.

[M46] Sharma, Vibhu Saujanya, and Vikrant Kaulgud. "Pivot: Project insights and visualization toolkit." *Emerging Trends in Software Metrics (WETSoM), 2012 3rd International Workshop on*. IEEE, 2012.

[M47] Li, Paul Luo, Jim Herbsleb, and Mary Shaw. "Finding predictors of field defects for open source software systems in commonly available data sources: A case study of openbsd." *Software Metrics, 2005. 11th IEEE International Symposium*. IEEE, 2005.

[M48] Gupta, Swastik, and Nilesh Kumar Dokania. "Predicting health of a project using metric generator." *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-*. IEEE, 2014.

References containing OSS projects (additional to the above):

[P1] German, Daniel, and Audris Mockus. "Automating the measurement of open source projects." *Proceedings of the 3rd workshop on open source software engineering*. 2003.

[P2] Sunindyo, Wikan Danar, and Fajar Juang Ekaputra. "OSMF: a framework for OSS process measurement." *Information and Communication Technology*. Springer Berlin Heidelberg, 2013. 71-80. 2013.

[P3] Sunindyo, Wikan Danar, et al. "Improving Open Source Software Process Quality Based on Defect Data Mining." *SWQD*. 2012.

References containing tools (additional to the above):

[T1] Lanza, Michele. "The evolution matrix: Recovering software evolution using software visualization techniques." Proceedings of the 4th international workshop on principles of software evolution. ACM, 2001.

[T2] Rahman, Mohammad Masudur, and Chanchal K. Roy. "An insight into the pull requests of GitHub." Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014.

[T3] Capraro, Maximilian. "Towards a Representative and Diverse Analysis of Issue-Tracker Related Code and Process Metrics." (2013).

[T4] Gotel, Orlena CZ, and Francis T. Marchese. "Scouting Requirements Quality Using Visual Representations." Information Visualisation, 2009 13th International Conference. IEEE, 2009.

12.3 Results of SLR on Health Indicators

In this section we want to give the full details of the SLR on health indicators described in chapter 5.

12.3.1 List of all projects found

While section 5.1 provides only grouped information about the projects found in literature, the full list is shown here in Table 14:

Table 14 Projects found in SLR

No.	Project	Project Type	Reference(s)
1	Apache Ant	Apache	[M28]
2	Apache Cocoon	Apache	[M27], [M28]
3	Apache Excalibur	Apache	[M34]
4	Apache HTTPD	Apache	[M1], [M2], [M3], [M4], [M5], [M6], [M7], [M8], [M9], [M10], [M11], [M12], [M13], [M14]
5	Apache Jakarta	Apache	[M9]
6	Apache Lenya	Apache	[M28], [M34]
7	Apache Log4J	Apache	[M28], [M34]
8	Apache Lucene	Apache	[M39]
9	Apache MyFaces	Apache	[M1], [M29], [M33]
10	Apache Ode	Apache	[M39]
11	Apache OJB	Apache	[M34]

12	Apache OpenJPA	Apache	[M39]
13	Apache POI	Apache	[M28]
14	Apache Roller	Apache	[M39]
15	Apache Slide	Apache	[M1], [M2], [M3]
16	Apache Struts	Apache	[M1]
17	Apache Tomcat	Apache	[M27], [M28], [M1], [M2], [M3], [M29]
18	Apache Woden	Apache	[M39]
19	Apache Xindice	Apache	[M1], [M2], [M3]
20	ArgoUML	Special Purpose Tools	[M16], [M31]
21	Azureus	Special Purpose Tools	[M16]
22	BIND	Special Purpose Tools	[M4]
23	CMS	Special Purpose Tools	[M32]
24	cross-project	cross-project	[M15], [M16], [M17], [M18], [M19], [M20], [M21], [M22], [M23], [M24], [M25], [M26]
25	Cultivate	Special Purpose Tools	[M41]
26	Eclipse	Special Purpose Tools	[M14], [M31]
27	Evolution	Special Purpose Tools	[P1]
28	Fedora Linux	Operating System	[P2]
29	FreeBSD	Operating System	[M4]
30	gcc	Special Purpose Tools	[M8]
31	Ghostscript	Special Purpose Tools	[M4]
32	Gnome	Special Purpose Tools	[M8], [M14], [M32]
33	Gutenprint	Special Purpose Tools	[M40]
34	HSQLDB	DBMS	[M40]
35	Industry	Industry	[M14], [M38], [M42], [M43]

