TECHNICAL UNIVERSITY OF DENMARK

Quality and IT Security assessment of Open Source Software projects

by

Michael B Nielsen

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Security in Distributed Systems
Department of Applied Mathematics and Computer Science

January 2017

Declaration of Authorship

I, Michael B Nielsen, declare that this thesis titled, 'Quality and IT Security assessment of Open Source Software projects' and the work presented in it, is my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Michael B Nielsen

Date: 22nd January, 2017

TECHNICAL UNIVERSITY OF DENMARK

Abstract

Security in Distributed Systems
Department of Applied Mathematics and Computer Science

Master of Science

by Michael B Nielsen

Trustworthiness in open source software can be evaluated on attributes of software engineering. The attributes possibilities are to describe trustworthiness is vast, but security have to be evaluated as it has always been a consideration in trustworthiness. The attributes security, maintainability and team capabilities can evaluate trustworthiness as a metric.

The trustworthiness is evaluated using a software product with information on open source software. The software product is an assistance for people to understand the trustworthiness of a software product. The security metric is evaluated based on vulnerabilities in the CVE register and open source software projects' data from OpenHub. Maintainability using source code data to determine understandability and maintainability, and team capabilities are described using contributor information on their projects contributions. The trustworthiness can thus be evaluated for any open source software project with information contained in the sources.

Trustworthiness metric can be further expanded by new metrics, which are easily added to the software product.

Acknowledgements

I would like to acknowledge Christian Damsgaard Jensen (Thesis Advisor) for his help and guidance through the master thesis without his help I am sure the project would not have been of the same quality and I appreciate his help greatly.

Furthermore I would like to thank my motivational coaches Signe Schønning and Freja Maas for helping with staying focused on the thesis. Thanks for help with the editing of the thesis to Ole Bøndergaard and Freja.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Trustworthiness is important for a Software Product to be successful, since most users are easily persuaded into using another product if a product is not trustworthy. The same applies for Open Source Software Products, which are only as popular as the users satisfaction with the software. Open Source Software do not normally use advertisement, and thus the reputation and recommendation of other users are the key essential in getting users to use and eventually trust the software.

Trustworthiness is not a uniform value, which can easily be measured in software. Software Trustworthiness is a combination of metrics, which combined can give an estimation of how trustworthy a software product is. The metrics are not in any way standard and academics are trying to find out how trustworthiness is best described from Software Engineering attributes. The project concentrates on evaluating an Open Source Software Project in terms of Trustworthiness, since Open Source Software Projects information is easier accessible compared to commercial software products. Security have already been considered the main attribute for trustworthiness, but depending on the software other parts play an essential part in trustworthiness as well. The chosen attributes for this project is the developers contributing to software, the product's maintainability and naturally the security metric including the project's dependencies as well. The chosen metrics are just a few of available metrics for evaluating trustworthiness.

The product will be a software product evaluating the trustworthiness of software by combining information from different sources based on the metrics previously mentioned. The metrics will be calculated and evaluated and with all the information on the security, team and maintainability of the project. The metrics will be combined into a score for the overall trustworthiness of the score. The trustworthiness score can be used by developers checking a library or users ensuring that a software product is trustworthy before the software is used. A standardised trustworthiness score would be a great asset

for developers, but this thesis will try to create a measurable evaluation of Open Source Software Projects.

Chapter 2

State of the art

The analysis encompass information on aspects important to trustworthiness in Open source software, which includes information on the Open Source concept and aspects of trustworthy software. Open source software has gained momentum with the start of the Internet, which made information sharing easier with more people. A large amount of the software is available through Open Source and almost anybody using a computer is using a piece of software, which is Open Source.

2.1 Open source software

Open Source Software Projects[4] vary greatly from project to project, and the way projects are organised and owned by organisations. Open Source software can be organised in many ways depending on the organisation behind the project. Open Source projects are mostly distributed through an Open Source license for the well established projects, while smaller projects are often distributed through Github or similar services and just available to anybody.

The idea of Open Source has flourished with the Internet, which made the distribution of software many times simpler than previously. Open Source creates the possibility for anybody to contribute to a project, which is part in their interest or a focus area of theirs. People can contribute more or less depending on the time available for the project, since the Open Source development is all volunteer work and people use Open Source as a hobby. Individuals contributing a greatly to projects will in communities have more power in the community, but this depends solely on the organisational structure.

Many large software projects are in fact Open Source and created by a a few individuals grown into a large community. The most popular browsers are Open Source projects

such as Mozilla's Firefox and Google's Chromium project. For developers several tools and products such as Oracle's MySQL and Git are open source for anybody to use and possible to join. Oracle is mostly known for their database, but also owns Java and the Open Source project MySQL, which is an industry database standard in many products. Apache is another Open Source projects owner, which is most know for Apache Server. Apache has a large open source community with many other Open Source Projects like Solr and Hadoop. Solr is used for indexing and searching documents for their content, and Hadoop is a tool to process very large data sets by using a method called MapReduce. Oracle like many other Open Source Communities has many different projects.

2.1.1 Open Source Definition

The Open Source Definition (OSD)[5] is derived from Debian Free Software Guideline in order to create a license enhancing the open source principles. The licenses of different open source software products have to be accepted to become an Open Source license, which can either be for a specific product or a distribution of software products. The Open Source Licenses are authorized by the Open Source Initiative (OSI), which is a Californian public benefit corporation. The distributions of software or individual software are distributed under a license by organisations such as Apache, Apple[1], and Mozilla. The requirements of the definition can be found below, where all licenses have to abide by all the requirements.

1. Free distribution of software
2. Available for free and well written Source Code
3. Derived works from license of original software
4. Integrity of The Author's Source Code. The license can restrict distribution of the software to only modified or derived work of the software, and the derived work is required to be of a different name or a new version number.
5. License must not discriminate persons or groups.
6. License must not discriminate against field of endeavours.
7. License must apply to all the programs redistributed.
8. License must not be specific to a product or distribution, and as long as the license is upheld the software can be redistributed with the same rights as it was distributed within the original software distribution.

[1]Well known brand, but is not known for Open Source. Apple though has an Open Source License Authorized by OSI. Project examples are WebKit, CareKit and programming language Swift

9. License must not restrict other software, which the licensed software is distributed with.

10. License must be technology neutral in order to ensure availability software re-use.

The software should be available for anybody to read, modify and study the product's source code. The project should thus be available to any individual interested in the project, whether the interest is in using the product, being part of the project, or working with the product to create a new product. An example could be the TOR browser, which creates a new browser on the basics of Mozilla Firefox to develop new functionality to the browser software.

The community can limit the participation of individuals, and accepted participants will have all their contributions examined for quality assurance. The projects are required to develop well structured and well written source code. The OSD ensures the availability of Open Source Software, and the availability is the main requirement of all for Open Source Licensing.

2.1.2 Open Source Software stakeholders

Open Source projects all have certain stakeholder types, which are *Contributors, Users and Vendors*. The different stakeholder types are the typical roles, which are interested or invested individuals in the project. Any of the stakeholders can be the owner depending on the organisation type, which is elaborated in section 2.1.3.

