Evaluating Quality of Open Source Components

Student

Friso Kluitenberg

Supervisors

Dr. A.B.J.M. Wijnhoven

Dr. M. Daneva

Date

January 21, 2018

UNIVERSITY
OF TWENTE.

ABSTRACT

Open source components are a great way for small and medium-sized enterprises to deliver product and services to the market faster. However, challenges arise when assessing the quality of these open-source components. While frameworks and models to assess quality of these components do exist, the open source market is neither governed nor regulated. No language specific or framework specific models or automated tools for analyzing open source software quality exist.

This research aims to solve that by selecting quality indicators of for an open source components on GitHub. In addition, a tool has been developed which evaluates open source components and information about these components from other sources. These sources include Stackexchange for external support and the National Vulnerability and Exposure database for security incident history.

Feedback on the developed prototype shows that developers are interested in an automated way to check for risks which exists in open source components, a judgement of quality and the same analysis for dependencies of such components.

Keywords: OSS Maturity, Risks, Open Source, GitHub, Triangulator

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Background

Increasingly more small and medium-sized enterprises(SMEs) use open source solutions and components in products and services [1]. An open source component effectively allows SMEs to decrease time to market since SMEs do not have to develop their own solution from scratch and thereby avoid reinventing the wheel. In addition, a SME can use a component build on a specific development philosophy or 'opinion'. A development philosophy is a set of structures, rules and guidelines developers can follow to reduce complexity, guarantee quality trough or make best practices easier to follow. In addition, to make use of such components with opinion about best practices development cost can be reduced, helping SMEs serve their customers faster.

1.2 Problem Definition

The explosive growth of open source initiatives results in the available of a wide range of components with different opinions and development philosophies. With this vast amount of available components and solutions to use in a project, risks arise as well. There is no generally accepted market standard within the industry to evaluate component quality. Furthermore, the open source market is neither governed nor regulated. However, even though these structures are not in place, the nature of open source technology allows for transparent comparison and assessment of available solutions.

Therefore, there is a need and a possibility to develop a set of guidelines and an accompanying tool to assess the quality of these components and solutions, find quality indicators and design a tool to evaluate these components. The purpose of this research is to do just that.

1.3 Project goals

As indicated earlier, the master project described in this thesis responds to the need of the software industry for guidelines and tools. More specifically, we set out to achieve the following goals:

1. To develop a methodology for identifying risks for SMEs when using open source components in their development process.
2. To identify suitable indicators for quality evaluation of open source components.
3. To design a tool that supports the quality evaluation of these components, and the associated risk of these components' adoption by SMEs.

1.4 Scope

Open source components are used by a very broad audience, ranging from hobbyist to developers working in bigger enterprises. This translates to a large set of requirements and general recommendations posed to open source components. To reduce these set of requirements and recommendations to a manageable level the scope of this research will be set to small software development companies (SME's). An has less than 250 employees and a turnover of less than €50m or a balance sheet total less than €43m[2].

Software development SMEs specialized in web development work will be considered since these firms rely heavily on open source components. The solutions to be evaluated are limited to common front-end and back-end (PHP) frameworks, solutions and knowledge. These solutions are most often used by software development SME's in the products they release to the market.

These open source components are available through multiple platforms, but in this research only GitHub is considered. The reason for this is that GitHub is the largest open source platform, exposes an Application Programming Interface (API) for easy data consumption and provides a variety of meta-data (which will be explored later) about open source component.

1.5 Research Questions

As stated in section 1.3, the objective of this research is improve the ability to assess quality to identify risks associated with OSS. The goal is to improve both the theoretical understanding about quality and risk in open source projects as well as provide the end-user with a tangible tool to automate evaluating the findings in this tool. To aid in reaching that goal, the following main research question is formulated:

RQ1. Can a tool reliably assess the quality of open source software information?

Quality of open source components is a subject which touches many other disciplines and subjects as well. For example, the legal aspect (licensing) is closely tied to the intended use case. If a project requires distribution for example, a viral license is not always suitable; however if a project includes a service a viral license might be perfectly fine. Thus using and selecting open source components is essentially a knowledge management challenge touching on internal knowledge, solution search and problem solving areas.

In order to show this interdependency clear, a diagram has been constructed. This diagrams shows a typical software development process and how problem solving, solution search and knowledge are interwoven into that development process.

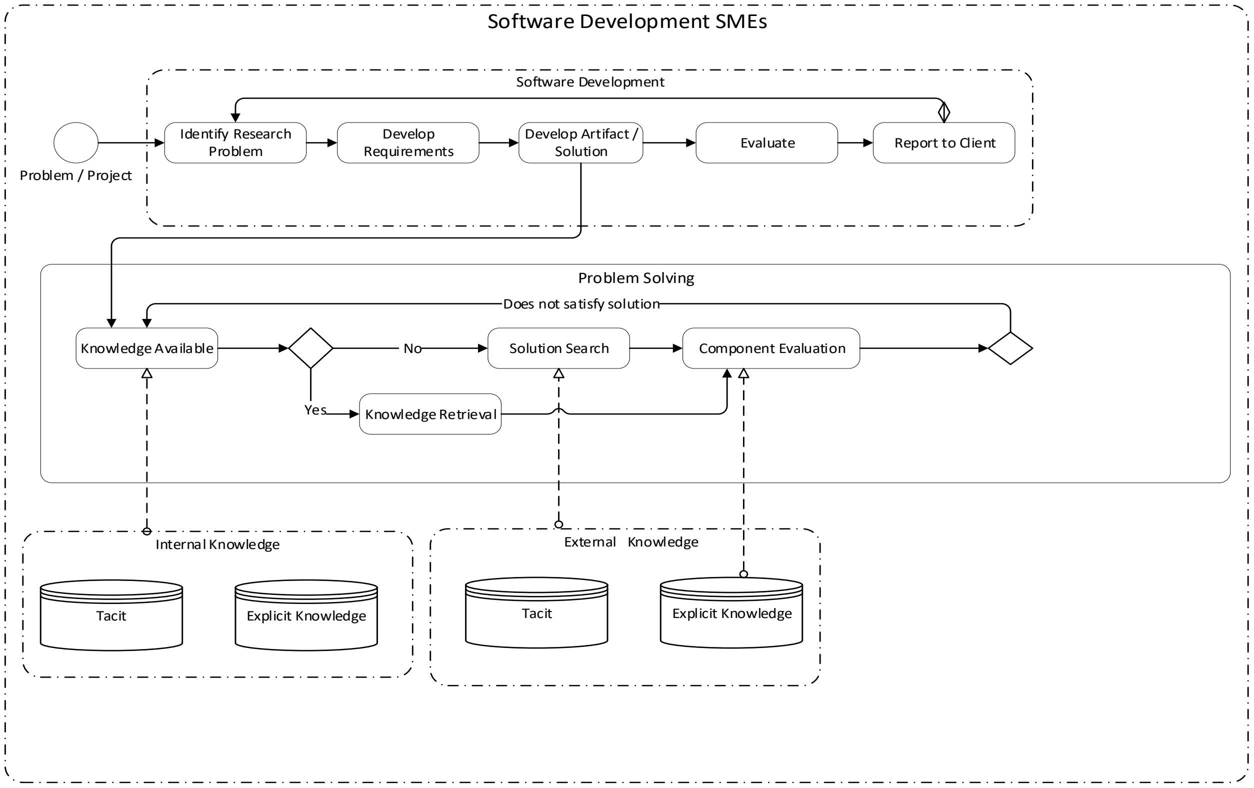


Figure 1 Relationship of Software Development with Knowledge Management

A software development company may start with either a market need or problem they identified or with a need or problem a client identified. Requirements are established for the solution to that problem or need. This process is visualized in figure 1 In addition, these requirements also take constraints of the resources and knowledge of a SME into account. Companies especially SMEs have constraints on their resources, knowledge and budget. These resources and constraints affect the requirements of the solution and thus apply to components used in the solution.

To explore how this context influences constraints and quality requirements of selecting open source the characteristics of small and medium sized enterprises will be researched trough the following question:

RQ 2. What are constrains and opportunities of software development SMEs?

In the development phase, open source components are often required. An open source component is essentially knowledge crystalized in code. In addition, people who are developing, searching for that component or are using that software generate questions, knowledge, ideas for improvement. These are essentially KM and solution search related activities.

In Figure 1, this is represented by explicit knowledge and tacit knowledge. Explicit knowledge is codebase, documentation and support of the open source component. Undocumented ways to implement the component, use the component and potential use-cases are regarded as tacit knowledge in this case. Thus selecting and evaluating the right open source components is a knowledge management problem.

To get a better understanding about how knowledge management influences selecting and evaluating open source components, the following research questions are raised:

RQ 3. How is knowledge created, stored and transferred in SMEs?
RQ 4. What km applications exists?

Open source components are regarded as external knowledge, provided that the same SMEs implementing it didn't develop the component. Open source software is exposed through various platforms, with various characteristics and various online communities discuss about those components, thus providing a valuable source of information.

To explore in which where these open source components and knowledge about these open source components are available, the following research questions are raised

RQ 5. Where do end-users seek information about open source components?

1.6 Characteristics of Small and Medium Sized Enterprises

SMEs differ in numerous ways from larger organizations[3] in both the challenges and the opportunities they face. When looking at the challenge, the following ones can be identified[4]

1. Size related
   An organization with a small number of employees has a risk profile which differs a lot from larger organizations. Common organizational themes like employee sickness, human resource management and retaining available specialized knowledge are all part of that. Specialized

knowledge may exist only in a few key persons of the organization, so it is vital to retain that information or skills and/or these individuals.

2. Insufficient Strategy
   SMEs are more concerned with their daily operational processes than with strategy due to the limited amount of human resources they have available. Business development and strategy in smaller firms are often the output of one person. This is a challenge since business development skills are needed to guide software development activities, matching demand with supply and partnership development[5]. All these activities require organizational support, which may be hard to achieve with this resources. In addition, business development skills take long to develop a deep understanding of the business, understanding of the external market and networks. Thus, if these skills leave the organization it would take long to replace[5], and may result in a weaker competitive position.

3. Resources
   SMEs often have challenges in raising funds. In addition, allocation of assets and procurement of software licenses for components and development tools have to be done more carefully. Allowing custom solutions and software licenses on a per user basis may hurt the company if it has ambitions to grow, since this results in more overhead managing and maintaining these licenses, updates and operating conditions.

There are also benefits small and medium-sized enterprises have over larger enterprises. The main advantages are[4]:

1. Shorter cycle times
   Due to their resource and size constraints, SMEs need to focus the service and products they deliver. They cannot offer a spectrum of products and services that are too broad. This focus results in shorter cycle times.

2. Better client communication
   By being less rigid in their organization's structure, SMEs mainly focus on client interaction. This client oriented focus can result in faster, but less deep, organizational learning and delivering solutions which are more in line with the client's wishes, wants and needs.

3. Less Formal
   A less formal organizational structure and culture allow for faster turn over. Bureaucratic processes are kept to a minimum in small and medium-sized enterprises.

1.7 Knowledge Management

The existence of each company is dependent on the knowledge available within the various departments and functions. Without sufficient knowledge, they will lose their competitive edge and eventually may even cease to exist[6]. Whereas the importance of knowledge storage and knowledge transfer is widely understood and researched, the value of the initial step, namely the creation of knowledge, tends to be sidetracked in small and medium-sized enterprises (SMEs)[4].

More and more companies run their operations in project structures rather than single departments and the importance of working in projects has an influence on how a company deals with the knowledge generated in such settings. This raises interesting questions, especially in Small and medium-sized enterprises, due to their limited size and resources.

Knowledge management can be categorized[4], [7] into three phases:

1. Knowledge Creation
   Knowledge creation entails developing new ideas through socialization, externalization and embedding knowledge in an individual's *'tacit knowledge space'.* Since knowledge creation is in essence learning, unlearning old skills and knowledge is also part of this[4]. Solution search is part of knowledge creation since solution search processes facilitate the creation of new knowledge based on what is already out there.

2. Knowledge Storage
   The process of storing and retrieving all forms of knowledge. Knowledge is dynamic. It enters and leaves the company during the course of its life. To keep a competitive edge, a company must be aware of this and use knowledge storage or knowledge retention strategies and tactics.

3. Knowledge Transfer
   The processes of transferring knowledge, skills and information to other processes of the organization. Essentially, knowledge transfer means that existing knowledge gets reused in a similar or different context.

Whilst, it is debatable whether or not knowledge management phases are really separated[8], much of the literature is organized into these three phases and thus should be taken into account.

1.7.1 Knowledge Creation

Ideally, the knowledge creation process can be seen as a cycle with the following phases[9]:

1. Developing understanding
2. Creating alternatives
3. Finalizing an action
4. Raising awareness
5. Exploring complexities

This cycle resembles a problem-solving process since knowledge is created in response to a specific problem. Knowledge creation and knowledge retention subsequently are difficult for SMEs, since they face challenges in each of these five phases. To address these issues, an SME has to mobilize resources in each of these phases and this resource allocation is costly in smaller firms.

Another important view about knowledge creation comes from Nonaka. He states that creating knowledge is not a sequential process, but a constant 'dialogue' between tacit or skill based knowledge and explicit knowledge[10]. Nonaka states that there are four '*modes of knowledge conversion*' which are iterated in a spiraling manner. The spiral signifies the fact that it is not just a cycle, but with every iteration, knowledge expands and becomes deeper. This spiraling process also applies to software engineering[11]. The interaction between requirement experts, developers' end-users and their feedback results in constantly adjusting requirement or user stories.

1.7.2 Knowledge Storage

Once knowledge has been created, it needs to be retained. Knowledge storage processes are concerned with this task. In order to be of use, knowledge needs to be retrievable and available at the right time. Knowledge storage does not simply entail storing information onto a wiki, database or note application. Various types of knowledge exist and therefore the requirements for storage is very heterogeneous. Tacit knowledge or skill-based knowledge, for example, may be available as manuals, but it is also stored in human assets. People who possess certain skills can also teach other people (retrieval of knowledge from the perspective of the apprentice). This all needs to be taken into account

Binney[12] researched all these KM applications in the literature and developed a KM spectrum to categorize and group these KM applications. Binney found the following categories, together this can be seen as the Knowledge Management Spectrum:

1. Transactional
   In transactional KM, how specific knowledge is used, is codified in the application itself. It can be seen as embedded business logic.
   Movie Recommendation systems are an example of a KM Application based on transactional knowledge. The algorithm which determines which movies to recommend based on the database of movies and a user's conscious or unconscious input can be seen as embedded knowledge.