36	JBoss AS	Special Purpose Tools	[M31]
37	jEdit	Special Purpose Tools	[M16], [M35]
38	Jfreechart	Special Purpose Tools	[M16], [M35]
39	Jgit	Special Purpose Tools	[M41]
40	JUnit	Special Purpose Tools	[M30]
41	KDE	Special Purpose Tools	[M8]
42	Linux	Operating System	[M4], [M7], [M8]
43	Mozilla	Special Purpose Tools	[M4], [M8], [M14]
44	MySQL	DBMS	[M13]
45	NetBeans	Special Purpose Tools	[M8], [M14]
46	OpenBSD	Operating System	[M45], [M48]
47	OpenOffice	Special Purpose Tools	[M14]
48	Perl	Programming Language	[M4], [M8]
49	PostGreSQL	DBMS	[M11], [M12], [M13]
50	Python	Programming Language	[M4], [M8], [M11], [M13], [M32]
51	RedHat Enterprise Linux	Operating System	[P3]
52	RedHat Enterprise Linux	Operating System	[P2]
53	Sendmail	Special Purpose Tools	[M4]
54	Squirrel	Special Purpose Tools	[M40]
55	StarOffice	Special Purpose Tools	[M4]
56	SUSE Linux	Operating System	[M37]
57	Xdoclet	Special Purpose Tools	[M40]
58	XFree86	Special Purpose Tools	[M8]

12.3.2 List of all metrics found

While section 5.2 provides only some highlights of the metrics found in literature, the full list of all metrics in alphabetical order is shown here in Table 15:

Table 15 Metrics found in SLR

No.	Metric	Metrics Group	Reference(s)
1	age of project versa number of developers	project success measures	[M23]
2	Average attachments per bug report	activity measures	[M14]
3	Average bug report links per linked bug report	comments measures	[M14]
4	Average bug reporters per developer	defect/quality measures	[M14]
5	Average bug reports per bug reporter	activity measures	[M14]
6	Average bug reports per developer	defect/quality measures	[M14]
7	Average comments per bug report	activity measures	[M14]
8	Average commits per bug report (all bug reports)	defect/quality measures	[M14]
9	Average commits per bug report (only fixed bug reports)	defect/quality measures	[M14]
10	Average commits per developer	defect/quality measures	[M14]
11	Average fixed bug reports per developer	defect/quality measures	[M14]
12	Average length of commit messages (w/o empty)	comments measures	[M14]
13	Average number of bug report status changers per developer	activity measures	[M14]
14	Average number of emails per month	communication measures	[M39]
15	Average relative error; error measure	defect/quality measures	[M18]
16	average size	code size measures	[M37]
17	Average status changes per bug report	activity measures	[M14]
18	Bug History Metric (bug activities during a certain period of time)	defect/quality measures	[M28]

19	Build Graph Depth: The depth of a build in terms of the maximum level of depth references made	code size measures	[M31]
20	Centrality (Degree Centrality, Closeness Centrality, Betweenness Centrality)	communication measures	[M40]
21	change type patterns do describe development activities and affect the control flow, the exception flow, or change the API.	activity measures	[M35]
22	code churn	defect/quality measures	[M19]
23	Code churn (code activity)	activity measures	[M46]
24	code ownership	code size measures	[M5], [M6]
25	Code Quality	defect/quality measures	[M33], [M46]
26	Code Review Effectiveness	project success measures	[M48]
27	comment frequency	comments measures	[M37]
28	communication and use intensity (number of downloads compared to mailing list activity)	communication measures	[M2]
29	Consistency of changes	comments measures	[M16]
30	Control complexity	defect/quality measures	[M33]
31	cyclomatic complexity	code size measures	[M37]
32	Defect Closure Time	defect/quality measures	[M29]
33	Defect Collection Effectiveness	defect/quality measures	[M29]
34	Defect Density	defect/quality measures	[M48]
35	defect density (Post-release Defects/KLOCA)	defect/quality measures	[M5]
36	Defect Detection Frequency	defect/quality measures	[M29]
37	Defect Service Delay (Defect Response Time: Defect Reported Timestamp -	defect/quality measures	[M1]