The owner of the project is an individual or group in charge of the development of the project, distribution of the product and owns the copyright to the software. The different projects' software products are often owned by a group, which are interested in developing software for their own usage, usage by the masses or creating a profit by selling to paying users. The **Vendor** is a company payed to develop the Open Source Software for the project and a company paying full time employees for developing the software. Vendors are interested in creating a profit for the company by getting payed for their services, or in some cases owning the software for selling additional services with the product. The **Contributor** is an individual, who spends time developing the software in his spare time without focusing on monetary profit. The contributor often contributes to a project because of varying reasons of motivation. The most common reasons of motivation are interest in the product, the goodwill purpose of the project, or for software development experience and improvement in their skills. The **User** is the individuals and/or groups using the software, which can be on a computer or other devices, or in a product sold to other users.

The stakeholders in an Open Source Software Project can be organised in any fashion, but the most common Open Source Projects are organised in a management scheme presented in section 2.1.3

2.1.3 Open source organisation types

Open Source Projects are organised in a set of organisation types, but most projects are organised in one of 4 types[6]. The biggest difference is the ownership of the source code and management of the project.

1. Single Vendor Open Source Projects
2. Development Communities
3. User Communities
4. Open Source Competence Centers

The **Single Vendor Open Source Projects** are not the most commonly known kind of Open source projects. The single vendor projects are as the name states a single company or organisation in charge of the entire project. These projects have different contributor types for example professional developer in their organisation, external developers and open source contributors. The contributors for a single vendor project are required to sign a contract, where the source code developed by contributors becomes property of the organisation in charge of the project. Finding contributors outside of the organisation can be a great challenge as for most Open Source Projects the source code remains the property of the contributors and the community. The contributors normally give up the rights to their code in order to be part of a large project, where they can contribute and gain great experience in software development.

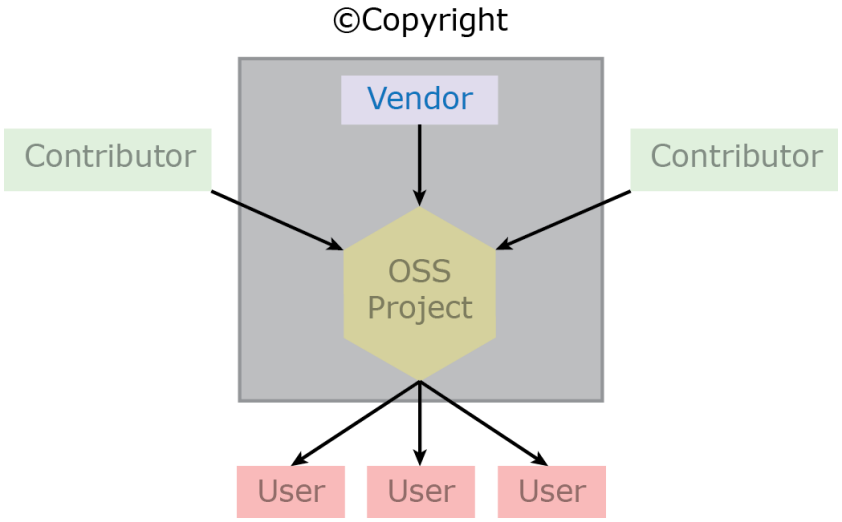


FIGURE 2.1: Single Vendor Open Source Projects are owned by a single organisation or company with contributors of their own and external contributors. The users are using the distribution of the software.

The Single Vendor Open Source projects are licensed under an Open Source licence, which means the Open Source requirements are fulfilled. The source code is available to anybody, but the distribution of the software is still the responsibility of the organisation. An example of a Single Vendor Open Source Project is MySQL, which was owned by the Swedish firm MySQL AB but is now owned by Oracle. MySQL like other Single Vendor Open Source Projects are prone to be forked by teams of developers. Forking a project means to create a new derived project or organiser split into several projects. MySQL have been forked several times to MariaDB for example, and the developers are allowed to create the project as Open Source with a different kind of organisation not owned by Oracle. MySQL has been forked into several projects, but MySQL has still remained a leading contender for database solutions. The project being forked is a great risk for this type of Open Source organisations, where all the source code is reused by a new organisation. The organisation type is close to the commercial software products and does usually have a commercial extension to the Open Source product.

Developer Communities are the well known organisation type of an Open Source project. The organisation of a development community has a large number of contributors, where the contributors are the owners of the project. The contributors and the community are thus controlling the distribution of the software from the project and owning the source code. This kind of Open Source project are often licensed under either GNU Open Source License or another collective of Open Source licenses.

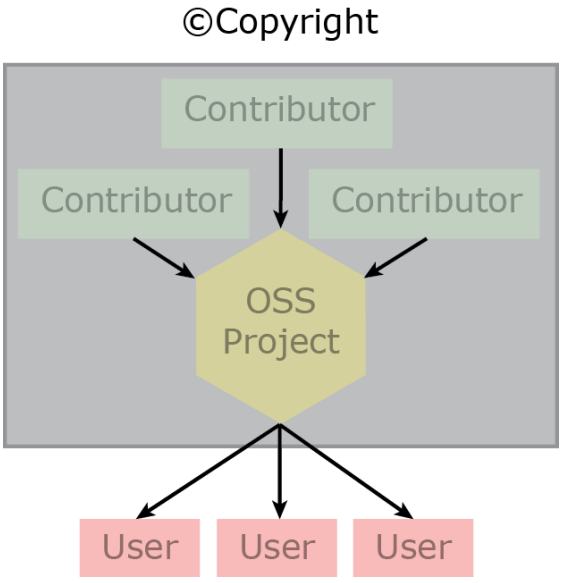


FIGURE 2.2: Developer Communities are owned by the contributors in a hierarchal organisation with users using the software.

The internal organisation will have leaders to make executive decisions for the project. The project leaders can either be chosen based on their contribution to the project, or the decisions are made as a community. In the community the contributor's level of contribution decides how much decision power the individual contributor has. The Open Source project will have a list of guidelines for contribution on how the code and other contributions should be formatted to be accepted. The contributions will be checked for correct format and if necessary the design before being accepted into the project. An example of this organisation type is Linux or Mozilla Firefox. Mozilla has different levels in their organisation based on contribution, and the level and ball park figure of contributors can be seen in figure 2.3.

FIGURE 2.3: Mozilla is a large Open Source project showing the hierarchy for the contributors in the project. The ball park figure tells the numbers for the different hierarchal levels.

Developer Communities generally have a Project Core with extremely active contributors to the project, which are in charge of the overall project from accepting contributions, distributing tasks and decisions on the product or project. The Core are experienced developers, system designers and have been part of the project for a long time, who will have to provide feedback on solutions and accept the solutions with acceptable quality. The decisions have to originate from part of the organisation, and the Core Contributors will have more decisional powers compared to the less contributing contributors.

The **User Communities** are similar to the development communities, but the projects are owned by the users of the software compared to the developers. The users or user communities of the software, develop the software in house or pay to have the system developed for them. The software can be developed under an Open Source License or be released from the development into an Open Source License as the project is being finished and distributed. The user communities are sharing the ownership of the software and the distribution. The user communities can be an industry sharing the expenses to develop and maintain a system with the specific requirements for similar user group. An example could be universities collaborating in developing an intranet for information between the students and teachers about courses and university groups.