2. Analytical
   Analytical KM translates data into actionable knowledge. Examples of KM Applications are business intelligence systems, data visualization (Tableau). This category of KM based applications is hard to take advantage of for SMEs, since it requires lots of data to create a significant sample size.

3. Asset management
   Asset management is the management of explicit knowledge and/or intellectual property. In KM management, the word asset is interpreted as broad as possible. For SMEs, examples of these assets are documents, knowledge and/or Code Repositories (GIT).

4. Process Based
   An inefficient process can be a major contributor to information waste. Process based KM is concerned with process improvement and codification of workflows. Process based KM applications are often used when optimizing specific processes.

   Examples of process based KM assets are often lessons learned from '*On-The-Job*' – experience, internal best practices.

5. Developmental
   Developmental KM is concerned with the human KM assets. Development Km is becoming increasingly more important due to the shift of labor intensive work to knowledge intensive work. Examples of development based KM is training, teaching and skill / competence development.

6. Innovation / Creation
   Innovation / Creation KM applications are concerned with creating a platform for new knowledge to emerge. An example of these types of applications are professional groups like LinkedIn groups. Developers can use these groups to create new knowledge based on feedback.

1.7.3 Knowledge Transfer

Knowledge transfer is defined as sharing of knowledge and skills between individuals, units and departments of organizations[13]. Knowledge transfer is indispensable for the competitiveness of SMEs.

Knowledge transfer can be seen in two different contexts: intra-organizational and inter-organizational, also called external knowledge creation.

Szulanski[14] describes knowledge transfer by four phases[14]. The initiation phase plants a 'transfer seed' like a meeting, brainstorm session, interaction with a colleague or 'On the Job' training.

The second stage is Implementation stage. This stage starts with the decision to transfer knowledge. This can either be push or pull so it can be initiated by an individual feeling that he or she requires more information or skills or through company-wide training for example.

The third stage, the ramp-up stage consists of the first day of use of new knowledge. This usage spans some time which leads to the next phase, integration of the knowledge in an individual's skillset or being. This only happens when the results of the new skill or knowledge are perceived as useful or satisfactory.

This process can be visualized by the following diagram:

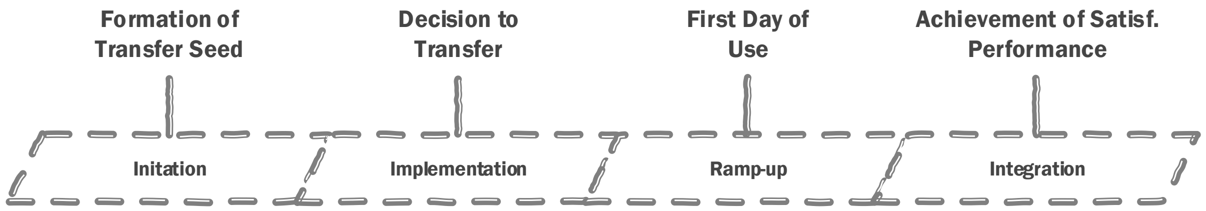


Figure 2 Stages of knowledge transfer

The main challenges with knowledge transfer can be referred to as 'stickiness' or retention rate [14]. These challenges can be mitigated with strategies like peer mentoring, according to Bryant[15].

Peer mentoring can have a profound effect on organizational learning and knowledge retention[15]. Therefore, peer mentoring may be viable to mitigate the lack of knowledge retention in software companies. Peer mentoring often happens informally in organizations, but having structures, tools and processes in place will result in better retention, knowledge and skills.

1.8 Sources of External Knowledge

This section will explore what kind of data is available for free on open source platforms and other platforms which regularly discuss components on these open source platforms.

The main platform that will be used to assess open sources components is GitHub.

1. GitHub
   GitHub is the leading platform for open source components. Whole eco systems of components tap into GitHub as well. An example of this is a project template generator called Yeoman. This piece of software is installable via GitHub, but the templates it generates are also pulled from GitHub. User can add their own templates to that ecosystem relatively easy.

In addition to the main platform which will be evaluated (GitHub), other platforms which contain external knowledge about GitHub exist. The main platforms identified by the author are:

2. Professional Online Communities
   Lot of communities exist online, some are more oriented to professionals and cooperate environments than others. The main difference between this platforms is that discussions are closely moderated and more formal. These professional online communities often provide a base for additional services as well, such as connecting with businesses looking to hire or establish authority in a professional area. In Information Technology, an example of such professional online communities is LinkedIn groups and Stackexchange.com or Medium.com.

   Such communities also offer various other functions. LinkedIn, for example, allows companies to find and reach developers searching for jobs. This can be an effective vehicle to onboard new organizational knowledge and open new opportunities.

3. Open Data
   Open data is all data which large companies, government agencies or other parties have collected and provided for free. Sometimes, a richer or premium data set is available for a small or large fee.

   The main problem for SMEs is that they lack access, resources or expertise to sift through and find meaning in large amounts of open data. The resource requirements of this task can be alleviated by using third parties to analyze the data or interpret the results.

   In order for the information search to be of use, the final objective they try to achieve with the knowledge or intelligence sought after needs to be considered. This main objective is to change action so that better solutions can be provided with lower resources requirements (higher efficiency) for SMEs.

   Examples of questions which can be answered with open data in the context of software development SMEs are:
   Which server or pre-processor versions should be supported?
   What are common problems associated with using certain licenses?

   Which trends are emerging with regards to cloud computing, software architecture?

   Often, tools for open data collection and interpretation are available on GitHub as well. Sometimes, even data dumps of open data or interpreted open data are available.

4. Discussion Boards
   Various discussions board exists online. These discussion boards are monitored and regulated by moderators. Users help each other for points or prestige. Not only are professional solutions

and components found, these discussion board also are ways of customers to form and evaluate buying decisions[16]. Discussion boards exist in all niches you can think off. For example, there are discussion boards with the main focus of evaluating and discussing hosting providers and architecture Webhosting Talk. Numerous forums for entrepreneurs are available as well, Digital Point for example. In addition, for software developers forums like Webmaster Talk exist. The main point is that it is safe to assume that members of an organization consume knowledge from any of this source.

5. SME Network Collaboration
   Although not commonly utilized and shared among SMEs, human capital is an important asset. fear to lose internal knowledge and talent, SMEs still tend to be protective of their intellectual property and disregard the advantages of what collaboration and knowledge sharing can offer. However, when this bridge in perception is crossed and appropriate incentives are put in place, massive business gains can be achieved[17]

   In an SME network – just with any other external knowledge source - when a solution has been found, it needs to be retrieved in a useable form. Knowledge retrieval processes are concerned with this. Information or knowledge retrieval happens within the organization from the '*organizational memory'*. The organizational memory can be defined as the set of tacit and explicit knowledge available to an organization. Information retrieval methodologies address various challenges in retrieving knowledge from the '*organizational memory*'. De Graaf et al[18] identifies the following six main challenges associated with information retrieval.
   1. Document understanding
   2. Locating relevant architectural knowledge
   3. Support for traceability between different entities
   4. Support for change impact analysis
   5. Assessment of design maturity
   6. Credibility of information

   These challenges do apply to searching and evaluating open source projects as well. Open source systems don't always have clear boundaries in use cases they support. For example, as projects grow larger documentation seems to lack behind as well.

Lots of open platforms and data sources exists online, but few lend them self to easy and accurate parsing within the means of this research. Therefore, in the research data collection is constraint to Open Data and sources which expose an API.

2 RESEARCH METHODOLOGY

In order to allow software developers to assess the quality, challenges and implications of using open source component, tools need to be developed. Therefore, this research will be a design science study. The design science methodology/ consists of six steps[19]:

1. Problem Definition & Motivation
2. Objectives of a solution
3. Design & Development
4. Demonstration

5. Evaluation
6. Communication

The problem definition and objectives are already covered in the introduction. The rest of this study will be concerned with designing requirements for a tool which measures open source software quality.

A search of the literature is performed to identify issues and solution and to provide a solid understanding of the state-of-the art in solution search processes, IT Quality Frameworks and open source components. Relevant factors indicating software quality and quality categories from the IT Quality literature will be explored and evaluated for relevance and viability. After establishing a solid foundation, the author designs, develops and validates a tool for assessing quality of open-source components. The design science research model will be used for this[19], since the main goal of DSR is to develop and evaluate new IT artifacts through the following six steps are part of this model:

1. *Problem Definition & Motivation*
   The problem definition and motivation has been explored in this chapter 1. The main problem is that SMEs have limited resources and there are no automatic quality evaluation tools available for front-end and PHP – based projects.

2. *Objectives of a solution*
   The objective of the solution – also called requirement are an integral part of any project delivering a software artifact. The proposed tools main purpose is to automatically evaluate software quality and list any '*red flags*' *it encounters.* This will be addressed in chapter 4.

3. Design & Development
   A tool will be developed according to requirements already explored in this study, the final design and development will be address in chapter 5. The tool should:
   1. provide an interface to search GitHub.com
   2. Evaluate open source software quality accurately as described in the literature review and for each category the literature review deemed relevant.
   3. be able to parse and analyze commit messages
   4. be able to parse and analyze the working directory
   5. be able to parse and analyze issues on GitHub.com
   6. be able to report an quality indication per category

4. *Demonstration*
   The final tool will be available on GitHub as an open source project for anyone to use, analyses and/or improve. Section 5.4 demonstrates the tool, as well as link to a live version and comment on the source code of the developed tool.

5. *Evaluation*
   The developed tool will be evaluated on a number of criteria. Cleven et al[20] describes two ways an artifact can be evaluated. An artifact can be evaluated either internally or externally. First, an internal evaluation will be conducted. In addition, software developers may be contacted for external evaluation. This evaluation will be performed in chapter 7.

The tool will be evaluated on the following two criteria

1. Does the tool provide information that influences a user’s decision making process by either changing or affirming any preconceptions they might have about a specific component?
2. Does the tool provide the end-user with relevant information?

6. Communication

The results of this study, the developed tool will be made public and included in the project's repository on GitHub.com

3 DESIGN THEORY AND META-REQUIREMENTS

The purpose of this chapter is to analyze relevant scientific evidence to support the research questions and main problem definition as has been outlined in the beginning of this thesis. This section follows a top down approach, starting by researching abstract quality methodologies to provide insight into the challenges of measuring quality. Next, the knowledge and insight from those abstract models will be applied to find out which metrics are available to assess an open source component on these more general models. Finally, risks and challenges specific to open source components are researched to find out if there may be risks and challenges missed by traditional models but are relevant to open source components

This approach translates into the following categories to be researched:

1. OS Quality Methodologies
   Various IT quality methodologies exist. These quality methodologies or elements thereof will be explored and described. Furthermore, an evaluation of their potential use cases and insights they provide for open source solutions will be made.

2. GitHub Available Quality Indicators
   Secondly, OS quality indicators will be explored and discovered. These are metrics which open source platforms (in this case in the scope of GitHub) make available. In addition, it will be explored if these indicators can predict problems or advantages described earlier.

3. OS Component Challenges
   Open source development brings a number of challenges and advantages by itself. These challenges and advantages directly affect a project or off the shelf product developed by a commercial software development companies.

For the purpose of literature search Scopus, Web of Science and Google Scholar are queried. Since not all platforms provide automatic term mapping, the following term mapping has been constructed and is used on all platforms.

Main Term	Term Mapping
Software Development Companies	("Software SME" OR "Software Firm" OR "Software Enterprise" OR "software business")
SME	("SME" OR "Small Medium" OR "Small and medium sized" AND ("Firm" OR "Company" OR "Enterprise"))
Problems	("challenges" OR "barriers" OR "problems")
Requirements	(“Needs” OR “demands” OR “request” OR “Matter” OR “Terms” OR “requisites” OR “necessities”)
Solution Search	Solution AND ("Search" OR "Searching" OR "Finding" OR "Discovering")
Solution Search	(“Solution Search” OR “Innovative Search”)
Solution	("Framework" OR "Asset" OR "Component")
Open Source	("Open Source" OR "OSS" OR “FLOSS”)

Tabel 1 Term mapping

The search process resulted in a big number literature sources. However, to stay pragmatic in our literature review, we chose for inclusion only those that we considered highly relevant for this graduation project. “Relevant” means that a literature source directly addresses the topics on the left column in Table 1.

3.1 OS Quality Methodologies

Adewumi et al [21] identifies the most applicable open source quality frameworks. He checked the literature for quality review models and evaluated them and analyzed them on various characteristics. Strengths, features and potential tool support are taken into account by Adewumi. The following frameworks and methodologies are identified and selected:

1. ISO/IEC 25010
2. Cap Gemini OSMM (C-OSMM)
3. Open Business Readiness Rating (O-BRR)
4. Navica Open Source Maturity Model (N-OSMM)
5. Methodology of Qualification and Selection of Open Source software (Q-SOSS)
6. Open Source Maturity Model (OSMM)

3.1.1 ISO/IEC 25010

ISO/IEC 25010[22] is a standard which offers an up-to-date understanding of quality in systems and software projects. The standard describes various quality categories and which element contribute to it. It identifies eight main categories

1. Functional suitability
   This software quality concept how suitable functionality is for specific use cases. Both implied and explicitly stated functionality should match actual functionality. This indicator consists of three criteria: completeness, correctness and appropriateness.

2. Performance efficiency
   Efficiency is the reduction of waste and using the available resource effectively. In software quality efficiency consists of three areas: Time Behavior, Resource Utilization and Capacity. Capacity is defined as the limits of a product relative to what is specified in the requirements of the product.
   The degree which performance efficiency is required from open source components differs per solution implementing the open source component. For example, one time report generation may be less resource efficient than mission critical SaaS applications consumed by lots of clients.

3. Compatibility
   In the software quality context, compatibility is not the ability to work with previous versions of the same software, but how well an open-source component can expose information to other components and systems. Compatibility consists of two areas: Co-existence with other application on the same architecture or system and inter-operability

4. Usability

Every component is designed with a specific purpose or use case in mind – either consciously or subconsciously. Usability is about how well the implementation of this design matches with the expectations of the user. Sub criteria for use ability are the learning curve required to operate the software, aesthetics, error protection and accessibility.

5. Reliability
   Reliability is all about how well a software is able to function failure free. Criteria on which reliability can be measured are how available the software is, can it recover well and fast if an error does happen and how mature is it. If we look at the reliability of an open-source components, the architecture which it runs on must be taken into account as well. At the very least it must be factored out when comparing several open-source solutions on this criteria.

6. Security
   Security is an important issue in software engineering. It applies even more to open-source web-components. Security is concerned with confidentiality, authentication (user authenticity) and accountability.
   The main advantage of open source, also gives rise to security issues. Whilst the community can review the source code, malicious actors can use this to find security holes as well. The knife really cuts both ways In addition, integrating many open-source components introduce risk for the application as a whole.