	Defect Confiremd Timestamp, Defect Removal Time: Defect Confirmed		
38	Design Review Effectiveness	project success measures	[M48]
39	Developer Contribution Pattern (Number of SCM Commits/Number of Email Conversation, Number of Defect Status Changes)	activity measures	[M1]
40	Development Efficiency	project success measures	[M46]
41	direct business and organizational success: Creating a large market share	project success measures	[M43]
42	distribution of the work in the community	communication measures	[M5], [M6]
43	Dynamic Author-File Graph	activity measures	[M30]
44	Dynamic Build Graph Length: The length of a build graph either in terms of the total number of executed tasks or of the total number of executed targets	code size measures	[M31]
45	Dynamic Build Lines of Code (DBLOC): The percentage of code in the build system that is exercised by the default or clean targets	code size measures	[M31]
46	Effort Variance	project success measures	[M48]
47	evolution patterns based on the code structure	defect/quality measures	[M41]
48	Feedback during evolution	communication measures	[M16]
49	File Author Matrix	activity measures	[M30]
50	File Count: The number of specification	code size measures	[M31]

	files in the build system		
51	Halstead Complexity: The quantity of information contained in the build system (Volume), the mental difficulty, associated with understanding the build system specification files (Difficulty), and the weighted Difficulty with respect to Volume (Effort)	code size measures	[M31]
52	impact on the customer: Meeting technical specifications, Fulfilling customer needs, Solving a customer's problem, The customer is using the product, Customer satisfaction	project success measures	[M43]
53	KLOC/time period added	code size measures	[M32]
54	maximum levels	code size measures	[M37]
55	Measures of Centrality: Strength and Out-degree	communication measures	[M44]
56	Modularity	defect/quality measures	[M33]
57	number of active contributors	activity measures	[M32]
58	Number of Adiministrators	activity measures	[M36]
59	Number of CVS commits	activity measures	[M36]
60	number of defects per thousands line of code	defect/quality measures	[M21]
61	Number of developers	activity measures	[M36]
62	number of downloads versa number of developers	project success measures	[M23]
63	Number of forum messages	activity measures	[M36]
64	number of inputs/outputs	code size measures	[M37]
65	Number of mailing lists	activity measures	[M36]
66	Number of patches completed, Number of	defect/quality measures	[M36]

	bugs found, The percentage of bugs fixed, Pct. of support req. Completed		
67	Number of patches started	activity measures	[M36]
68	number of paths	code size measures	[M37]
69	Number of public forums	activity measures	[M36]
70	number of statements	code size measures	[M37]
71	Number of support requests	activity measures	[M36]
72	Number of weekly downloads	project success measures	[M17]
73	page views versa number of developers	project success measures	[M23]
74	Patch Submission Detection, Finding Patch Applications	defect/quality measures	[M13]
75	potential critical success factors of Agile projects: a set of 12 possible critical success factors for each of the four project success categories – Quality, Scope, Time, and Cost	project success measures	[M25]
76	predicting model parameters of software reliability growth models (SRGMs) using metrics-based modeling methods	defect/quality measures	[M45], [M47]
77	predictive algorithm	risk measures	[M42]
78	preparing for the future: Creating a new market, Creating a new product line, Developing a new technology	project success measures	[M43]
79	probability of a fault in a module (Basili et al 1994)	defect/quality measures	[M21]
80	program length	code size measures	[M37]
81	project efficiency: Meeting schedule goal, Meeting budget goal	project success measures	[M43]

82	Proportion of code to comments	comments measures	[M16]
83	Proportion of Verified Solution	defect/quality measures	[M29]
84	proportions of activities in the community	activity measures	[M2], [M34]
85	Quality of Component Testing Effort	defect/quality measures	[M46]
86	Rate of commit messages with bug report links (w/o empty)	comments measures	[M14]
87	Rate of duplicate bug reports	defect/quality measures	[M14]
88	Rate of fixed bug reports	defect/quality measures	[M14]
89	Rate of invalid bug reports	defect/quality measures	[M14]
90	Rate of linked bug reports and Rate of linked bug reports (only fixed bug reports)	comments measures	[M14]
91	Requirement Stability Index	project success measures	[M48]
92	responding speed to bug reports	activity measures	[M21]
93	Rework@Coding	defect/quality measures	[M48]
94	Risk Factor Analysis and Classification	risk measures	[M38]
95	Schecule Variance	project success measures	[M48]
96	service delays on open issues	activity measures	[M2], [M34]
97	size of the Apache development community	code size measures	[M5], [M6]
98	SRS Review Effectiveness	project success measures	[M48]
99	SRS Review Efficiency	project success measures	[M48]
100	Stakeholder Value for Quality Assurance (win conditions)	project success measures	[M29]
101	Static Build Lines of Code (SBLOC): The number of lines of code in build	code size measures	[M31]