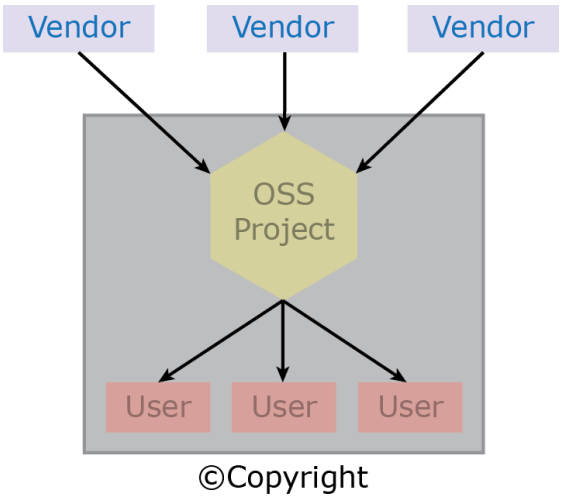


FIGURE 2.4: User communities are organisations owning the project and having specific customers or users as intended users. The open source project is developed in collaboration with the owners either in house development or purchasing the software product from a vendor.

The **Open Source Competence Centers** is as the name states a competence center for Open Source projects. The competence center shares resources, advise and information of how to create a successful Open Source project. Activities are organised by the competence center like conferences and workshops. The competence center role includes creating the facilities for an Open Source project to thrive, which can be anything from assistance or utilities for the contributors and users of the project. The competence centers will include various organisations like small projects, Non-Governmental Organisations or private companies as users. The projects can be all kinds of projects, or the Competence center might have a specialty for a certain type of projects. The competence centers do exist all over the world, but the Open Source Competence Centers are normally geographically restricted. The restriction are caused by the attempt to emphasise the environment for Open Source projects within the region.

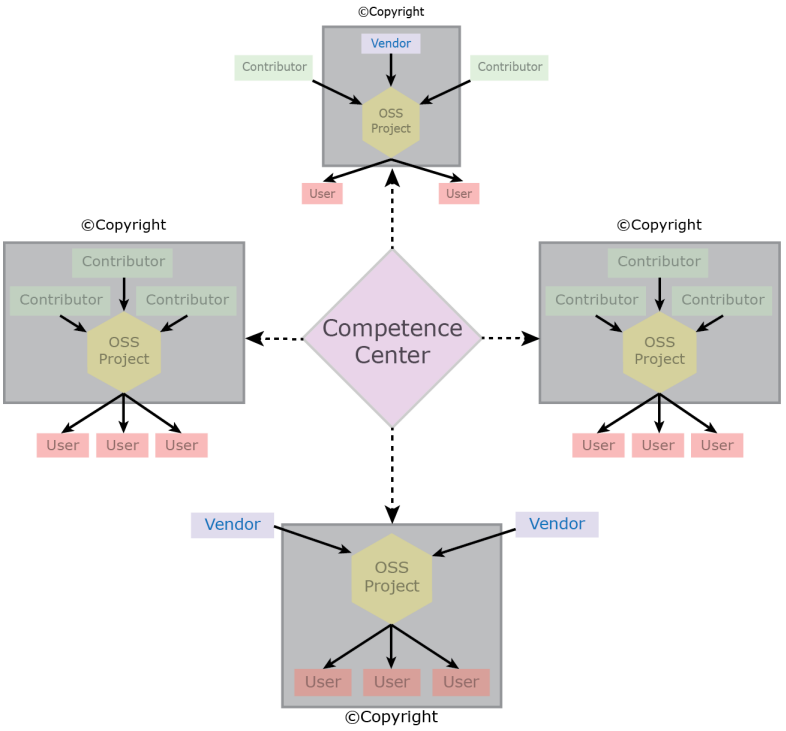


FIGURE 2.5: The Competence Center is in an advisor role for Open Source Software Projects, which can get assistance in different aspects of creating a successful project.

The organisation types are oriented toward a specific Stakeholder being the owner and another (or the same) being the developing part of the software product. The Compentence Center is usually a governmental institution creating an incubating environment for the projects. The Competence Centers are usually geographic limited to a nation or region, and the Compentence Center helps and guides the projects to be successful. Development Communities are contributor oriented with the contributors being both the software Owner and Developers of the products. The User Communities are Vendor Oriented as they are the deveelopers of the software product altough a User group is the Owner. The Single Vendor Open Source Project is Vendor Oriented with being both Owner and main Developer of the software, although Contributors can develop parts but these products are challenged on finding contributors.

2.2 Software reuse

The reuse of software happens greatly in software development, where developers exempted from developing system functionalities from scratch. Open Source Software projects can reuse software internally in the project or from other Open Source Software projects to gain functionality. Software development have a vast variety of software development tools made available to developers for different development environment.

Software solution and Open Source Software tools are available to ease the re-use of software such as Git.

The reuse of software can in both low and high level programming languages utilise different methods to include source code from other developers. The most common reuse is the libraries included in the development environment chosen for the development of the software, which includes general functionality by the owner and from 3rd party developers which libraries have been made available for everyone. The libraries included in the software languages are libraries the organisation in charge of development have decided to include and thus used correctly are safe to use. High level programming languages have basic functionality included, but get access to more tools and often more advanced tools. 3rd party software tools can be made available through package managers or from version control solutions. A simple example of this can be the Python programming languages, which have many different tools, and through PIP (Pip Install Packages) many more tools like Numpy and PyMongo. These libraries enables developers to use advanced data processing with Numpy and to connect Python scripts with MongoDB databases.

Krueger[7] explains the view of Software Reuse as of 1992, which has a few different methods of reusing software. Software reuse have changed since 1992, but he explains a few methods for high level programming languages. The high level programming languages have shifted from back in 1992 to today, where C and definitely C++ was seen as high level programming languages. Today C++ and C are viewed as in-between high and low level programming languages, because the developer have to take care of a few more issues than other high level programming languages as of today. A few of these issues are memory allocation and garbage control, where C and C++ require this from the developer. The possibility of closer hardware interaction and speed is the advantage of C and C++ compared to the high level programming languages. Today's high level programming languages, such as Python, Ruby, Java and etc., does handle these issues and more for the developer to focus on the project.

Krueger explains the methods of Scavenging, source code components, schemas and application generators. **Scavenging code** is to use duplicating code into a new project, which can give the project new functionality in an easy way with little modification of the original code. This is a simple way of reusing code, but require the code to be available either through an open source project or from available source code in an organisation's previous projects. The idea is to add the source code with the desired functionality to the project. This is a simple way, but should be avoided if more modern approaches are available for software reuse.

Source Code Components are components developed usually with an object oriented language approach, where components can be reused from other developers and have a

large set of functionalities from available library components. The components can be generic data types as most computer scientists know such as String, Stacks, Queue, List, Maps and etc. These components can then be reused in any system, where information needs to be stored for data processing. The inheritance and subclass structures for components are an advantage of re-using components, which can increase the abstraction level for the developers of the system. Components are not only simple structures for data handling, but can be larger components with several classes and object reused internally in a system or in an external system. These 3rd party components can have all kind of functionality, but can add specialized functionality for an area of expertise, like mathematics, data mining, or another area. Using these data types or components lets the developer work on a higher level of abstraction and not having to deal with developing and testing the component to operate as intended.

A few less used software reused methods are Schemas and Application Generators, which are used for a specific task. **Schemas** are able to create a conceptual connection between Services and Models, which are often used in SOA. Schemas such as XML uses an XML Schema Definition (XSD) for the service to understand and verify the XML structure. Schemas can similarly be used to handle data objects as well with a specific structure as a replacement for databases, but was normally done prior to this millennium as database technologies have matured since. **Application Generators** are used to generate applications for a specific purpose. The application can be generated from a definition of the task to be solved. The definition can be made in different ways, where SOA server functionality can be generated based on XML files. Other possibilities are to generate the application based on interaction with a simplified user interface from pictures and drag and drop functionality, which is available in certain industries.