7. Maintainability
   Maintainability is a very important aspect of open source projects since open-source projects encourage contributions from their users. If a solution is not maintainable, community support will wither, since forking and contributing to project will be a difficult endeavor. Maintainability consists of modularity, reusability, testability and modifiability.

8. Portability
   In essence, portability is the ease in which an open-source component can be used in another context. It consists of three factors, ease of installation, adaptability and replicability. Many open-source web-components are highly portable, since this is the main reason why it has been developed as a component. The most common exceptions are requirements for the architecture it runs on (for example, certain PHP extensions). Luckily, these extensions can be identified easily and installed by the end-use. This should not result in much implementation delay or challenges and is therefore not considered.

Of these eight categories, portability and compatibility will not be considered. Whilst, compatibility and portability are important subjects in software design and development, it gives only few issues when PHP and front-end components are considered. Most of issues in those two categories are caused by server configuration, the cost in both resources and time to adapt to those issue are low in comparison to the development costs of a solution.

In addition, performance efficiency will not be considered. Web application employ caching mechanisms on top of the actual processing, thus reducing the need for highly optimized software in terms of speed and resources usage. Also, a SMEs main goal is to get a product on the market as fast as possible, therefore if there are performance issues arise, other quicker methods to mitigate these issues can be employed instead. An example is to scale up server architecture and optimize an application at a later stage.

3.1.2 Open Business Readiness Rating (O-BRR)

The open business readiness rating (O-BRR) is an open standard with its main aim to measure the quality of open source software. Contrary to other frameworks it's focus is concerned with reliability and testing[23].

However, the website of the author has been suspended, the framework is a bit dated(from 2003) and due to the lack of support by the authors of this framework it won't be considered in the research. In addition, references to the Open BRR white paper are obsolete or broken.

3.1.3 Navica's N-OSMM

Navicasoft has developed its own open source maturity model in 2004[24]. It's a very practical and flexible approach to judge open source software quality.

The model consists of three phases. The first phase is a selecting open source software products to evaluate. How this selection is made is left at a user's own discretion, no support, criteria or guidance is given by the model.

In phase two, a weighting factor is determined based on business needs and in the final phase, a maturity score is calculated with all these factors taken into account. The model describes six categories on which maturity is assessed:

1. Product Software
2. Support
3. Documentation
4. Training
5. Product Integration
6. Professional Services

From these six category, product software, support, documentation and training will be included in the tool's design developed in this research. Professional services are excluded since these are not commonly offered with open source components. Professional services are offered with very large open source projects, as is the case with Redhat Linux, for example.

In addition, OS components are built to be integrated in other systems. Therefore it is safe to assume that all available components and solution the final tool will be evaluate can easily be integrated. In section 1.4, the scope is set to front-end components and back-end PHP components for small and medium-sized enterprises. These scripts can be - inherently to the programming language they are developed in - be integrated with minimal development effort. Any issues with product integration which may arise are less likely the result of the open source component, but more likely caused by tightly coupled code in the main project.

3.2 GitHub Available Quality Indicators

The next step is to find indicators which can be used to assess quality in the categories established in section 3.1. The indicators found will be used as a scale. Since SMEs have different priorities when developing products, the weights of these indicators are open for interpretation and will be user defined. A set of templates will be provided in the tool as well for some guidance.

Open Source project consists of various sources of quality indicators. The project and code file itself can be used to establish quality or the projects metadata can be used. Most often, metadata is generated from an open source project's code base and published. Examples of these are documentation, which is most often generated with PHPDoc or JavaDoc. The following sources can be identified by Kalliamvakou et al[25], Aggarwal et al[26] and the GitHub developer resources[27].

1. Version Control meta data
2. Platform and Community meta data
3. Platform Support & Documentation

3.2.1 Version Control Meta Data

All projects use or should use some form of version control. This enables developers to work together on the same software and keeping track of changes. In version control, extra information used for collaboration is added to code. In this research, the most popular version control system, Git is evaluated. Git exposes the following information that can be used in order to assess software quality:

Branches
A branch diverts the code base from the original version. This enables a developer to make changes based on the main version. Various branching models and naming conventions exist. These are important to make sure the project remains structured, maintainable and bug-free.

Git's best practices dictate that developers should never push code directly to a master-branch. Instead, a pull-request should be made and other developers should review the code before it gets merged.

In addition, features should have its own branch and prefixed as such. Bug fixes and hot fixes should be labeled in this way as well. Branches should be single purposed with low branching activity[28].

Branching is in essence a way for developers to organize collaboration and provide structure to development activities. Project using a good branching strategy will be highly maintainable, since every change, feature and bug-fix is identifiable and merge-able into the main project[29].

Commits
A commit is a change to the project. A commit consists of lines of changed codes. It is basically the difference between the working directory of the previous commit and the working directory. A commit is a message attached to a commit which can be used to analyze what the change set was about.

Commit messages (description of submitted code) should be short and descriptive and not be omitted. In addition, a commit message should indicate if something is fixed, added or changed. Also, the message shouldn't be to abstract, it should describe exactly what was changed. For example, '*added a function*' is not a helpful commit message, '*Added function validateTransaction*' *for example is a better, more descriptive alternative.*

Analyzing commit messages will provide insight into the maintainability category of quality due to exposing the main type of activity a project is current concerned with. Examples of these activities are re-engineering activities, development activities or documentation activities[30].

Tags
A tag is a bookmark for a specific commit. It can represent a release version for example. Tagging exposes some information about the project. For example, how the tag is structured is an important aspect for the package manager.

A recommended tagging system is Semantic versioning for example. Semantic versioning divides a version number up in API-breaking changes, major changes and minor changes like bug-fixes. Changes in one of these three values can be used to detect how well a project is maintained or how likely it is to introduce API-breaking changes.

Tagging is important tool to make a distinction between various milestones and versions of a project. Therefore, a good tagging system will contribute to the maintainability category of quality identified in section 3.1. In addition, the use of semantic versioning will improve the ease of integration, since a developer can anticipate when an upgrade may break how existing code works.

3.2.2 Platform and Community Meta data

In addition to the data a version control system exposes, the website the repository is hosted on provides us with meta-data it collects from its users and developers. In some ways, the website providing the repositories, in this case, GitHub, benefits by ranking high-quality solutions higher in the search results. This will ensure that the developers and users of the website have a better experience as well. Therefore, GitHub collects and tries to predict various factors. In addition, GitHub visualizes data such as a projects '*pulse*'. A metric that indicates how active it is.

Popularity ratings are one of the indicators GitHub provides. Other indicators are:

1. *Stars*
   The number of times developers mark the repository as favorite. The number of stars of a repository is also correlated with how often a component is integrated into an application, according to Borges et al[31], thus an important metric to check popularity and an indicator for the usability category.

2. Watchers
   People who wish to get notified about updates. The reasons why people 'watch' a repository are most often to receive updates about future functionality or bug fixes. According to Sheoran et al[32], 'watchers' are likely to become contributors in the future as well. These contributions are not limited to just providing code, therefore this metric contributes to the support and maintainability category. In addition, Sheoran et al[32] found these contributions extend to

providing documentation for open source components as well, there for the amount of watchers can be used as a metric for documentation aswell.

3. Traffic
   The number of times users visit a repository. This is a measure of overall popularity, but does not conclusively give indications about the quality of an open-source component. The source of the traffic (referrer) however, may indicate the demographics of users of the open source component.

4. Clones
   The number of times users clone or download the content of the repository. A higher number of clones does not indicate a higher number of users per se. It is an indicator of interest in a certain repository, since a user can clone the repository but decide not to use it in the end.

5. Opinions
   Software components need to make choices and assumptions about various topics. If a software is opinionated it forces or strongly suggest a certain way to use the open-source component. This is the result of the software developers' personal opinion about how their software should function. These so called 'opinions' are not always clear to assess, however these opinions are why people like or dislike certain frameworks, so they match closely to peoples personal opinions

3.2.3 Platform Support & Documentation

Support and contribution are another feature that GitHub provides. Open Source project allows third parties to support other users, provide bug-fixes or indicate issues. Repository owners and contributors are marked as such in conversations, pull-requests, and issues, so they carry a higher authority.

Documentation is essential for a project to be reusable by other developers, so these facilities are provided as well. Good support and documentation are not a given for every open source project, therefore it is important to evaluate the amount and depth of documentation and support provided. Therefore, the depth and amount of support and documentation provided will be evaluated trough the ways GitHub Provides. The ways GitHub provides for a developer to expose documentation and for the community to support an open source component are:

1. Link to demos to demonstrate various use cases.
   This helps developers getting started and seeing the benefits of using a certain library. Most often this increases the ease of implanting a library and seeing the benefits. In addition, it helps shorten the learning curve.

2. A Readme.md file in markdown syntax
   A readme files conveys important information regarding the component. It includes most often, the license, contribution guidelines and any opinions the framework might have. We can use natural language processing or a simpler keyword search algorithm to estimate various sentiments conveyed by this file

3. A wiki
   Wikis may be used for larger components to convey more use cases and information regarding the component.

4. GitHub pages using Jekyll CMS
   A GitHub page conveys the same information as a Wiki, but its input is Markdown syntax. Its content is stored under version control (git) as well.

5. The presence of a Package.json file
   A, package.json file is essentially a file with meta-data to the repository. It includes build scripts, dependencies and licensing information. In addition, test scripts are also listed in this file.

3.3 OS Component Risks and Challenges

Through a systematic literature review Moradini et al. [33] identifies various categories of risks associated with Open Source Component Selection. These risks are:

1. Risk of having insufficient quality
2. Component Integration
3. Component operation and maintenance risk
4. Legal Risks
5. Security Risks

The developed tool will identify and evaluate factors contributing to the above risks.

3.3.1 Component Integration

Various risks arise when trying to incorporate open source components in a solution. SMEs need to judge how much effort and resources are required to integrate a component into an existing solution. Misjudging this factors can result in missed deadlines. The ease in which a component can be integrated can be evaluated by the documentation a component exposes.

The following available documents are good indicators that a component lends itself for easy integration with the final product:

1. Well thought of use-cases and example with working demos
   This shows that a developer has put himself in a developer shoes and has a clear understanding of how his component fits into the bigger picture.

2. API – Documentation
   A very specific and systemic API documentation allows for easy integration. In addition, it reduces the learning curve and helps to evaluate a user if the component is useable for his or her specific use cases.

Another type of integration issue is related to deployment. Complex components often require a custom build process. Luckily, with front-end component, distribution files are also provided. In addition, most front-end packages include a build script based on Node Package manager. This allows all dependencies and their correct version to be pulled and compiled.

If the component doesn't need any custom work or adaptation before being used, this build step shouldn't pose any problems. This is often the case since front-end and PHP-backend components often fulfill only one purpose are relatively small. Deployment issues, therefore, are not considered.

However, the following statement do result in an easy component integration.

1. The project contains test cases
2. The projects uses continuous integration

3.3.2 Risk of having insufficient Quality

Another issue is that the final product won't be of sufficient quality. How is sufficient quality defined? Sufficient quality is defined by the requirements of the final deliverable. If selecting a specific component results in these requirements not being met, then the components cannot be used in the final product.

This includes the risk that the component does not meet the criteria for the use case of the implementing solution. In addition, open source components are ever evolving so a component which is a good fit, might not be in the future. Thus it is important to find indicators to establish if the aforementioned change is in progress or if that risk exists.

3.3.3 Component operation and Maintenance Risk

When a component is put in operation, maintenance is needed. As no software is free of bugs and when implementing a component. Lack of support from either the author or from the community will result in a component that is being abandoned and not kept up to date. This is a security risk for the final deliverable.

Another pitfall is technological debt [34], [35]. Technological debt (or code debt) is a situation which occurs when a component or solution has very low entry barriers or implementation barriers, but other challenges arise down the road. For example, if the API of a solution isn't well defined, but the solution is easily integrated. If this is the case and features need to be added, it will be costly for the organization. Technological debt exists when a best practice or a better solution is disregarded in favor of the one which is easier to implement, given time and resource constraints.

The debt usually occurs as problematic when new features or maintenance is required. At some point in time, these changes need to happen and can no longer be avoided. Like any kind of debt, it is just a tool. Just like financial debt can be used to accelerate the growth of a company, technological debt can be used to quickly create working prototypes to ascertain the validity of a market idea. However, it must be paid off in the end. To developers, this translates to refactoring of the system architecture and re-engineering of critical application components.

3.3.4 Legal Risk

The main legal risks of using OS components are in the domain of Intellectual Property (IP). The main concern of enterprise users of open source software is licensing. To better understand these concerns, challenges and requirements with regards to licensing, common licensing models are evaluated [36] [37]. Licenses operate in two ways, by restricting what a user is allowed to do and by requiring what a user must do

Applying a license to software happens for various reasons. The most common reasons to license software are to either:

1. *Reduce liability or protect the author*
   By specifying explicitly that software is delivered as-is and no responsibility for use is accepted, authors try to reduce the potential for litigation. When software doesn't function the way it needs to in specific environments, the open source author cannot be held accountable.

2. *Commercial Exploitation*
   By specifying conditions of use this type of license makes sure that only clients who purchased the software can use it. Without commercial licensing, users are free to buy the software and share it.

3. *Intellectual Property and attribution*
   Licenses provide open source authors the tools to protect their intellectual property and or require attribution when using their libraries.
   This practice is very common for creating creative materials such as images, vectors or banners.

4. *Brand Protection*
   Another important aspect of licensing is that provides the tools to protect the reputation of the brand. Some brand or people are more reserved than others, which means that certain licenses place restrictions on how a solution may be attributed or referred to. This allows the author to enforce the removal of perceived association with software or product of a questionable nature.

Open source projects consist of creative content – images, icons and vectors– and code. These two types of components often have different licenses. The most commonly used licenses will be researched and the intention or philosophy behind the licenses explored .

1. *GNU Public License (GPL) and Derivatives*
   Versions of the GPL license can be considered a viral license. This entails that if a library uses a GPL licensed software, the whole project must be licensed under a GPL compatible license as well. When distributing a GPL-licensed binary, it requires that the source code is available in a non-encrypted form as well. In addition, a user has the freedom – rights explicitly stated by the GPL – that he or she is free to modify or share the product. As imaginable, this requirement is problematic for commercial software as an only one user needs to purchase the software and then he can make it available for free.

   However, these commercial challenges of viral licenses can be circumvented by offering a solution as service. This way, the application itself is not distributed but access is provided via the web.

2. *MIT License*
   The MIT license is generally considered a copy-left license. This means that there are as few restrictions as possible placed on the use of the software.