	specification files		
102	Target Count: The number of build targets in the build specification files	code size measures	[M31]
103	Target Coverage: The percentage of targets in the build system that are exercised by the default or clean targets	code size measures	[M31]
104	Task Count: The number of tasks in the build specification files	code size measures	[M31]
105	Team Analysis and Composite Graphs	project success measures	[M46]
106	the dynamics of developers' participation	communication measures	[M3]
107	The percentage of bugs fixed	defect/quality measures	[M36]
108	the performance of bug tracking	defect/quality measures	[M3]
109	time to resolve problem reports	defect/quality measures	[M6]
110	Total Communication Metric (No of communication artifacts / time)	communication measures	[M28]
111	Transaction Overview (Number and frequency of the transactions, Number of developers, Number of changed files in one single transaction, Hierarchy-level of the changed file and Sequence of developers that are responsible for the changes)	activity measures	[M30]
112	unconditional jumps	code size measures	[M37]
113	User Coupling Metric (Communication graph based on mailing list)	communication measures	[M28]
114	vocabulary frequency	comments measures	[M37]
115	Volume or size	code size measures	[M33]

12.3.3 List of all tools found

While section 5.3 provides only some highlights of the tools found in literature, the full list of all tools in alphabetical order is shown here in Table 16:

Table 16 Tools found in SLR

No.	Tool	Reference(s)	available	active
1	Apache Jena	[M27]	yes	yes
2	Association Rules Network (ARN)	[M36]	no	?
3	Bug History Collector	[P2]	no	?
4	Bugzilla Query Commands	[M1]	yes	yes
5	Bugzilla Web Service Interface	[P3]	yes	yes
6	c_count	[M47]	yes	yes
7	ChangeDistiller	[M16], [M35]	yes	no
8	Cmetrics	[M47]	yes	no
9	CodeCrawler	[T1]	yes	no
10	Eclipse Checkstyle Plugin	[M1], [M33]	yes	yes
11	Eclipse Metrics plugin	[M1], [M33]	yes	?
12	Engineering Cockpit (EnCo)	[M38]	no	?
13	Engineering Service Bus (EngSB)	[M38]	yes	yes
14	Event Processing Agent	[M39]	no	?
15	Evolizer	[M35]	yes	no
16	FLOSSmole	[M22]	yes	yes
17	Google code search	[M26]	yes	no
18	individual	[M5], [M6], [M45], [M47], [M48]	no	?
19	Initial Delivery Index	[M42]	no	?
20	JIRA Query Commands	[M1]	yes	yes

21	Koders	[M26]	yes	yes
22	Labeled LDA in Java (based on JGibbLDA)	[T2]	yes	no
23	Linguistic Inquiry and Word Count (LIWC)	[M10]	yes	yes
24	Logiscope	[M37]	yes	yes
25	Mailinglist ARChive marc.info	[M2]	yes	yes
26	Merobase	[M26]	no	?
27	Moose	[M26], [T1]	yes	yes
28	Open Hub	[T3]	yes	yes
29	Open Source Ecosystem Health Operationalization	[M32]	no	?
30	Process Mining Tool ProM	[P2], [P3]	yes	yes
31	Project Data Fetcher	[M34]	no	?
32	Project Insights and Visualizations Toolkit (PIVoT)	[M19], [M46]	no	?
33	Project Monitoring Cockpit (ProMonCo)	[M28], [M34]	no	?
34	Protege	[M27]	yes	yes
35	Rational Team Concert/Jazz	[M20]	yes	yes
36	Resource Standard Metrics (RSM) by M Squared Technologies	[M47]	yes	yes
37	RiskIt	[M38]	yes	?
38	SLOCCount	[M8], [M31]	yes	yes
39	SNAnalzyer	[M40]	yes	no
40	SoftChange	[P1]	yes	no
41	Source Monitor by Campwood Software	[M47]	yes	yes

42	Sourcerer	[M15]	yes	yes
43	SPSS	[M28], [M37]	yes	yes
44	StatSVN tool	[M1], [M33]	yes	no
45	SVNKit	[M27]	yes	yes
46	Understand	[M47]	yes	yes
47	Wordle	[T4]	yes	yes

12.4 Results of Tool Evaluation

In this section we want to give the full results of the tool evaluation described in chapter 6.