Today Object-Oriented software is greatly used in many software projects, but a few other kinds are available. Krueger brings a few older examples up, which are outdated or less frequently used today. The components describe the basics of how Object Oriented Programming works with vast amount of libraries available with many resources and functionalities for developer. A few package management systems make lesser tested and known libraries available for the developer to choose from depending on the functionality needed in his system.

2.3 Software Trustworthiness

In the history of computer science trustworthiness have been defined differently over time, as the academic society changed from the original view of software trustworthiness, only to include security[1]. Security is an important part of trustworthiness, as

insecure software will not be trusted by the users. The security is of course a simple version of looking at trustworthiness from a computer scientist's point of view, where general users will have other aspects to consider as they use applications and services. The definition of trustworthiness was extended to include the quality of software. The quality of software have many aspects to measure just like security, for example complexity, reliability, availability, or life cycle cost.

The Trusted Software Methodology (TSM) found 44 trust principles for software to rate the quality of the software project. The principles both have aspects from security and software engineering. In the figure 2.6 the TSM considers the relationship of software trustworthiness and the created trust principles.

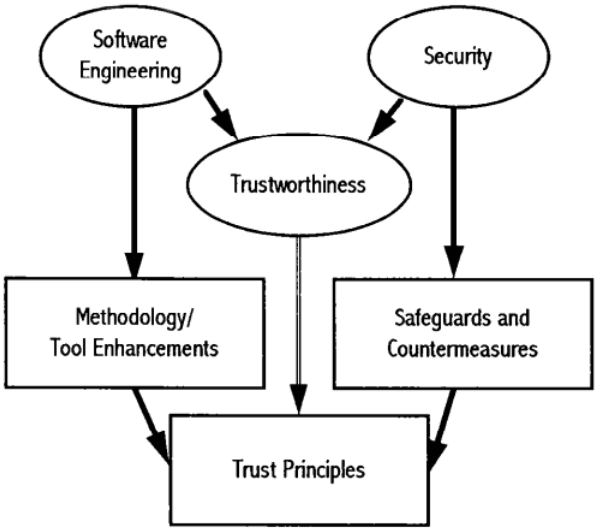


FIGURE 2.6: Graphic display of how [1] is considering trustworthiness and their developed 44 trust principles. Software trustworthiness is a combination of security and software engineering matrices. The authors have then used different methods to find the resulting trust principles.

The trust principles are in other words created from software engineering methodology, security safeguards and countermeasures, and trustworthiness principles. These trust principles are prone to the complete life cycle of the software development. The principles are used in TSM to make a rating from T0 to T5 depending on the number fulfilled trust principles.

The TSM does state that the trustworthiness of software often depends on the investment, which the stakeholders are willingly to invest in the project. To create a completely trustworthy software product requires tightly controlled quality assurance for the software never to fail or crash. To develop completely trustworthy software is similar to developing complete secure software, which will be extremely expensive compared to

the level needed for the project to be sufficiently secure or trustworthy based on the requirements. Only a few organisations are willing to invest in a completely trustworthy or secure software, which are often organisations in the military or organisations with high risk purpose like space travel. Space travel invest a large amount of money in the software being reliable, since the space shuttles used are very expensive, but the space industry have seen a few failures caused by small issues. The security aspect is a design and development consideration for a team to improve, but most companies design sufficiently secure software for the purpose of the system. All companies might not have the same requirements to the software as military organizations have, and thus these military or high security organizations make a larger investment in security and trustworthiness.

The trust principles can be found in Appendix A.

Trustworthiness is discussed as a concept in comparison to Socio-Technical System (STS), which describes any humans who uses the system as means for communication[8]. The trustworthiness is with a focus on the STS but instead of having 44 principles, the article comes up with attributes in different aspects of the development. The STS is a limited kind of system, but these attributes does mostly apply to any other kind of system, the attributes are known software engineering attributes. The attributes found from the 44 trust principles and attributes are similar. The STS have a focus on the users perception of the system, as a concept for software engineering.

STSs' can be anything used for communication between people on any kind of platform, which means the system can use any kind of media as well. An STS can be a service, system, applications, or mobile apps, thus the STS definition is very versatile to use to describe systems. The STS definition can then be used to describe a large set of Open Source projects as well, but Open Source project will not all be part of the STS definition. The attributes can be used for systems that are not an STS, since the attributes for most part are terms found in Software Engineering. The user perception of the system is important for many systems, because if the system has a feeling of untrustworthy, a reputation for sharing private information, leaks, or a bad service. The users or developers using the system will have to feel secure and be able to trust the system, otherwise the users will find an alternative with similar functionality.

The public is becoming more aware of the security of the systems they use, and many systems have security problems like the Heartbleed scandal. Also, user information is being leaked in millions every year caused by security vulnerabilities. People are often concerned with the quality of software in regards to privacy and availability. People are being more aware than previously on the information, which they share with other people and especially what people. Availability or lack thereof is easily noticed by the users,

when the systems are down or unavailable. The attributes do have these as a concern along with other attributes, which can be found in figure 2.7.

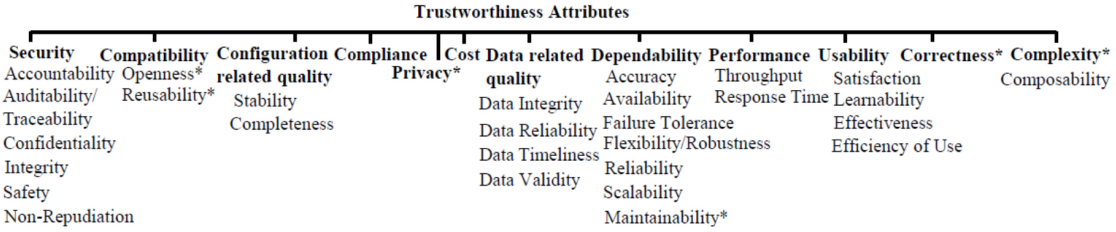


FIGURE 2.7: The figure shows the attributes, which have been deemed of importance in terms of trustworthiness. The attributes can be used to describe how trustworthy a system is for a user. The attributes are sorted into a categories, which the term describes a share of.

The figure 2.7 shows a large set of Software Engineering terms, which describes a section of the overall trustworthiness of a system. The attributes each explains a part of the software quality, which all contributes to an indication of the system's trustworthiness. Security is a category with a large set of attributes to define the security, where the CIA attributes are represented as integrity and confidentiality. Availability is a dependability measurement as is reliability with several others. The names with the asterisks have been examined further in the article[8].

Surato[9] describes a few ways to evaluate the trustworthiness of a software project. A way is an Eclipse plugin, which can evaluate and test the trustworthiness of the system based on the architecture. The plugin is created by Immonen & Palviainen (2007)[10], which can test an Open Source Component's trustworthiness based on the component's reliability.

The product developed by Immonen and Palviainen is called Reliability Analysis Tool or Reliability and Availability Predictor (RAP), which analyses how reliable and likely the component is to fail. The article describes their focus on reliability of a component as part of a trustworthiness evaluation. The authors describe critical requirements to include security, reliability, performance and functional requirements. The developer implementing the components in a system will have reliability requirements, since a failure in the component will likewise result in lack of the functionality or of the entire service depending on the criticality. The RAP tool is created as a plugin for Eclipse IDE and utilises a model based analysis on the system.