   The MIT license only places restrictions on liability. The licenser cannot be held responsible for actions of the licensee. If the software is used as-is – which means that it is not a derivative – the MIT license must be included in the distribution and a copyright notice must be distributed as well

   In addition, a developer may

   1. Modify the software
   2. Use it commercially
   3. Sublicense it under a more restrictive license
   4. Use it privately.

3. *BSD Style Licenses*
   These set of licenses are the licenses used in Berkeley Software Distribution, a derivative of the Unix operating system. BSD style licenses come in similar but slightly different flavors. Various clauses are included or excluded based on which BSD License an author uses. However, in each of these flavors, liability is restricted. In addition, it restricts using trade names. The BSD License – with the exception of FreeBSD 0 – Clause- aims to restrict associated with a code author with any other solution. Effectively, the BSD license is a way to protect one's good name or good brand name.

4. *Creative Commons*
   The GPL and MIT style licenses are mainly used for solutions and components containing code. Whilst they can be used for creative content, creative commons licenses are used more frequently for this matter. A software solution frequently contains creative content such as images, vectors, icons and buttons as well.

   The creative commons license aims to give authors control of their products in four different layers:

   1. *Attribution*
      An author is allowed to specific if, when and how they want users of their product to give credit back. This could be with a link back, an included document or even a sound fragment.

   2. *Sharing*
      Specify if, when, how and under which condition a user is allowed to share an author's work. In addition, the right to modify is specified

   3. *Commercial usage*
      Since this license frequently used for works of art, the author can specify whether or not a user of the product may reuse it commercially. This allows an author more control and a tool to enforce the context of the created works.

   4. Derivatives
      In addition to commercial use, derivatives works may not always be wanted by the author. This degree of control allows an author to restrict derivate for either commercial and/or non-commercial use.

5. *Creative Commons - CC0*
   There is one exception: the Creative Commons 0 (CC0) license. The creative commons 0 license is a tool developed to solve a very specific problem. Whilst it may seem simple dedicating a work to the public domain is difficult. This is due to the fact that copyright is

automatically granted. The CC0 allows an author to waive all rights they have by using a license. Essentially, the CC0 license does not impose any restrictions on the end-user but only on the copyright holder. The copyright holder still retains copyright, however an agreement by means of a license is made that he waives his rights.

License violations can occur in various ways. Developers may copy code from professional communities such as LinkedIn Groups or Stack Exchange. Copyright can be violated since it is not clear how this code is licensed or if the individual contributing the solution is the author of the copyright. Contributions by either pull request (code contribution by 3rd parties) or online suggestions can include code from these sources. In addition, these pull requests can be added with malicious intent. However, violating code can easily be removed and replaced if a version control system is used.

3.3.5 Security Risks

In addition to legal risk, security threats are also an important topic in open source software development and usage [38]. The open-source components evaluated in this research are most often integrated in web applications, so they are exposed to the same risks as these web applications. However, due to the nature of open source, risks related to the collaborative aspect and the open aspect of open-source components are added. These three aspects will be touched individually where relevant:

1. Web-Facing Application Risks
   The open-source components in the scope of this research will most likely be accessible via the internet or via the local intranet. Therefore, we need to take these security issues into account. These issue range from Cross-Site-Scripting to SQL Injection attacks.

2. Collaborative Aspect
   Malicious third parties can use – and are using[39] – the collaborative aspect of free open software to spread malware and gain unauthorized access to information systems. For example, an advisory can introduce bugs deliberately or even try to commit a backdoor to the repository.

3. Open Aspect
   Open-source software gives an advisory more insight into possible attack vectors by eposing the source code. The knife cuts both ways since the community can review the code aswell and detect possible security issues.

3.4 Summary and Conclusion

Knowledge management is a very broad topic and it is easy to get lost in all theories the literature presents. This study identified, evaluated and applied theories from knowledge management, software quality and SME business characteristics in the context of open source component selection.

1. *Knowledge Management and Solution Search in software development SMEs*
   Software development companies produce artifacts using the design science research methodology. SMEs face challenges in each of these phases, but the most important ones for

open source solution selection and evaluation is the problem definition and the objectives for a solution. Issues with requirements such as poor stakeholder identification and poor understanding of complex requirement are often a source of failed projects.

2. Software SME Business Characteristics
   SMEs have a lot of advantages when compared to larger firms. Less rigid organizational structures result in an agile and flexible organization. In turn, this allows for a shorter time to market and the ability to change strategic course. However, challenges for a small software company are also numerous. Size related and resource related challenges are inherent to small organizations. In addition, there is not enough human resources to be concerned with organizational strategy. In small companies often only one person is concerned with strategy and the strategy is not always clear for every individual in the organization

3. *OS Quality Methodologies*
   Research has been conducted to see which quality methodologies are available and applicable to open source solutions. The researched frameworks differ on strictness and application area. Navica's Open Source Maturity Model, for example, includes professional services. Training and product software. These practices are more common in larger enterprise.

   In contrast, there is also an overlap of categories which the frameworks evaluate. Overall, the categories defined by ISO/IEC25010 seems to be most relevant. Various categories are excluded because there is not much difference due to the scope set. An example of this is portability, front-end code is by nature highly portable so comparing solutions on this aspect is an exercise in futility.

4. OS Component Problems and Advantages
   Using open source code is not without risk. Generally, four risks can be identified. Components need to integrate into larger solutions. The risk of a misfit exists. In addition, a component may introduce bugs or performance issues. These issues all add up, especially in larger projects. Therefore, a component has maintenance and operation risks associated with it which must be assessed. The open source community must be able to address these issues when they arise or else this will affect the quality of the end product. In the end, open source is a

   Another issue with open source components is legal risk of using components. Code and creative content such as images are intellectual property and is owned by the individuals which produce it. This protection is assigned by default in most countries as stated in the law. Therefore, we need to have licenses which restrict or allow use of this intellectual property. Failure to do so can result in litigation.

5. *Open Source Quality Indicators*
   Open source solutions have multiple sources of information which a tool can process to evaluate the quality of an OS component.

In this research data available through GitHub is evaluated. Data which the platforms provide can be processed by the tool. These include issues, the wiki and pull request. In addition, the version control system used and working directory files will be processed by the tool. Various OS quality frameworks have been identified and evaluated. The following categories from current quality control methodologies have been deemed relevant and will be evaluated with the help of the tool:

A.Maintainability
For maintainability the following indicators are used:
1. Empty commit messages
2. Branching Models
3. The presence of a package.json file
4. Parsing Package.json for build scripts

B.Security
Security issues are being analyzed by the following indicators:
1. Security incident history of the software is analyzed

C. Support
How well a solution is supported by the community can be estimated by checking:
1. How many contributors there are
2. The number of issues opened, closed and the average issue life time
3. Presence of a Wiki

D. *Documentation*
The depth of documentation available will be evaluated by:

1. Parsing the readme file
2. The presence of demo use cases.

4 GOAL & DESIGN PROPOSITIONS OF THE QUALITY ASSESSMENT TOOL

4.1 Goal

Open-source components are a great way to shorten the time to market. However, open- source components do have risks and challenges. This tool's main objective is to help companies in selecting open-source components, taking those risks into account. In addition, the match with a SMEs business requirements will be taken into account.

Moreover, by evaluating all open-source components a user or company produces, expectations can be established in case a company wants to hire that specific developer. The goal of this tool will be to expose useful information about a component with regards to its maturity and risks.

This tool will be designed in such a way that other sources can be added in the future. In this research those 'other sources' will be constraint to public GitHub repository meta data exposed via the website. Example of this are clones, forks and downloads (open source Meta Data).

The tool will process the following types of input:

1. Open Source Component Inputs
   Due to version control system tracking changes, open source components have a lot of data for

us to use. However, to make use of that data, it needs to be interpreted. If we take a closer look at this data it can be separated into various types:

2. Natural Language
   Commit messages, Readme's and documentation are all available as natural language. This makes it hard to interpret it in an automatic fashion, however, it is possible to perform pattern and natural language processing models onto this data. This type of processing will be used for any natural (English) language encountered.

3. Structured Data Files
   In addition to natural language, there are also structured data files available in a repository. Structured data in this context is data which can be interpreted by machines in an exact way. Examples are build scripts and dependencies. These types of data are much easier to process since there available in structured formats (like json or xml) and can be good indicators of quality. When encountering this type of data, a combination of available research and empirical data will be used to evaluate the consequence of this data.

4. Open Source Meta Data
   In addition to data which software developers directly produce, third parties and consumers also generate data indicative of quality. Therefore, these will also be used as an input for the tool. This metadata has been explored in section 4.2. The third party data sources which the tool will consume are support topics on stackexchange.com, security data from the Common Vulnerability and Exposure database and meta-data from GitHub.

4.2 Design Propositions

Considering section 1, the quality of open source components can be an issue in small and medium sized organizations. The first hypothesis will establish if users experience this. The following hypothesis will evaluate that:

> P1: Users are interested in a tool which automatically reviews open source components on risk and quality.

The tool uses a variety of metrics and communicate this to the user to support their decision making about whether to use a specific open source component. The tool assumes that users of the prototype have a certain technical background. To test this the following hypothesis are established:

> P2: Users are able to interpret the quality metrics of an open source component
>
> P3: Users gather new insight by using a tool which automatically reviews open source component quality.
>
> P4: Users change their perception and decision whether or not to use the component after viewing the metrics provided by the prototype.
>
> P5: The metrics used were found to be useful by the end user.

5 DESIGN METHOD

This section develops a methodology and requirements to evaluate the risks and challenges identified in section 3.3, categorizes them according to section 3.1, based on the quality indicators explored in section 3.2 and section 3.3.

The tool will be developed according the Action Research methodology. An action research process is of a cyclical nature, consisting of three phases: input (planning), Transformation (Action) and output (results).

In the first step the problem is evaluated and theorized and a hypotheses is established. This can also be done on the basis of results of the previous cycle. The main output of this step is action planning. In the second phase the action planning is executed. In the last step the data form the previous step is gathered and changes in behavior are measured.

The main hypothesis in our action research process is that we can accurately measure software quality by the input-to-output mapping provided in this section. This mapping or method will consolidate functional requirements with the inputs that an open source component provides. The resulting output will be an evaluation of maturity criteria on each of the in section 4 defined categories. In addition, potential risks defined in section 4 will also be evaluated and brought to the user's attention if the risk is high.

It is important to note that the resulting output of the tool is not a final judgement call for a developer on selecting open source components. The tool will not decide for a user which component is the best in each situation. Rather, it is the goal of this tool to complement the knowledge that the user already has and draw his/her attention to potential issues, pitfalls and establish validity of claims made by the author. Were possible, the tool will link to other sources so that a decision maker or developer can get more information about a potential issue.

5.1 Maturity Criteria for Assessment of Quality

Open source component quality will be assessed on the before mentioned categories, with the following statements:

1. Maintainability
   In order to check maintainability we ascertain the following statements.
   - The OSS Project does not contain empty commit messages
   - The OSS Project contains descriptive commit messages.
   - The OSS project contains a valid Package.json file.
   - The OSS Project contains build scripts (grunt, gulp, phing or other)

2. Security
   Static test of source code files for possible security incidents are hard to perform. There are tools which can test security, they require components to be build and present in a live environment these tools are called 'F*uzzers* or security testers. This depth of security tested is outside the scope of this paper and tool. The tool will use other metrics:
   1. The tool will check security incident security incident data from the National Vulnerability Database to check any history of security vulnerabilities or exposures.

The cross-reference is done by a query on a component's name and will then list all the security incident data including a description available.

3. Support
   the tool will triangulate support data with another Stack exchange, another popular professional community for software developers. The data returned will consists of the amount of questions per day and the date of the last question.

   How well a solution is supported by the community can be estimated by checking.

   1. The tool will check if support is provided on Stackoverflow.com
   2. How many contributors are active on GitHub
   3. The number of issues opened, closed and the average issue life time
   4. Presence of a Wiki

4. Documentation
   First, it is necessary to make a distinction between documentation and support. Documentation differs from support in that support is actively provided by either the author, enthusiast or cooperate employees looking to establish authority in that specific niche.

Documentation is a tool, in this case in the form of information, which allows users to solve challenge they face with the OSS component on their own. This section tends to quickly evaluate the depth of information the documentation provides. The intention of estimation this, is to ascertain if a user will be able to solve most challenges without getting support.

The depth of the documentation will be evaluated by checking the following statements:

1. A Readme file is available.
2. Either a link to external documentation is available in the readme file or the readme file contains a "getting started", "Usage", "API Section" and a section about how to get support.

5.2 Risks Criteria for Assessment of Quality

Components need to be integrated in a whole in order to be useful. When using OSS components, developers need to be aware of any risk they expose themselves to. The risks found in section 3 will be evaluated with the following criteria per category.

5.2.1 Effort Estimation

There needs to be a match between the estimated effort of integrating a component into the final solution and the actual effort. Whilst, the actual effort can only be measured after the fact, it is possible to estimate how easy it is to integrate a component. The following factors influence how well a developer is able to determine implementation efforts required:

The availability of a readme file
if a readme file is available, we can use language patterns to estimate if it contains use cases for the actual use of the software. The closer these use cases are to the use case the developer intends to use the component for, the closer the match.

Package information
A package file (package.json or composer.json) is frequently used to describe the dependencies of an open source component. If this file is available, there should also be a section of build scripts that allow a developer to easily rebuild the software from scratch. This allows a party who is using the component, to easily make little or big adjustments and is common practice.

However, absence of this file can also mean that an component can also be used out of the box. This fact can be ascertained by looking for a 'build' or 'dist' directory

Demo's
the main goal of a readme file is to convey important information about the component, making sure that a developer can get started integrating as soon as possible. Code example and demos are very important tools for this. A demo allows a third party developer to quickly judge it the component is a fit for their project.

The existence of demo's and use cases can be found with natural language processing of the readme files. This will be done by parsing readme files for a series of language and url patterns.

5.2.2 Risk of insufficient quality

Risk of insufficient quality entails that a component may not be suited for an intended use case or indicators are present that this may be the case in the future. Morandini identified contributing factors security risk (covered in section 6.1) and loss of control. Loss of control is an important issue in open source projects. This loss of control manifest itself in OS projects often trough loss of community support. The loss of community support can be measured through search volume on Google Trends[40]. The tool however, cannot parse google trends data automatically metrics of another source (Stackexchange) are used as an indicator leading to the following metric:

1. Developers searching for alternative solutions on external platforms

In addition, risk of insufficient quality lends itself for static code analysis. This is however, out of scope for the application of this tool

5.2.3 Component operation and maintenance

All systems and product are subjected to natural cycles of introduction, growth, maturity and decline. An open source software components is no exception to that[41]. This fact is very important to consider for various reasons. If a component in the declining phase is used, this could lead to increased maintenance or technical debt. Technical debt in this context is when the solution is initially easy to integrate, but leads to more work need in the end. Just as with real debt, this technological debt needs to be paid off in some point of time.