12.4.1 Pre-selection using Mandatory Requirements

Table 17 shows the full list of all tools passing the pre-selection of the tool study using the mandatory requirements defined in Table 5:

Table 17 Results of the pre-selection

No.	Tool	Result	Reasoning
1	Apache Jena	no	no metrics calculated
2	Bugzilla Query Commands	no	obviously outdated API - no metrics calculated; does not analyse code, but bug data
3	Bugzilla Web Service Interface	no	no metrics calculated; does not analyse code, but bug data
4	c_count	no	does not support java
5	Eclipse Checkstyle plug-in	yes	
6	Engineering Service Bus (EngSB)	no	Engineering Service Bus (EngSB) does not calculate any metrics; it just integrates other tools. It is an alternative to the framework we are building, if it can integrate tools that calculate metrics (i.e. other tools of this list)
7	FLOSSmole	no	no metrics available directly - needs to be

			extracted from the database
8	JIRA Query Commands	no	commercial tool not available for testing, no metrics calculated
9	Koders	no	commercial tool, no metrics calculated
10	Linguistic Inquiry and Word Count (LIWC)	no	no metric of interest calculated
11	Logiscope	no	commercial tool not available for testing
12	Mailinglist ARChive marc.info	no	this tool just searches mailings lists, no metrics calculated
13	Moose	no	no metrics calculated
14	Open Hub	yes	assesses only publically available open source projects
15	Process Mining Tool ProM	no	no metrics calculated
16	Protege	no	no metric of interest calculated
17	Rational Team Concert/Jazz	no	commercial tool test version not working
18	Resource Standard Metrics (RSM)	yes	
19	SLOCCount	no	no export or API available
20	Source Monitor	yes	
21	Sourcerer	no	no metrics available directly - needs to be extracted from the database
22	SPSS	no	no metrics calculated
23	SVNKit	no	no metrics calculated
24	Understand	yes	
25	Wordle	no	no metrics calculated
26	Alitheia-Core	no	project seems dead, no installer/just code exists, metrics limited and API only announced

27	BugzillaMetrics	yes	no metrics calculated; does not analyse code, but bug data
28	CLOC	yes	
29	JArchitect	no	very interesting tool, also plug-ins possible, but no export or API available
30	Kiuwan	no	no export or API available
31	nDepend	no	does not support java
32	ProjectCodeMeter	no	fast tool, export just reports (no API)
33	Sonargraph	no	very interesting tool, many features, no export or API available
34	SonarQube	no	metrics limited, no export or API for metrics
35	SourceMeter	no	no export or API available

12.4.2 Tool Selection Results

Table 18 and Table 19 show the full results of the tool selection. It contains the scales for all pre-selected tools using the selection criteria defined in Table 8:

Table 18 Tool Selection Scales 1/2

No.	**Selection Criteria**	**BugzillaMetrics**	**CLOC**
A	General Requirements		
1	Availability for Testing Purpose	1	1
2	Platform independency: tool runs on Windows	1	0,7
3	Platform independency: tool runs on Linux	1	1
4	Platform independency: tool runs on Mac OS	1	1
5	Time of Installation not longer than 30 minutes	0,5	1
6	Simple Installation using script(s) or installer	0,35	0,35
B	Metrics Calculation		
7	Ability to calculate multiple metrics of at least one	1	1

	metrics group of Table 3		
8	Support of the programming language Java	0,35	1
9	Ability to calculate metrics of both metric groups “code size metrics” and “defect/quality metrics”	0	1
10	Ability to calculate metrics of at least another metrics groups (e.g. “activity metrics” or “communication metrics”)	1	0
11	Support of additional sources (e.g. bug tracker, mailings lists)	1	0
12	Support of evaluating and checking design documents	0	0
C	Export Functionality		
13	Export calculated metrics as XML or another interoperable format via API	1	1
14	Usable User / Developer Guide for the Export / API exists	1	1
15	Support for a client to use the export	0	0
16	Examples available	1	0
17	Coding examples available	0	0,5
Score		**66,175**	**69,125**

Table 19 Tool Selection Scales 2/2

No.	**Eclipse Checkstyle plug-in**	**Open Hub**	**RSM**	**Source Monitor**	**Understand**
A					
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	0	1
4	1	1	1	0	1
5	1	0,7	1	1	1

6	0	0	1	1	1
B					
7	1	1	1	1	1
8	1	1	1	1	1
9	0,35	1	1	1	1
10	0	1	0	0	0
11	0	0	0	0	0
12	0	0	0	0	0
C					
13	0,35	1	1	1	1
14		1	0,7	1	1
15	0	0,35	0	0	0
16	0,35	0,7	0	0,35	1
17	0,5	1	0,5	0,5	1
Score	**53,18**	**75,575**	**70,9**	**69,725**	**77,5**