First the tool uses the model of the system architecture for the user to create requirements for the components. As the requirements are finished the components are then tested for reliability of failure from a generated Markov Model. The final step of the analysis is to test the reliability of the single component and the components integration in the overall

system. The components are tested with unit tests to check the individual requirements to the components. The product is mostly usable for developers of a closed project or an Open Source project with access to the source code and a system design. The requirements of the system is to have internal knowledge of the design to use the RAP tool to test both the design and its requirements. This information is essential and the design is not available for most systems unless you are part of the developing process.

Another option for reusing existing software can be Commercial Off The Shelf (COTS) solutions in order for the system to easily gain new features. The COTS is a component that can be added to a product for the purpose of adding a service or functionality. The difficulty with COTS is finding the best fit for the project, where no overview of solutions and products are available. The trustworthiness solutions are not easily found from solutions with problems.

A large number of articles covers the topic of trustworthiness, although no current industry standard in trustworthiness seems to be close. Trustworthiness is described using attributes otherwise used in Computer Science, such attributes can be seen in Figure 2.7. Trustworthiness depends greatly on the type of product and its intended use, where trustworthiness can be difficult to incorporate for all kinds of software.

2.4 Risk assessment

Risk assessment is a large discipline for many areas in industries and product development. Risk assessment in Information Security is an important issue for many organisations, and the International Organisation for Standardisation (ISO) and National Institute of Standards and Technology (NIST) have created a standardised process each to assess the threats for an IT system. The ISO 27005[2] and NIST Special Publications 800-30[11] are both process to assess the risk on IT Security for a system. These 2 processes are similar, and although the vocabulary covers the same concepts, both organisations have created their own definitions of the concepts. The NIST concepts are more clear, simple, and hands-on, which is why it will be the one used to describe these aspects.

The process in figure 2.8 is from the ISO 27005, and shows how the process advances. The process contains a 7 stage context establishment, risk analysis, risk assessment, risk treatment, risk acceptance, risk communication and risk monitoring.

The process starts with the Context Establishment and iterates through the process, and the process is a continuous process. The process is continuous and should be active for the life time of the product. The process is thus not only needed in the beginning

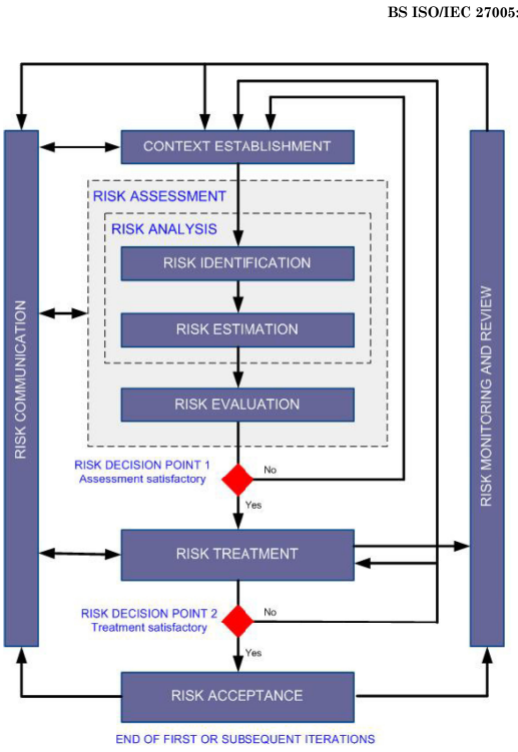


FIGURE 2.8: From ISO27005 [2], the risk assessment process is a repeated process as the environment or presumptions change for the IT system. The process starts in the top and is repeated throughout the life cycle of the system.

of the project, but needs to be kept updated as the product and the context of the product changes. The context establishment is about establishing the context of the system, which includes the scope, assumptions and restrictions of the environment both computational and in the organisation. Restrictions can be made in case the organisation are obligated to report information to the public, and this should be possible for the information from the system. The risk analysis is part of the risk assessment and is the phase, where all possible threat scenarios are found for the systems in the organisation. The risk analysis is about establishing a scenario that could be a risk for the system and organisation. The Threat Scenario found consists of a Threat Source (adversary), Threat Event, and Vulnerability. The Risk Analysis will be an iterative process with all possible or all relevant Threat Scenarios being listed and risk assessed for the organisation. The **Threat Source** is the adversary of the threat, which can be an individual, a group or an organisation, who wishes to do harm to the organisation or system. Threat Source will have different reasons of motivations and means for initialising an attack, where the NSA would have more resources and thus higher likelihood of success. The **Threat Event** is the specific attack the Threat Source is carrying out. The **Vulnerability** is the component or part of the system used by the Threat Source to initiate the Threat Event. The second part of the Risk Analysis is the Risk Evaluation, which is about estimating the risk of a scenario based on Likelihood of occurrence and success. The

estimation is usually evaluated on a scale from 1 - 4 for both Likelihood and the Risk with the combination of higher scores being more critical.

Likelihood \ Impact	Low	Medium	High	Critical
Improbable	1	2	3	4
Unlikely	2	4	6	8
Likely	3	6	9	12
Frequent	4	8	12	16

TABLE 2.1: The combination of Likelihood and Impact scores are multiplied to give an impression of the overall risk of a Threat Scenario. The values are grouped to give an impression of the severity of the risk on the system and organisation.

The Risk Treatment is how to handle and mitigate the Risks identified for the system to a satisfactory level. The Treatment depends on the Threat Event and Vulnerability and an example can be to implement a new level of authentication in the system, if the problem is with the confidentiality. The Risk Acceptance is a level, where based on the Evaluation and Risk Treatment the Risk is deemed acceptable for the system and the organisation. The Risk Treatment can be redone in case the Risk level is still too significant. The level of the risk is usually mitigated to an acceptable level or the best compromise in terms of cost, since removing the risk completely can be a significant expense for the organisation.

The vulnerabilities are the interesting aspect, since the software system should make a risk assessment of what vulnerabilities are acceptable for the necessary level of protection. The vulnerabilities should be minimised for the software system and should for very confidential systems be able to protect even against an attack of any adversary. The threat source is not of significant importance as any adversary can be motivated to attack any system and can be difficult to rate the system on their level of confidentiality needed. The vulnerabilities are a factor along with the threat events, which the system should be able to resist.

2.5 Vulnerabilities

In a risk assessment process the vulnerabilities are an important factor for assessing the software quality. Vulnerabilities is a weakness in the software for an adversary to exploit in order to harm, alter or steal information from the system. Vulnerabilities are entry points to a software system, and a vulnerability in a software dependency will in

most cases create the same vulnerability for the implementing system. The most famous vulnerability as of recently is Heartbleed, which will be further discussed in section 2.5.3.

2.5.1 CVE

Common Vulnerability and Exploits (CVE)[12] is a dictionary for finding known vulnerabilities in software systems. CVE register was created by Mitre in 1999 and have since become the industry standard for vulnerabilities, where previously many vulnerability databases were available but none for general systems with all vulnerabilities to become a general reference.

The process for creating a vulnerability starts with finding a potential vulnerability in a system. The CVE id is then created for referencing this particular vulnerability by an authority called CVE Numbering Authority (CNA). The CVE ids are using the format CVE-YYYY-XXXXX, where Y is the year the vulnerability was discovered and X is the id of the vulnerability. Previously only 4 digits were used to classify the CVEs, but with more vulnerabilities being discovered every year. The CNA changed the ids to include as many digits as necessary with a simple expansion of a digit to include 10 times as many ids.