1. The project contains test cases
2. The projects uses continuous integration

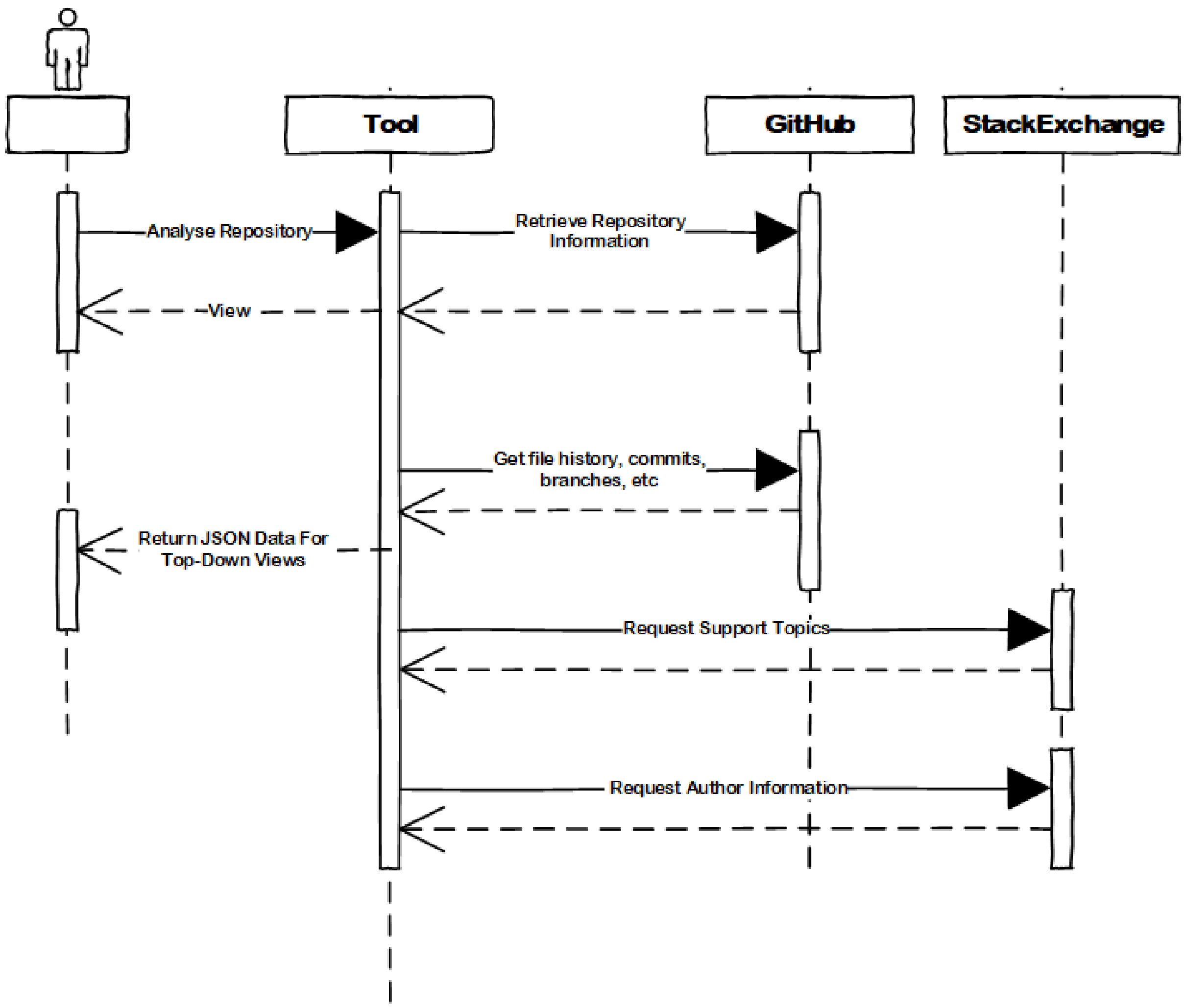
5.2.4 Legal Risks

The tool will help the user identify any legal risk which they might expose themselves to by selecting a component. The main legal risk is when a viral license such a GPL is used. LGPL is not considered a viral license and will be excluded.

1. The tool will notify the user if a component is licensed with a viral license (such as GPL).

5.3 Conceptual Architecture

To illustrate how the tool's components tie together and what the role of the user in the system is, an UML architectural diagram is constructed (Figure 3)



Tool
GitHub
StackExchange
Analyse Repository
Retrieve Repository Information
View
Get file history, commits, branches, etc
Return JSON Data For Top-Down Views
Request Support Topics
Request Author Information

Figure 3 Illustrating user interaction with the tool and the tool interaction with 3rd parties

The user supplies input for the tool in the form of a link to a GitHub repository. In the future this could be automated as a Google Chrome or Firefox extension which provides users that info when they browse for repositories on GitHub.com.

The prototype will than query basic information about a repository, such as name and author from GitHub and return that to the user. After retrieving this main data, the tool retrieves data from all previously defined sources and uses this data to measure OS component on quality and risk. The source of this data and how it contributes to the goal of the prototype is hierarchically displayed in the diagram seen in figure 4.

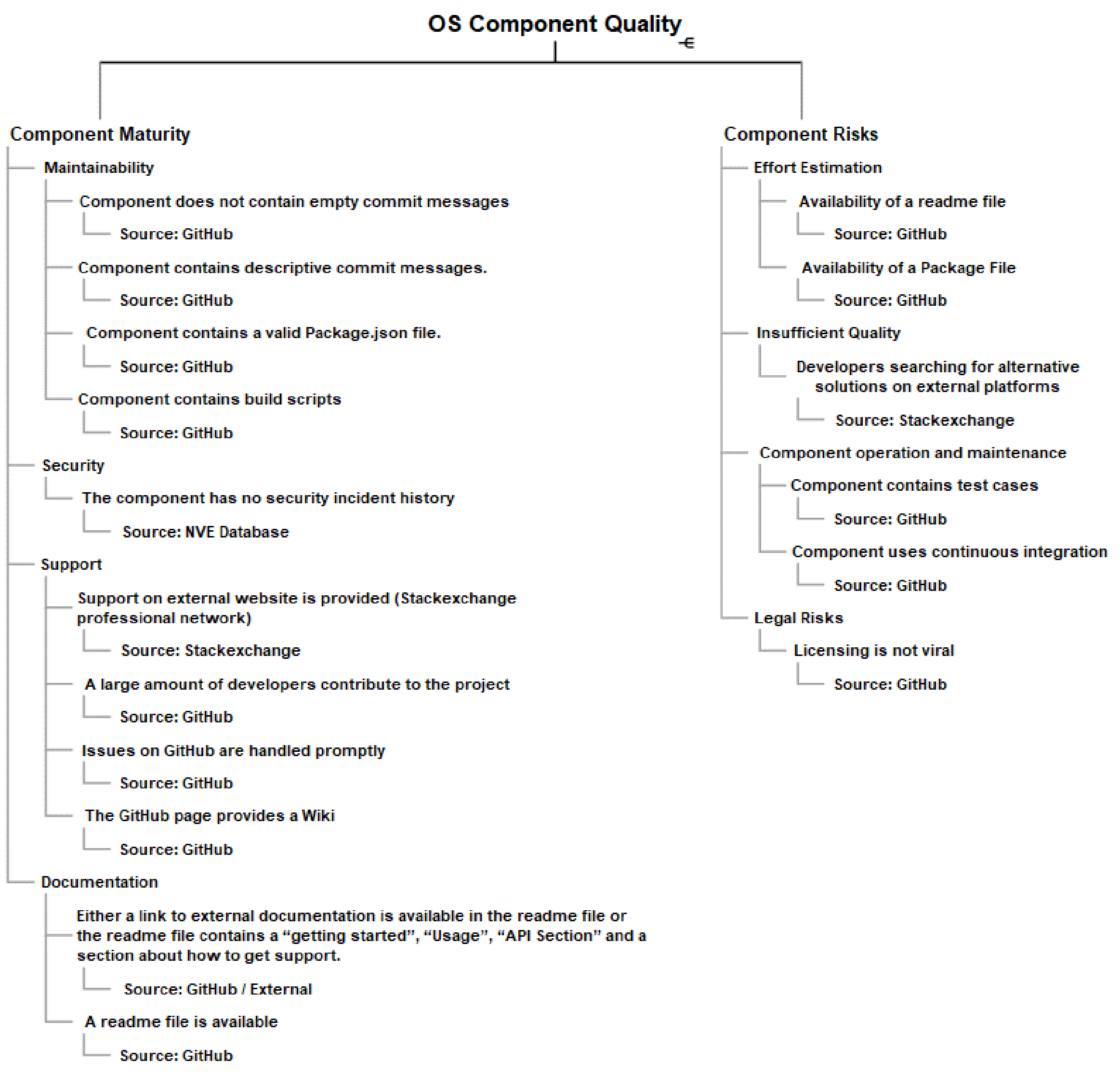

OS Component Quality
Component Maturity
Maintainability
Component does not contain empty commit messages
Source: GitHub
Component contains descriptive commit messages.
Source: GitHub
Component contains a valid Package.json file.
Source: GitHub
Component contains build scripts
Source: GitHub
Security
The component has no security incident history
Source: NVE Database
Support
Support on external website is provided (Stackexchange professional network)
Source: Stackexchange
A large amount of developers contribute to the project
Source: GitHub
Issues on GitHub are handled promptly
Source: GitHub
The GitHub page provides a Wiki
Source: GitHub
Documentation
Either a link to external documentation is available in the readme file or the readme file contains a "getting started", "Usage", "API Section" and a section about how to get support.
Source: GitHub / External
A readme file is available
Source: GitHub
Component Risks
Effort Estimation
Availability of a readme file
Source: GitHub
Availability of a Package File
Source: GitHub
Insufficient Quality
Developers searching for alternative solutions on external platforms
Source: Stackexchange
Component operation and maintenance
Component contains test cases
Source: GitHub
Component uses continuous integration
Source: GitHub
Legal Risks
Licensing is not viral
Source: GitHub

Figure 4 Data sources

For long running background operations or extra data, the tool will send that data in the form of JSON to the client. The client, running in the user's browser will parse and update the view accordingly. One relation which may seem like it's missing in figure 3, is how the tool receives data from the Security Vulnerability Database. This however, is done through a local lookup in a database dump which is downloaded in advance in XML format. The advantage of this, is that we are not reliable upon extra HTTP requests for API calls, however the database does need to be updated regularly.

To keep development time of this prototype as short as possible, a selection of relevant components which the author is proficient in will be used. The following components are used for the development of the prototype:

1. Laravel Model View Controller
   This PHP based model view controller frameworks provides structure out of the box. It will

be used to route user requests between various pages and organize logic in Model, View, Controller and service layers.
https://GitHub.com/laravel/laravel

2. Laravel GitHub
   This extension allows a developer to consume the GitHub API. The tool will be using non-authenticated request (request without oAuth authentication) in order to retrieve data such as commits and tags.
   https://GitHub.com/GrahamCampbell/Laravel-GitHub

3. VueJS
   This library will be used to render views in a reactive way. It integrates well with PHP and Laravel due to its ability to consume JSON.
   https://GitHub.com/vuejs/vue

4. XHTTP
   A small front end JavaScript component for asynchronous JavaScript and xml request. This library will be used to fetch support data from Stackexchange.com and retrieve and renew data without refreshing the page.
   https://GitHub.com/Mitranim/xhttp

In addition, the usual front-end tooling – based on NPM – will be used. The non-exhaustive list of components used are WebPack, Babel and various Babel loaders. By using these open source components, developers can easily extend this tool to include more or other criteria to assess software quality.

5.4 Prototype Design

This section contains details about the implementation of the tool. The final implementation is available via GitLab https://gitlab.com/Arevico/oss-quality-triangulator for anyone to use, review and contribute to. This sections explains the prototype's screens, usage and information the prototype exposes.

The first screen accepts a link to a specific GitHub page and is where the process begins. The user finds component and want to get an assessment about its maturity and risks, he/she will enter the link to it's GitHub page in the input field below.

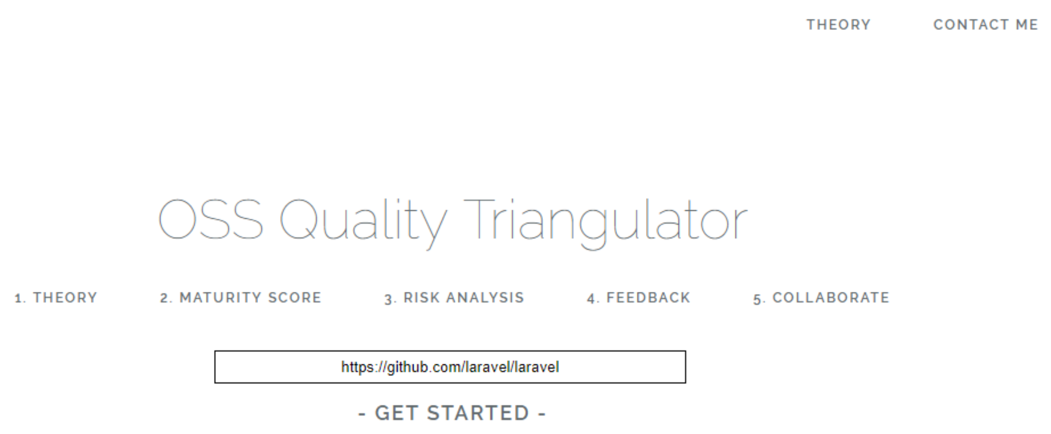


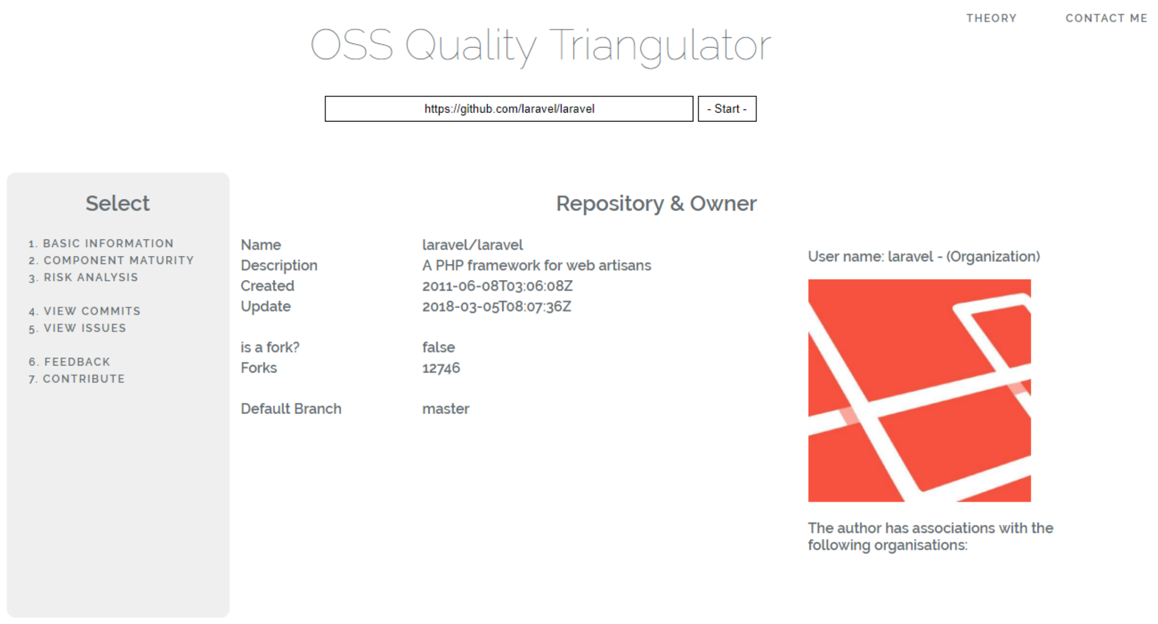
Figure 1 OSS Quality Triangulator: Getting Started

After clicking on 'Getting Started' or pressing enter, the tool will either retrieve earlier retrieved data from the cache or contact GitHub, Stack exchange and the local vulnerabilities database.

After this step, a decision maker will be exposed to some basic information of the component.

5.4.1 Basic Information

First, some general information about the component is retrieved.



Repository & Owner

Name	laravel/laravel
Description	A PHP framework for web artisans
Created	2011-06-08T03:06:08Z
Update	2018-03-05T08:07:36Z
is a fork?	false
Forks	12746
Default Branch	master

User name: laravel - (Organization)

The author has associations with the following organisations:

Figure 2 OSS Triangulator: Basic Information

The decision maker or developer can directly see, when the component was created and how long it has been around. In this example, the Laravel framework is evaluated. The following information is available for the end-user on this screen:

1. The component is created by Laravel (an organization)
   This may indicate that there is a commercial eco system of support, extensions and development available in addition to free support on common platforms.

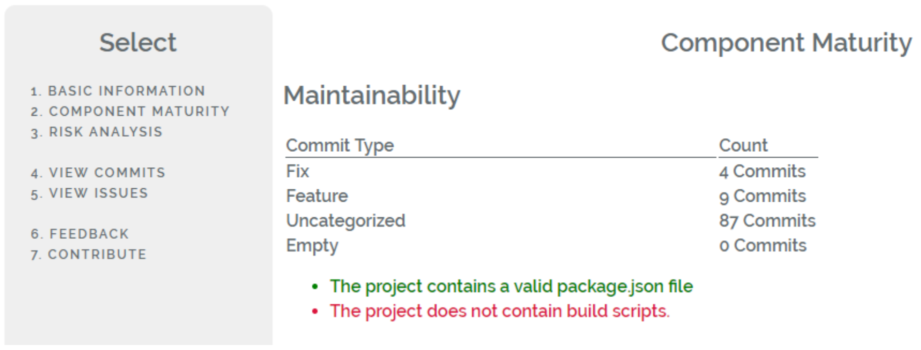
2. Is the project based on another component (a fork)
   This may indicate that the component is tightly coupled to another component and thus is affected by their design decisions.

3. Associations of the author.
   This indicates if the author is associated to any other organization thus exposing any possible conflict of interest and influence.

After taking a look at this information, the user can click through to the next screen: “Component Maturity”.

5.4.2 Component Maturity

This screen provides details for a decision maker or a developer to assess the maturity and which problems might exist.



Component Maturity

Maintainability

Commit Type	Count
Fix	4 Commits
Feature	9 Commits
Uncategorized	87 Commits
Empty	0 Commits

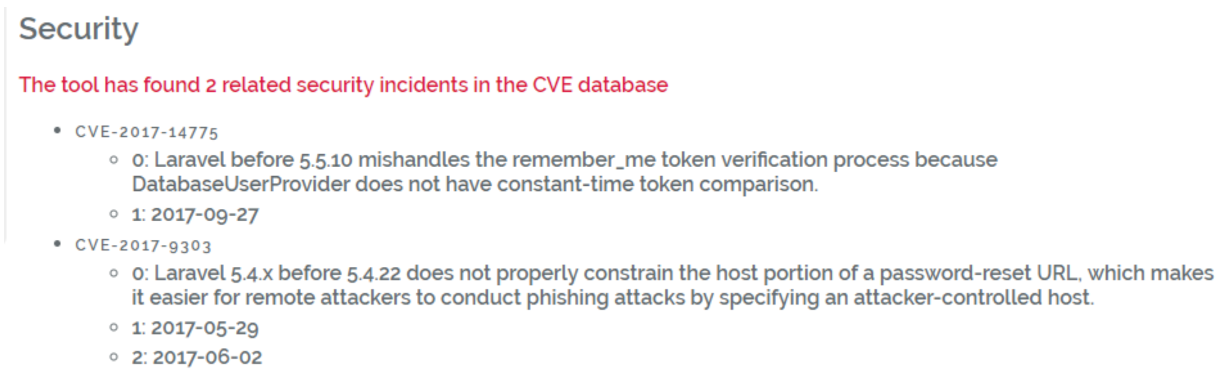
- The project contains a valid package.json file
- The project does not contain build scripts.

Figure 3 Component Maturity: Maintainability

The balance between Fix and feature commits may indicate how stable the software is in terms of, for example API or bugs. In this example there are no empty commit messages, but if there are, it should be considered a big red flag. According to GitHub (and overall) best practices commit messages must not be empty. Code commits without attached description of what is changed may indicate sloppy or low quality development

The next quality factor being assessed is security. It retrieves data from the Common Vulnerability Exposure database to check if there are security incidents. If a decision maker would like to learn more about that issues to that specific issue on the NVE (National Vulnerability and Exposure Database) website.

In this example, the component has two known security issues:



Security

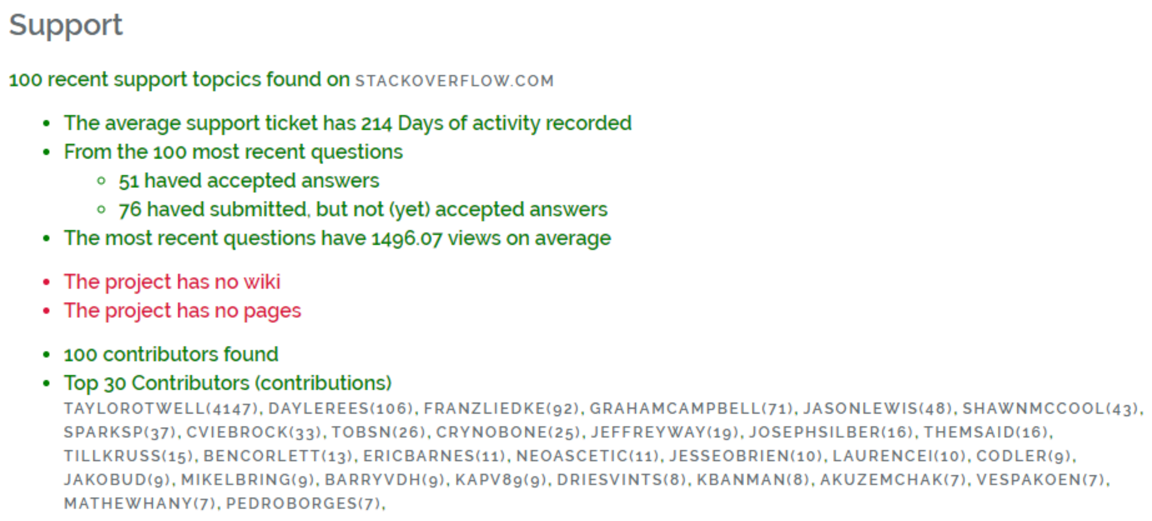
The tool has found 2 related security incidents in the CVE database

- CVE-2017-14775
  - 0: Laravel before 5.5.10 mishandles the remember_me token verification process because DatabaseUserProvider does not have constant-time token comparison.
  - 1: 2017-09-27
- CVE-2017-9303
  - 0: Laravel 5.4.x before 5.4.22 does not properly constrain the host portion of a password-reset URL, which makes it easier for remote attackers to conduct phishing attacks by specifying an attacker-controlled host.
  - 1: 2017-05-29
  - 2: 2017-06-02

Figure 4 Security incidents

For a developer or a decision maker with a technical background, these two issues can be interpreted and used in decision making. However, if this is not the case, a developer or decision maker can click trough and get an impact rating, attack complexity rating and much more information.

In addition to security incidents, a user can also check how well a component is supported on external platform. The following figure is an example of a well-supported component.



Support

100 recent support topcics found on STACKOVERFLOW.COM

- The average support ticket has 214 Days of activity recorded
- From the 100 most recent questions
  - 51 haved accepted answers
  - 76 haved submitted, but not (yet) accepted answers
- The most recent questions have 1496.07 views on average

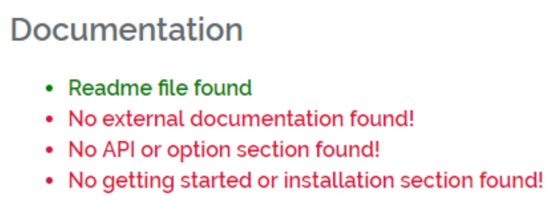
- The project has no wiki
- The project has no pages

- 100 contributors found
- Top 30 Contributors (contributions)
  TAYLOROTWELL(4147), DAYLEREES(106), FRANZLIEDKE(92), GRAHAMCAMPBELL(71), JASONLEWIS(48), SHAWNMCCOOL(43), SPARKSP(37), CVIEBROCK(33), TOBSN(26), CRYNOBONE(25), JEFFREYWAY(19), JOSEPHSILBER(16), THEMSAID(16), TILLKRUSS(15), BENCORLETT(13), ERICBARNES(11), NEOASCETIC(11), JESSEOBRIEN(10), LAURENCEI(10), CODLER(9), JAKOBUD(9), MIKELBRING(9), BARRYVDH(9), KAPV89(9), DRIESVINTS(8), KBANMAN(8), AKUZEMCHAK(7), VESPAKOEN(7), MATHEWHANY(7), PEDROBORGES(7),

Figure 5 Support data

On Stackexchange.com, users can ask questions and other members can answer. The original question author can then select the most useful answer (accept). In addition, a developer can get more information on the top contributors and click trough to their GitHub profile.

Last but not least, the tool parses the Readme file and displays

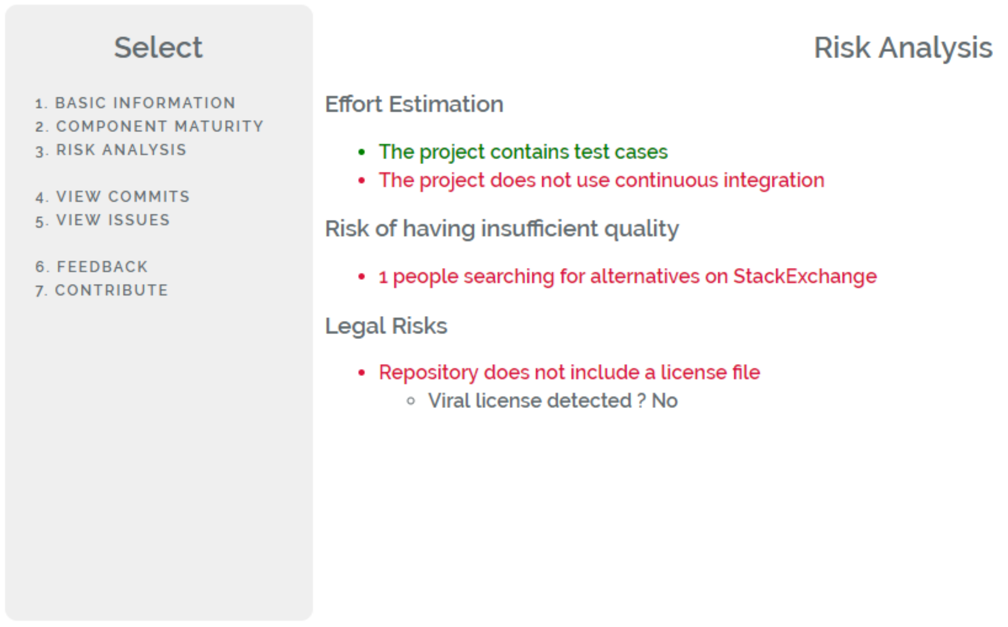

Documentation

- Readme file found
- No external documentation found!
- No API or option section found!
- No getting started or installation section found!

Figure 6 Documentation

5.4.3 Risk Analysis

The second part of the analysis is a risk analysis. All potential issues are colored red, whilst passed tests are colored green.