The CVE dictionary is used by many organizations and various security products are made compatible with CVE. NIST have advised the use of CVE ids for security vulnerabilities and have made the National Vulnerability Database[13], which is synchronized and based on the CVE register. The CVE ids are used to have a point of reference, when talking about security vulnerabilities especially in literature and articles. The CVEs are further investigated by Common Vulnerability Scoring System (CVSS), which evaluates the vulnerability based on several metrics to assign a score for the severity impact on the system.

2.5.2 CVSS

[14]The Common Vulnerability Scoring System is a third party entity for scoring the CVEs. The score is split into 3 matrices of scoring the vulnerability, which are Base Metric Group, Temporal Metric Group and Environmental Metric Group. The Base Metric Group consists of Exploit metric, Impact metric and the Scope for the scoring. The Base Metric Group is the only required group for scoring the vulnerability, while the other metric groups depend on the vulnerability exploitation and the environment

of the system with the vulnerability. The score is a severity score for the vulnerability ranging from 0 to 10 with 0 being a low risk vulnerability and 10 being a critical risk for the system. The CVSS investigates the CVEs and if a vulnerability is found, the vulnerability is given an evaluation, but the vulnerability have the possibility of being rejected as well. A CVE being rejected means the registered vulnerability does not grant additional access into the system and is thus not given a score.

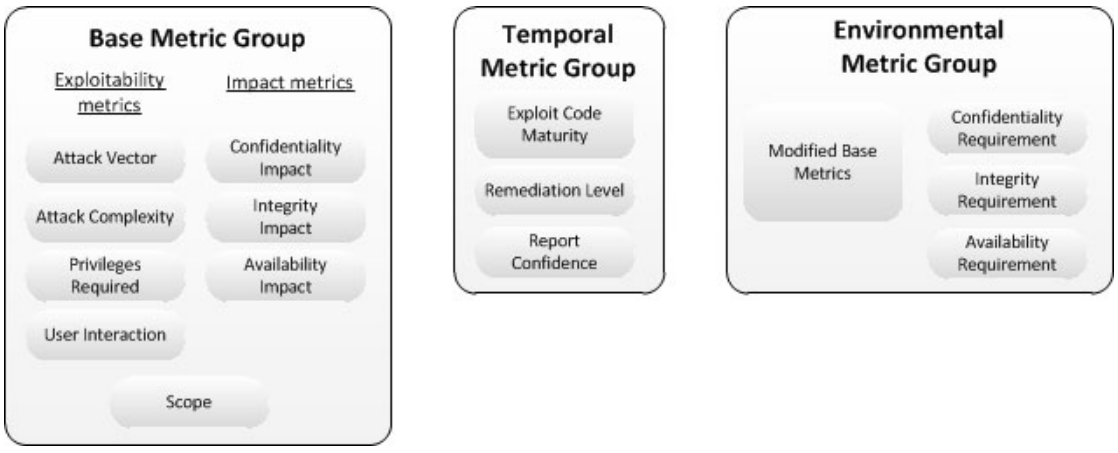


FIGURE 2.9: The CVSS version 3 metric groups for scoring vulnerabilities in CVE. The Base Metric Group is required for the score, while the Temporal and Environmental are optional depending on the vulnerability. The result is a score based on the severity of the vulnerability ranging from 0 to 10.

This project focuses on Open Source Software, while CVE and CVSS show an vulnerability in any software system, hardware system and network resource. The vulnerability does not have a boundary for the systems, which are scored by MITRE and can be any kind of system including Open Source Systems.

2.5.2.1 Base Metric Group

The metric base metric group is split into 3 types of metrics, which can be seen in figure 2.9. The Exploitability Metrics, Authorization scope and Impact Metrics, which scores the vulnerability in different basic aspects. As previously stated these metrics are required for an CVSS score to be assigned, as these metrics contain standard information for a vulnerability.

The **Exploitability Metrics** are metrics to rate the exploit or attack, which the vulnerability is exposed to. The metrics are Attack Vector, Attack Complexity, Privileges Required and User Interaction.

The **Attack Vector** is based on the entry point of the vulnerability. The connectivity needed for an attacker to exploit the vulnerability. The score is evaluated with higher

severity for the access over the Internet or otherwise open network access, while lowest score is in case a physical access is necessary to exploit the vulnerability. The **Attack Complexity** describes the exploit complexity needed for a successful attack. These complexities can be information needed about the system, the configuration of the system or certain elements out of the attacker's control. The lowest complexity needed results in a higher severity score, while the more complex the attack the more unlikely the vulnerability is to be exploited by a large number of adversaries.

Privileges Required for the exploit specifies the user privileges in the system an attacker need for an attack to occur. The attacker does not have to qualify for these privileges himself, but need to receive or attain these privileges in one way or the other. No privileges deem the highest score, while administrative or harder user privileges result in a lower score as they are more difficult to achieve. **User Interaction** relates to requiring a user's help to exploit the vulnerability. The user might need to configure the system in a specific way or leave the system open and vulnerable for the attacker. No user interaction gives the highest score, while if a user is needed the score is significant lower.

Authorization scope scores the vulnerability for a system granting access to another system or a host system. An example could be a vulnerability in a virtual environment granting access to the environment, which hosts the virtual environment. The change of the environment would be a severe risk to any system as many servers hosts virtual servers, where the hosting server should not be accessible to most of the users in the system. The change of the system would result in a severe score.

The **Impact Metrics** are based on the impact of CIA principals, which stand for Confidentiality, Integrity and Availability. The Impact Metrics are thus Confidentiality Impact, Integrity Impact and Availability Impact, which are the factors the vulnerability can impact on the system. **Confidentiality** is used to control the flow of information only for the individuals or systems authenticated. The Confidentiality Impact is high, when an attacker be granted access to information without having the privileges in the system. **Integrity** is the trustworthiness of the information and the source of the information. Integrity Impact is in case an attacker is able to change or destroy information in a system and the system believing the information originated from the original source. **Availability** is the information being available to the system and its users. The Availability Impact can range from total loss of information to no impact at all. The Availability is impacted in case the bandwidth is low from the server and the information cannot be made available to all the users. An example of Availability Impact can be a DDoS attack, where computers send a large number of requests to a service and the service is not able to handle the amount of requests. The service is thus not able to

make the information available to the actual users requesting the information or not all of them because of the server load.

2.5.2.2 Temporal Metric Group

The Temporal Metrics are a description of how well defined and exploited the vulnerability is. The Temporal Metric Group consists of the elements Exploit Code Maturity, Remediation Level and Report Confidence, which as stated earlier is not required for the CVSS scoring but will influence it if presented.

The **Exploit Code Maturity** explains how mature the exploitation of the vulnerability is developed as a piece of software. Is the exploit an automated software like a virus or a worm, is it a script for people to use, or is it developed especially for a single purpose of a single attack. These variable does make a remarkable difference for the severity of the vulnerability from a conceptual idea to an autonomous worm.

The **Remediation Level** is the state of the software having this vulnerability. The vulnerability is often fixed if the severity is high for the system and thus actually only a vulnerability until the issue is fixed by the company behind the system or another entity. The system is vulnerable in this exact version of the software and possible earlier, where the lowest score is an official fix from the software company. The other entities of a remediation or mitigation are a temporal fix, a workaround for the software to mitigate the vulnerability to no fix at all, which would be the highest score for the vulnerability.

Report Confidence simply describes the confidence of the person or organization, which found the vulnerability. The confidence can include the technical specification of the report and the details in which the report is described.