Risk Analysis

Effort Estimation

- The project contains test cases
- The project does not use continuous integration

Risk of having insufficient quality

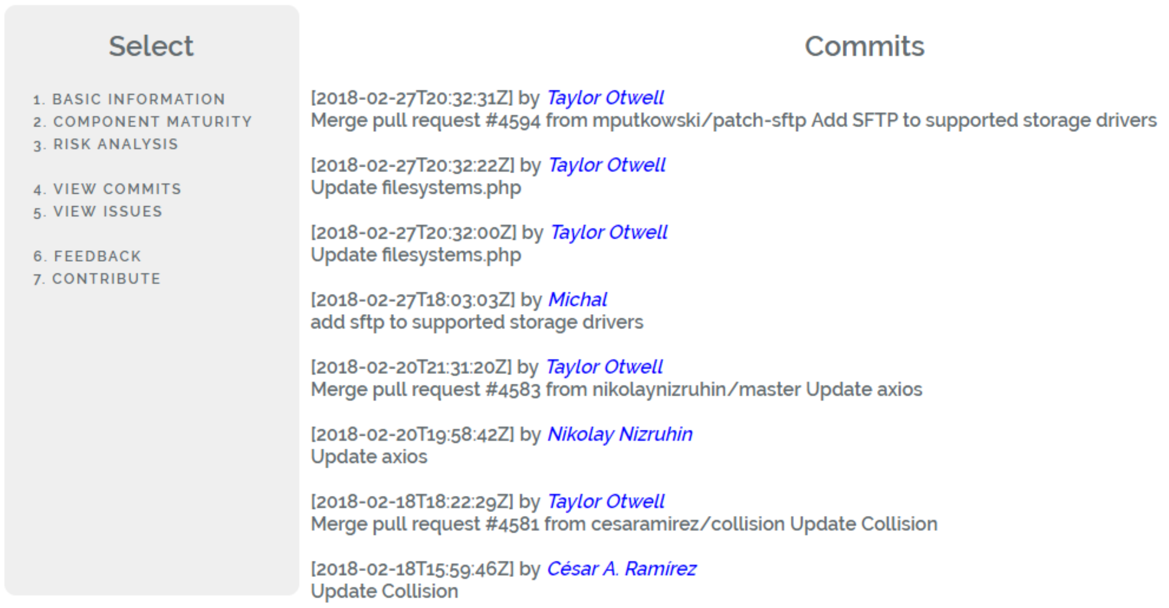
- 1 people searching for alternatives on StackExchange

Legal Risks

- Repository does not include a license file
  - Viral license detected ? No

5.4.4 View Commits

Commit messages provide a good way to get to know what’s going on. This tool makes it easy to view the last commits and their dates. It provides a way for the developer to quickly check what issues the project is fixing or struggling with. This can help in deciding if the component is a good fit for a product or client.



Commits

[2018-02-27T20:32:31Z] by *Taylor Otwell*
Merge pull request #4594 from mputkowski/patch-sftp Add SFTP to supported storage drivers

[2018-02-27T20:32:22Z] by *Taylor Otwell*
Update filesystems.php

[2018-02-27T20:32:00Z] by *Taylor Otwell*
Update filesystems.php

[2018-02-27T18:03:03Z] by *Michal*
add sftp to supported storage drivers

[2018-02-20T21:31:20Z] by *Taylor Otwell*
Merge pull request #4583 from nikolaynizruhin/master Update axios

[2018-02-20T19:58:42Z] by *Nikolay Nizruhin*
Update axios

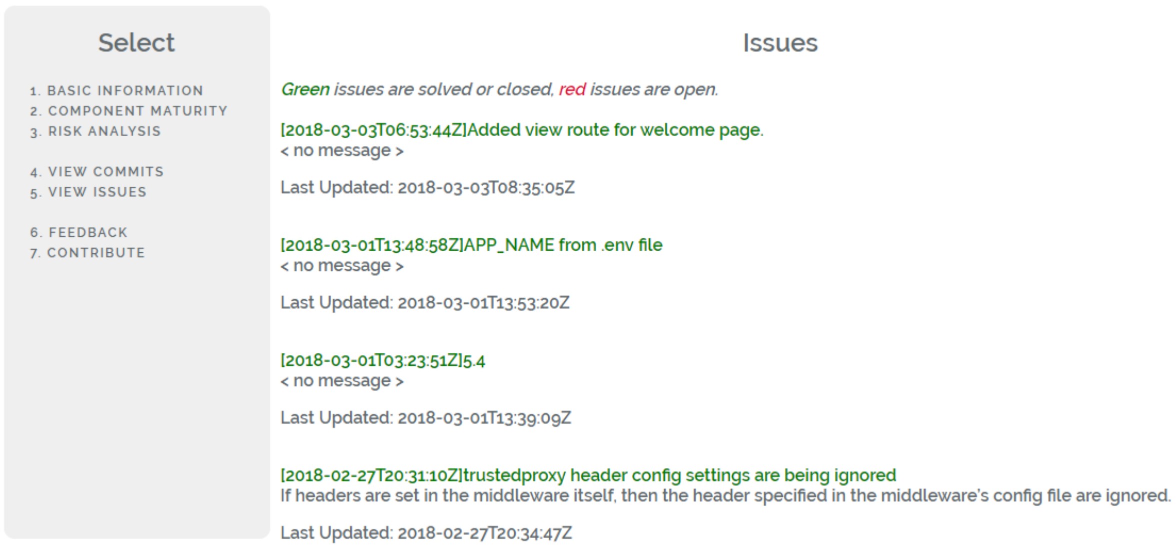
[2018-02-18T18:22:29Z] by *Taylor Otwell*
Merge pull request #4581 from cesaramirez/collision Update Collision

[2018-02-18T15:59:46Z] by *César A. Ramírez*
Update Collision

Figure 7 Most recent commit messages

5.4.5 View Issues

The view issues tab quickly shows the most recent open and closed issues. This can give a developer insight into what issues the community is facing and if the author accepts solutions from other developers and contributors.



Issues

Green issues are solved or closed, red issues are open.

[2018-03-03T06:53:44Z]Added view route for welcome page.
< no message >

Last Updated: 2018-03-03T08:35:05Z

[2018-03-01T13:48:58Z]APP_NAME from .env file
< no message >

Last Updated: 2018-03-01T13:53:20Z

[2018-03-01T03:23:51Z]5.4
< no message >

Last Updated: 2018-03-01T13:39:09Z

[2018-02-27T20:31:10Z]trustedproxy header config settings are being ignored
If headers are set in the middleware itself, then the header specified in the middleware's config file are ignored.

Last Updated: 2018-02-27T20:34:47Z

Figure 8 Issues

The tool will display solved issues in green and open issues in red.

6 RESULTS

An important step in the DSR process is the empirical evaluation of the proposed artefact [42]– in this case, the tool proposed in Chapter 6. To this end, we carried out a first evaluation based on collection of practitioners' feedback. The goal of our evaluation was to understand if the tool meets its goals and if it does what it is supposed to do as described in chapter 4.

Feedback about the tool has been collected through various channels and platforms, such as Stackoverflow, Reddit (Open Source and PHP groups) and LinkedIn groups, as well as direct email. The respondents include mainly freelance developers that have created relevant topics in the professional communities discussed earlier in this research.

The tool collected feedback via the feedback tab which is implemented in the prototype itself and asked the following questions:

1. How useful is the information provided by this tool?
2. Did the tool influence decision about using this component or not?
3. If it did, in which way did it change your decision to use or not use the component?
4. Did your opinion about the component's quality become more negative, positive or neutral?
5. What can we do to improve this tool / are there other things you want to share?

For question 1, we employed a Likert scale. The practitioners' answers can range from 1 to 5, where 1 means "not useful at all", and 5 means "extremely useful". Questions 2, 3, 4 and 5 are qualitative in nature.

6.1 Participants

In total there were ten people leaving quality feedback via the feedback tab in the tool. In addition, three responses were received in response to direct mail or on open communities (Reddit, LinkedIn) instead of people using the feedback form. The results can be seen in Appendix A. Over the feedback period (21 days), 265 unique users accessed the tool. The amount of feedback received can be attributed to various factors. It could be a lack of interest from the target market about software quality, seasonal factors or the feedback form not being displayed in a prominent place. The geographic distribution of users is mainly in the Netherlands, United States and the European Union.

6.2 Data

The users rated the usefulness of the information provided about the component on average at 3.1 on a scale of 5. Seven ratings of three, one rating of four and one rating of 3.5. Four people indicated that the tool influenced their decision about the tool, but they did not or could not provide details what the result of that change is. One person indicated that his/her decision about using the component did not change, but their overall view on the component became more positive

Users indicated that they would like to use the tool to compare various components, instead of just analyzing one. In addition, the tool should work in a recursive manner, looping over all dependencies a project might have analyzing their quality.

In addition, users provided as feedback that static code analysis is a good way to get more in depth knowledge about the quality of a component. Users did not provide feedback about which metrics they are looking for in static code analysis.

7 CONCLUSIONS AND FUTURE WORK

The main problem addressed in this research is the difficulty of assessing quality of open source components due to a market without widely accepted standards or governance. Various conclusions can be drawn about the method used to assess open source software quality. In addition, conclusions can be drawn about how users intended to use the tool and what they had expected.

7.1 Method

The maturity and quality criteria based on the literature review were well received by some users, by others as less useful. The difference might be explained by technical skill of the end-user, but this should be researched in a future study.

Users indicated that they valued static code analysis as an important way to assess code quality. However, no specific metrics were proposed. In order to be able to cater to this, the tool can be revised in the future to consider the language of which a component is coded, the scientific literature about those metrics and integrate with third party static code analysis tools.

7.2 Participants

To summarize, this research can conclude that end users were much more interested in risk factors that may jeopardize their final result than they were in the maturity of the component. The security analysis was well received, but the implications of component maturity were not clear to the end-user.

Moreover, users would like to be able to have all dependencies (components) an open source component uses analyzed a well. A decision should be made on which depth the tool should answer these questions. If this tool is to accommodate these use cases, the tool needs to be reengineered to process a component asynchronously. In the current configuration, the whole assessment is done in a single HTTP request. The above changes require more time to process data thus requiring the tool to make its assessment with or without an active TCP connection. Support for this is already built-in by using XHR technology (described in section 4.4 conceptual architecture).

7.3 Review of Solution

In this section, the main research question in section 1.5 and the design propositions established in chapter 5 are evaluated based on the result gathered from user interaction with the prototype.

The main research question is of this research is:

> Can a tool reliably assess the quality of open source software information?

Through the feedback from the participants the following conclusions can be drawn. Some data was off due to the type of search used. An example of this is that security data sometimes included irrelevant component. This can be mitigated by letting the user exclude these components or applying better to natural language processing and 'fuzzy' search.

Users commented that the tool did provide good data collection, but the judgement is left to the end-user. An example of proposed interpretation is a project health charts, based on the component's contributors. A time dimension of this metric would help the user to establish the quality in a more reliable way.

The following design propositions were established and are evaluated:

P1: Users are interested in a tool which automatically reviews open source components on risk and quality.

The interest in a tool which automatically evaluates quality of open source components (P1) can be confirmed by the responses of users trying out the prototype. Users indicated that they learned new things from the tool and found the way the prototype processes data from multiple sources useful.

Open source components often depend on other open source components. For example, a Content Management or SaaS library may depend on some front-end component or debugging utility. Users would also like to be able to use the quality assessment tool to iterate over all dependencies and evaluate the quality of a component's dependencies, not just the main component.

P2: Users are able to interpret the quality metrics of an open source component.
Some users were able to interpret the significance for their final product, but most users would like the tool provide some interpretation as well. Linking to the theory behind the tool was not considered enough, so the implications of failing and passing certain metrics needs to be included in a future iteration of the tool.

P3: Users change their perception and decision whether or not to use the component after viewing the metrics provided by the prototype.

Some components had serious security exposures in the past, even though the user acknowledge this it didn't change their decision making process. A use case which was not predicted, is that a user would like to compare similar components in the same overview. This way, they can compare how components for the same use stack up against each other.

P4: The metrics used were found to be useful by the end user.

The security incident data was well perceived. The opinions about risk data was divided. One user indicated that he would like to have it compared to a benchmark.

Users provided as feedback that static code analysis is a good way to get more in depth knowledge about the quality of a component. To incorporate this in the prototype, a developers need to research predictors for code quality in on various programming language. In addition. Users did not provide feedback about which metrics they are looking for in static code analysis.

7.4 Future work and Implications

Recommendations of future research with regards to the constructed model are twofold. First, the implications of maturity on component quality should be further researched and communicated to the user. Secondly, more risk factors should be explored and evaluated and potential impact of those risks should be communicated to the end-user if the component is at risk. This communication can either be done through a conclusions sections – although definitive conclusions are hard to draw or

The scope was restricted to SME's but it appears that all sorts of users are using open source solutions, the necessity of this scope is debatable. Future iterations of this tool should include questions to establish the difference between various types of users to check if the answer to the established hypotheses differs between users within SME’s and other user types.

This research has shown that there is a need and market demand to assess the quality for open source software quality automatically.

In addition, this open source tool can provide a base for researchers and interested individuals to expand on with more quality and risk indicators.

REFERENCES

[1] J. Dedrick and J. West, “Why firms adopt open source platforms: a grounded theory of innovation and standards adoption,” *MISQ Spec. Issue Work. Stand. Mak. A Crit. Res. Front. Inf. Syst.*, no. April, pp. 236–257, 2003.

[2] European Commision, “COMMISSION STAFF WORKING DOCUMENT on the implementation of Commission Recommendation of 6 May 2003 concerning the definition of micro, small and medium-sized enterprises,” 2003. [Online]. Available: http://ec.europa.eu/DocsRoom/documents/10033/attachments/1/translations.

[3] C. R. Lane, D. R. Pearson, and S. L. Aranoff, “Small and Medium- Sized Enterprises : Characteristics and Performance,” no. 332, 2010.

[4] S. Durst and I. Runar Edvardsson, “Knowledge management in SMEs: a literature review,” *J. Knowl. Manag.*, vol. 16, no. 6, pp. 879–903, 2012.

[5] C. H. Davis and E. Sun, “Business development capabilities in information technology SMEs in a regional economy: An exploratory study,” *J. Technol. Transf.*, vol. 31, no. 1, pp. 145–161, 2006.

[6] R. Reagans and B. McEvily, “Contradictory or compatible? reconsidering the ‘trade-off’ between brokerage and closure on knowledge sharing,” *Advances in Strategic Management*, vol. 25. pp. 275–313, 2008.

[7] B. D. Janz and P. Prasarnphanich, “Understanding Knowledge Creation, Transfer, and Application: Investigating Cooperative, Autonomous Systems Development Teams,” *Syst. Sci. 2005. HICSS ’05. Proc. 38th Annu. Hawaii Int. Conf. 03-06 Jan. 2005 pages 248a \r- 248a*, vol. 0, no. C, pp. 1–10, 2005.

[8] F. Wijnhoven, “Knowledge management: More than a buzzword,” in *Knowledge Integration: The Practice of Knowledge Management in Small and Medium Enterprises*, 2006, pp. 1–16.

[9] J. C. Riedel, M. Sakiroglu, P. Johal, and K. S. Pawar, “Knowledge creation in physical and virtually collocated product development teams,” in *2007 IEEE International Technology Management Conference, ICE 2007*, 2007.

[10] I. Nonaka, “A Dynamic Theory Knowledge of Organizational Creation,” *Organ. Sci.*, vol. 5, no. 1, pp. 14–37, 1994.

[11] J. Wan, H. Zhang, D. Wan, and D. Huang, “Research on Knowledge Creation in Software Requirement Development,” *J. Softw. Eng. Appl.*, vol. 3, no. 5, pp. 487–494, 2010.

[12] D. Binney, “The knowledge management spectrum – understanding the KM landscape,” *J. Knowl. Manag.*, vol. 5, no. 1, pp. 33–42, 2001.

[13] L. Argote and P. Ingram, “Knowledge Transfer: A Basis for Competitive Advantage in Firms,” *Organ. Behav. Hum. Decis. Process.*, vol. 82, no. 1, pp. 150–169, 2000.

[14] G. Szulanski, “The Process of Knowledge Transfer: A Diachronic Analysis of Stickiness,” *Organ. Behav. Hum. Decis. Process.*, vol. 82, no. 1, pp. 9–27, 2000.

[15] S. E. Bryant, “The Impact of Peer Mentoring on Organizational Knowledge Creation and Sharing: An Empirical Study in a Software Firm,” *Gr. Organ. Manag.*, vol. 30, no. 3, pp. 319–338, 2005.

[16] M. K. O. Lee, C. M. K. Cheung, K. H. Lim, and C. Ling Sia, “Understanding customer knowledge sharing in web-based discussion boards,” *Internet Res.*, vol. 16, no. 3, pp. 289–303, 2006.

[17] K. Möller and S. Svahn, “Crossing East-West boundaries: Knowledge sharing in intercultural business networks,” *Ind. Mark. Manag.*, vol. 33, no. 3, pp. 219–228, 2004.

[18] K. A. De Graaf, P. Liang, A. Tang, and H. Van Vliet, “How organisation of architecture documentation affects architectural knowledge retrieval,” *Sci. Comput. Program.*, vol. 121, pp. 75–99, 2016.

[19] K. Pfeffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research,” *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2007.

[20] A. Cleven and K. M. Hüner, “Design Alternatives for the Evaluation of Design Science Research Artifacts,” 2009.

[21] A. Adewumi, S. Misra, and N. Omoregbe, “A Review of Models for Evaluating Quality in Open Source Software,” *IERI Procedia*, vol. 4, pp. 88–92, 2013.

[22] ISO, “ISO/IEC 25010,” 2011. [Online]. Available: http://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Accessed: 22-Jun-2017].

[23] SpikeSource, Center for Open Source Investigation, and Intel Corporation have, “Business Readiness Rating for Open Source: A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software,” pp. 1–22, 2005.

[24] Navicasoft, “Open Source Maturity Model: Moving OSS into the Mainstream,” 2006.

[25] E. Kalliamvakou, G. Gousios, and K. Blincoe, “The promises and perils of mining GitHub,” *Proc. 11th*, pp. 92–101, 2014.

[26] K. Aggarwal, A. Hindle, and E. Stroulia, “Co-evolution of project documentation and popularity within github,” *Proc. 11th Work. Conf. Min. Softw. Repos. - MSR 2014*, pp. 360–363, 2014.

[27] GitHub, “GitHub API v3 | GitHub Developer Guide.” [Online]. Available: https://developer.github.com/v3/. [Accessed: 10-Feb-2018].

[28] E. Shihab, C. Bird, and T. Zimmermann, “The effect of branching strategies on software quality,” *Proc. ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. - ESEM ’12*, p. 301, 2012.

[29] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, “GitHub Projects. Quality Analysis of Open-Source Software,” no. c, pp. 80–94, 2014.

[30] A. Alali, H. Kagdi, and J. I. Maletic, “What’s a typical commit? A characterization of open source software repositories,” *IEEE Int. Conf. Progr. Compr.*, no. June 2014, pp. 182–191, 2008.

[31] H. Borges, M. T. Valente, A. Hora, and J. Coelho, “On the Popularity of GitHub Applications: A Preliminary Note,” pp. 1–11, 2015.

[32] J. Sheoran, K. Blincoe, E. Kalliamvakou, D. Damian, and J. Ell, “Understanding ‘ Watchers ’ on GitHub.”

[33] M. Morandini, A. Siena, and A. Susi, “Risk Awareness in Open Source Component Selection,” pp. 241–252, 2014.

[34] N. Zazworka, M. a. Shaw, F. Shull, and C. Seaman, “Investigating the Impact of Design Debt on Software Quality,” *Work. Manag. Tech. Debt*, pp. 17–23, 2011.

[35] E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” *J. Syst. Softw.*, vol. 86, no. 6, pp. 1498–1516, 2013.

[36] L. Rosen, *Open Source Licensing: Software Freedom and Intellectual Property Law*. 2004.

[37] J. Lerner and J. Tirole, “The scope of open source licensing,” *Journal of Law, Economics, and Organization*, vol. 21, no. 1. pp. 20–56, 2005.

[38] C. Cowan, “Software security for open-source systems,” *IEEE Security and Privacy*, vol. 1, no. 1. pp. 38–45, 2003.

[39] D. Godin, “Kernel.org Linux repository rooted in hack attack • The Register,” 2011. [Online]. Available: https://www.theregister.co.uk/2011/08/31/linux_kernel_security_breach/. [Accessed: 10-Oct-2017].

[40] H. Choi and H. Varian, “Predicting the Present with Google Trends,” *Econ. Rec.*, vol. 88, no. SUPPL.1, pp. 2–9, 2012.

[41] L. Michelle Grantham, “The validity of the product life cycle in the high-tech industry,” *Mark. Intell. Plan.*, vol. 15, no. 1, pp. 4–10, Feb. 1997.

[42] R. Wieringa, “Design Science Methodology: Principles and Practice,” in *ICSE ’10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010, vol. 2, pp. 493–494.

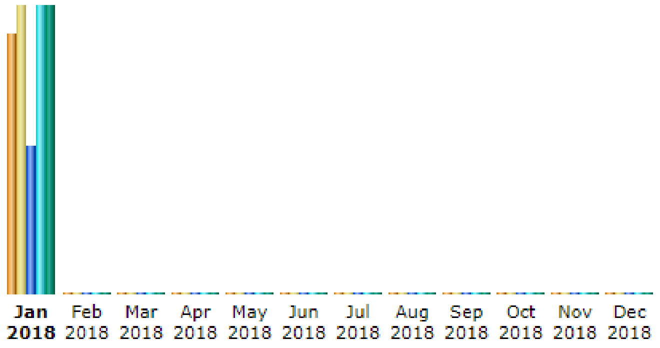
APPENDICES

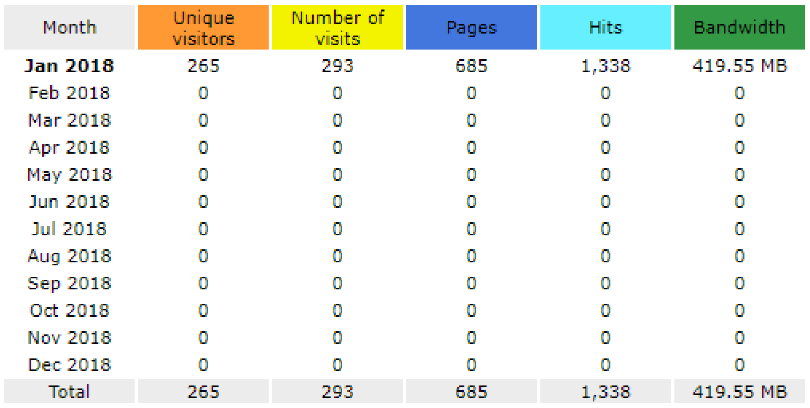
Appendix A: Feedback Form

< IP >	3	no		neutral		Customise it to work with languages other than JavaScript (for a Java library, this tool complained about a lack of a packages.json file, which does not matter in Java)	jmockit1
< IP >	3	yes		neutral		Currently only metadata of the code is assessed. SO it gives information about the project's viability status. A few suggestions: - Show trends in terms of increase/decrease in project activity - Take the number of contributors into account. Because if more people contribute to a project, it would be a project with a better health/future - Perform code analysis on the repository. If you're able to determine the programming language, a suitable static code analysis tool can be ran against the codebase, which might provide insights about the actual code quality	graphql
< IP >	3.5	no		positive		* Would be nice to be able to run the tool an all dependencies for a project at once and that dependencies with possible quality issues get flagged * Component maturity tab gives nice overview of relevant data, especially the security issues. * Commits tab dat is the least useful for me in the current form. Maybe an added analysis on comitters to see if there is a healthy ecosystem of developers would be useful.	delayed_job
< IP >	3	no		neutral			GitElephant
< IP >	3	no	no	neutral		The commit message i dont find it that intresting, but the security issues and the support activity are nice	laravel
< IP >	3	no		positive		The use of red and green colors are very useful to see quickly if something was evaluated positively or negatively. The evaluated project has 4 security issues, what does this mean? Its red, so I assume its bad, but how bad is it really when compared to similar projects? What do the colors on the issues page mean? Is green solved and red open issues? Its not really clear, now I just have to take a guess.	spark

< IP >	3	yes		neutral		Results seem not entirely correct, as this repository does use CI, has external documentation, has API documentation and has an installation guide. In general I see some data is collected, but any judgement is left to the user. If, and only if, the data quality is good an immediately visible score would be good to have. At this moment I do not see the benefit of this tool as some data is simply not correct (and probably hard to get fully correct for any case) and judgement is left to the user. To do this myself costs not that much time (at most a couple of minutes) and checking the data from this tool will require that anyways.	django-auditlog
< IP >	3	yes		neutral		I don't know which level would you like to reach with this app, but it would be truly useful to add a comparison tool to it, because when I'm searching for a new component I will find a few to consider and it would be quite effective if I could put them into a tool and it would show me a comparison table and even it could tell me which one is the best. With the current solution I will need multiple browser tabs and etc. which makes it not so comfortable.	laravel
< IP >	4	yes		neutral		Nice idea. A few things that could be improved: format numbers - one page said something like â€œ325.111111111 daysâ€• shorten lists - the Wordpress page lists 110 CVEs, would be better to just show the most recent few and maybe add pagination for the rest shorten issues - they show the entire issue text which can be super long explain the colours - I have no idea what the red and green mean in the issue list.	WordPress
< IP >		No		Positive		I think it is very helpful to see which parts of the component are great (green) and which are less or do not exist (red). This makes it easy to find out more about the pros and cons of using the component. Furthermore, the information that is being used in all sections is very helpful and interesting to see as well.	laravel

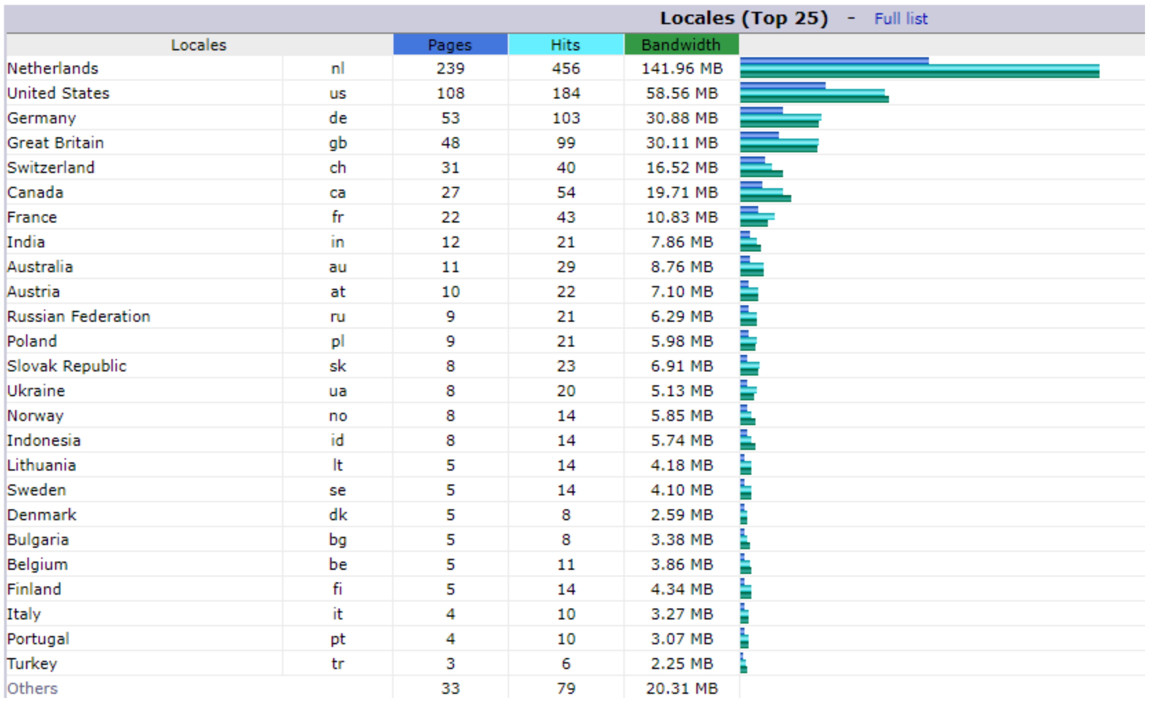
Appendix B: Users





Month	Unique visitors	Number of visits	Pages	Hits	Bandwidth
Jan 2018	265	293	685	1,338	419.55 MB
Feb 2018	0	0	0	0	0
Mar 2018	0	0	0	0	0
Apr 2018	0	0	0	0	0
May 2018	0	0	0	0	0
Jun 2018	0	0	0	0	0
Jul 2018	0	0	0	0	0
Aug 2018	0	0	0	0	0
Sep 2018	0	0	0	0	0
Oct 2018	0	0	0	0	0
Nov 2018	0	0	0	0	0
Dec 2018	0	0	0	0	0
Total	265	293	685	1,338	419.55 MB

Figure 9 Traffic log



Locales (Top 25) - Full list

Locales		Pages	Hits	Bandwidth
Netherlands	nl	239	456	141.96 MB
United States	us	108	184	58.56 MB
Germany	de	53	103	30.88 MB
Great Britain	gb	48	99	30.11 MB
Switzerland	ch	31	40	16.52 MB
Canada	ca	27	54	19.71 MB
France	fr	22	43	10.83 MB
India	in	12	21	7.86 MB
Australia	au	11	29	8.76 MB
Austria	at	10	22	7.10 MB
Russian Federation	ru	9	21	6.29 MB
Poland	pl	9	21	5.98 MB
Slovak Republic	sk	8	23	6.91 MB
Ukraine	ua	8	20	5.13 MB
Norway	no	8	14	5.85 MB
Indonesia	id	8	14	5.74 MB
Lithuania	lt	5	14	4.18 MB
Sweden	se	5	14	4.10 MB
Denmark	dk	5	8	2.59 MB
Bulgaria	bg	5	8	3.38 MB
Belgium	be	5	11	3.86 MB
Finland	fi	5	14	4.34 MB
Italy	it	4	10	3.27 MB
Portugal	pt	4	10	3.07 MB
Turkey	tr	3	6	2.25 MB
Others		33	79	20.31 MB

Figure 10 Geographic distribution