2.5.2.3 Environmental Metric

The Environmental Metric describes the environment and organizational infrastructure the system acts within, and the impact to the organization in regards to Confidentiality, Integrity and Availability. The Environmental Metrics contains the Security Requirements and Modified Base Metrics. The **Security Requirements** are described in terms of 3 factors Confidentiality Requirements, Integrity Requirements and Availability Requirements, which in terms describes the severity of the vulnerability impact to the organization by the 3 principals. The Requirements are given a score from High to Low, depending on the impact on the individual requirement and is only taken into consideration if the Modified Base Metric is not None. The specific organisation might be

responsible for many confidential documents, and the security requirements for Confidentiality will be high for this organisation.

The **Modified Base Metrics** is used by the analyst, the person who found the vulnerability, to describe the environment, which the software is running in. The analyst can be part of an organization, which uses the software and the access controls might be configured differently from the standard product, which results in a severity score deviating from the standard base metrics. The system environment can also include other services, which mitigates the vulnerability severity for the system infrastructure.

2.5.2.4 Outcome of the score

The score given is a combination of all these variables and their rating by the First, where the different Metrics have different constants for each possibility to result in an overall score. The score ranges from 0 to 10, where 10 is for a critical severity. FIRST has decided to use the severity levels in table 2.2.

Rating	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

TABLE 2.2: The division of severity levels based on CVSS score by FIRST

The score is an easy way to find out how severe the vulnerability reported is, but how the different factors influence the score can be seen in their Vector String. The string consists of abbreviation and evaluation results of the different metrics for the CVSS Score. An example could be the following Vector String for the Base Metric Group.

CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:L/A:N
CVSS:2.0/AV:N/AC:L/Au:N/C:P/I:N/A:N

The string is in the same sequence as presented previously and if more information is wanted on the CVSS Score this can be found at First's CVSS page[14].

2.5.3 Heartbleed

Heartbleed[15] is an example of a vulnerability, which is well known from 2014 in the OpenSSL project. Heartbleed is a vulnerability in version 1.0.1 until version 1.0.1g,

which fixed the issue. Heartbleed is a famous modern vulnerability, where most online communities and social networks were affected by the vulnerability. All the users of the systems had to change passwords and the media coverage was high during a period.

The National Vulnerability Database[16] contains information from CVE and CVSS on the vulnerabilities. The CVE id for Heartbleed was CVE-2014-0160 with the CVSS score of (5.0). The problem with Heartbleed was that hackers with network access, mostly through the Internet, could receive the user passwords in vulnerable systems and act on behalf of the system. The vulnerable systems were easily exploitable for hackers, and The OpenSSL X.509 signatures used in the encryption was revealed in memory, which meant that anybody could sign as both the user and the server. The vulnerability was a great problem on the web, but most organisations fixed the vulnerability in a hurry because of the severity.

The vulnerability vector for CVE-2014-0160 can be seen below:

CVSS:2.0/AV:N/AC:L/Au:N/C:P/I:N/A:N

The vector defines that Access Vector is Network exploitable, Access complexity is low and Authentication needed is none in the Exploitability score. The score is thus 10, as it is the easiest possible to access the system. The Impact Vector only includes the confidentiality to be partial whereas the others principles are not impacted.

The information is thus available for anybody over a network to access confidential information of the system impacted by the Heartbleed vulnerability. The systems impacted quickly asked all users to change their passwords, when the vulnerability was fixed on their system. The public was well aware of the fact that the Heartbleed vulnerability happened to most major and minor servers.

2.6 Vulture Mozilla project

Neuhaus et al.[17] have created a data mining and machine learning implementation called Vulture back in 2007, which can predicts vulnerable components in the Open Source project Mozilla. The article is very well written and interesting reading on how data mining and machine learning can be utilised within the Mozilla project. The Mozilla project is well known for their Internet browser Firefox and mail client Thunderbird, which in 2007 was the 2nd most used after Internet Explorer and Outlook. While Chrome has passed Firefox, Firefox is still the 3rd most used browser accessing the Internet. Mozilla have a core project called Mozilla Core, which contains utilities for all their

products, which is likely the most of the used source code for the data minning and machine learning. Vulture data mines the Bugzilla database to find vulnerabilities within the Mozilla project. The data is then used in order to find the correlation for imports and function calls between the components and their vulnerabilities. The Bugzilla is the Mozilla project database with all the bugs found within the Mozilla project, where Vulture data mines the bugs with security vulnerabilities.

The Mozilla is a large project with 3,1 million lines of code for Firefox as of December 2007 and have grown to 14 million lines. The project is huge with a large community to develop throughout their projects, which mainly consist of Firefox and Thunderbird, but other projects are created by Mozilla too. Mozilla also have many contributions to extensions and additional functionality added to both of their largest products from 3rd party sources.

> ”Mozilla as of 4 January 2007 contains 1,799 directories and 13,111 C/C++ files which are combined into 10,452 components. There were 134 vulnerability advisories, pointing to 302 bug reports. Of all 10,452 components, only 424 or 4.05% were vulnerable.” - Neuhaus et al. page 531

The first part of the project is to discover patterns within the Bugzilla database in order to find components, which have been vulnerable. The Mozilla project is well controlled and the bugs are found in the source code by looking for the bug id. The bug id is given in the source code where the fixes are classified by ”Bug #362213” or by ”fix 362213”, which eases auditing the bugs. The bugs are with this notation assigned a component. In the source code Vulture finds the function calls as well as the imported library in the classes of the component. The idea is to find the security vulnerabilities in regards to the library’s import and functions.

The components with security vulnerabilities are linked with the imported library and used function to find support, recall and significance within the data. The support shows vulnerable components with libraries and functions in common, which can be used to find the components possible being vulnerable and not yet discovered.

The second part is to make a prediction based on the data, where Vulture can predict if the component is a security risk based on the libraries used and function calls. The prediction is done with a machine learning classification called support vector machines (SVMs). The resulting classification is incredible fast, and the authors say that a real-time implementation would be possible although, only possible for the systems working with the Mozilla source code or with similar libraries. Using 2/3 of the data for training the classification, and the last 1/3 for evaluating the classification, which is standard

for machine learning classification. Vulture is able to predict with a 45% precision for imports, while predicting 70% function calls. Mozilla will have bugs and security issues that have not been found, but with all the data from Vulture, Mozilla will be able to find the most likely places with security problems. The precision can be lower caused by the fact that not all issues have been found, but is a good result based on a single project.

The concept used in Vulture is a great and innovative way of finding libraries, which are often faulty or incorrectly used. The requirement for creating a classification of the libraries and functions, is the Bugzilla database. Open Source projects probably have a database filled with bugs to correct, but the authors gained access to the database from the community. Gaining access to bug databases in all communities would be a great challenge for giving an evaluation on trustworthiness to any Open Source project. Vulture could be expanded with more data from other projects for an even better indication on, what libraries are most likely to cause a security threat. The problem is though, that the projects are based on different programming languages, and a large data set would be required to make a universal database with the hazardous libraries. The method is a good example of data mining showing its usefulness within a project, but unfortunately the method is unlikely for a general trustworthiness evaluation.

2.7 Maintainability

Maintainability is closely linked with the attribute Complexity, as more complex software is more difficult to maintain. Complexity has an opposite correlation with Maintainability, since a system with low Complexity has a high Maintainability. Maintainability can be seen as the opposite of Complexity, which is an elegant way of measuring Maintainability. Complexity has a large set of metrics to indicate the complexity of the software. The metrics all have advantages and disadvantages in the usage, and how well known the metrics are. The concerns with the metrics most often lie with the comparison between programming languages with their different syntaxes.

Hassan Bhatti's Master Thesis[18] gives an overview of the following complexity metrics.

2.7.1 Lines of Code

Complexity can be measured with the simple Line of Code (LOC), which is very common and well known. The Line of Code measurement describes complexity indirectly by the size of the overall project. The advantage of the Line of Code is the ease of computation, which is as simple as can be with just counting the amount of lines in the source code.

The disadvantage is as well the simplicity, because in itself Line of Code describes the size, which can increase the complexity for larger systems, but is not necessarily an exact correlation with size.

Line of Code can vary greatly with the implementation of the software, programming language and experience of the developer. The implementation of the Line of Code software can choose to count the commented lines or blank lines, where other implementation does not. The Line of Code seems simple to count, but the implementation can make a big difference for a large project, if counted differently. An example could be Mozilla Firefox[3] in table 2.3, which shows 72.7 % of the source code being actual code with the rest being blank lines and comments. Leaving out the comments and comparing Lines of Code to a similar project with the comments would reveal a significant different which is not present. This is a huge issue, when comparing Lines of Code from different sources.

Line type	Code Lines	Percent Code Lines
Code Lines	14.045.424	72.7 %
Comment Lines	2.825.225	14.6 %
Blank Lines	2.452.943	12.7 %

TABLE 2.3: The Lines of Code for Mozilla Firefox found on OpenHub[3], showing the distribution of lines of code compared to blank and comment lines. Only 72.7 % percent of the code is actual code.

Line of Code in different programming languages will similarly reveal a difference. Comparing indentation structured programming languages like Python with normal Object-Oriented programming language as C++ or Java using brackets, will reveal a big different for larger projects. In Object-Oriented programming an entire line will often consist of a curly bracket, where indentation structured programming language will not, which will create a significant different between the programming languages.

The developer's experience will reveal a difference, where developers with greater experience will be able to make a more compact and sophisticated solution. The novice will use more lines of code, and seem like a more complex solution although the solution would result in same functionality with better Maintainability.

Several extensions are possible for Lines of Code, where **Effective Lines of Code**, **Logical Lines of Code** and **Comment to Code Ratio** are a few solutions. Effective lines of code removes the lines with comments, blanks and standalone brackets, and thus removes a few of the previously stated concerns. Logical Lines of Code counts the amount of lines ending with a semi-colon, which makes it only applicable with some programming languages. Comment to Code Ratio is calculated by finding the percentage of comments compared to the Lines of Code. The extensions have a few disadvantages

as well, but tries to remove other disadvantages. The best solution for a Line of Code metric would be Effective Lines of Code, as it removes disadvantages without creating new disadvantages. The comment to code ratio does not create new disadvantages, but gives a good suggestion of understandability of the code.

2.7.2 Halstead formulas

Halstead formula[18] tries to remove the factor of the programming language by using software vocabulary and program length. These indicators can be calculated to the volume and effort of the system's source code. The vocabulary is the sum of distinct operators and operands, and the program length is the total count of operators and operands in the software. The effort indicates, how much effort is put into the system, and from the effort a calculation of the development time is possible. These indicators can be used to compare systems, but Halstead have academic critics with regards to a few indicators. The indicators are thus not unilateral from all academics perspective.

Halstead formula or calculations have to be familiar with the programming language in order to recognise the assignment and usage of variables. The implementation of Halstead formula is thus programming language specific in order to get the calculation, but the result will make the programming languages comparable. An implementation would likewise be significant longer to calculate compared to Lines of Code as the code has to be examined in more details.

Maintainability Index can be calculated as an extension of the effort and volume along with other indicators and can be calculated from a single factor by:

$$MI = 125 - log(avgE)$$

Where MI is the Maintainability Index and avgE is the average Effort per module. The maintainability index is found in figure 2.4, which shows how the ranking of the maintainability values is allocated. Visual Studio uses Maintainability Index to show the developer the level of maintainability of the software.

MI value	Color code	Maintainability
0-9	Red	Low
10-19	Yellow	Moderate
19-100	Green	Good

TABLE 2.4: The ranking of maintainability corresponding to the Maintainability Index

The Maintainability Index can be calculated from several more metrics and result in a more specific Maintainability Index, but this would complicate the calculations even further.

2.7.3 ABC Metric

The ABC Metric was developed in 1997 by Jerry Fitzpatrick as an alternative to the Lines of Code approach. The ABC Metric on the programming languages' fundamentals of the time, which are storing data in variables, branching and test conditions of the variables.

1. **A**ssign data to a variable located in the memory.
2. **B**ranching the software flow by calling functions.
3. **C**ondition the software flow based on variable values with if-sentences.

The assignment is used in programming languages to save a value in the software and massively used for a generic flow of the software. The assignment might vary from programming language to programming language, but is easy to find throughout the code. Branching is used for every function called in a program, where a piece of software is reused for a generic functionality. The branching is part of the software principal not to repeat code. Conditioning is used to split the software flow based on the value of a variable, and in most programming languages an if-sentence or the alternative is used to control the flow.

The ABC Metric is thus a 3 dimensional vector with a number for each of the counts in the software. The length of the vector can be used as a measure of the system size. An example of a vector could be $< 5, 4, 3 >$, which means 5 assignments, 4 function calls and 3 conditions are present in the examined code. The size of the vector would be:

$$|ABC| = \sqrt{5^2 + 4^2 + 3^2} = 7.07$$

2.8 Dependencies

Software re-using other software libraries or systems will have a relationship to the systems, which are used as the original system is installed. Easing the installation of software is assisted by package managers for the different operating systems. In the Unix based

operating systems package managers are very common with the Linux distributions having their own package manager to install all kinds of software products. MacOS similarly have different alternatives for a package managers such as Home brew, MacPorts and Apple's own Apple store, and Windows mostly have their own Windows Store but otherwise the package managers have not gotten the same tracktion.

A well known package manager and one of the first package managers is Debian Package (dpkg), which is used as inspiration for many package managers for Linux distributions. The relations for the dependencies in dpkg is found in the Debian Policy Manual[19], which describes all the possible relations between the systems and libraries. The relations have been used or a modification of this relationship have been used with many of the other package managers. The possible relationsships are *Depends, Pre-Depends, Recommends, Suggests, Enhances, Breaks and Conflicts* to describe the different relationships.

Depends has a strong dependency to the package and will only be installed if all the dependent packages are correctly installed or configured. The Depends relationship should only be used for a dependency to a package if the package is required for a significant set of the functionalities. The Depends relationship allows circular references to take place between the packages, and the smallest circular reference possible is when package 1 depends on package 2, and package 2 also depends on package 1. **Pre-Depends** are similar to Depends, but requires the dependent package to be fully configured before installation to be able to install the package, where Depends otherwise can install the packages simultaneous. The Pre-Depends does not allow circular references as Depends does. **Recommends** states that the packages should be installed together, but is not a requirement for the configuration or installation of the package. The packages with the relationship should be installed together in most cases, but can be installed without the Recommended package in unusual installations. **Suggests** will be an improvement for the package functionality, but is not a strong relationship between the packages. Similarly **Enhances** is the opposite relationship with the package enhancing the functionality of another package.

Relationships which are not as productive for the package are Breaks and Conflicts. **Breaks** is used to describe if a package installation will break the other package installation, and the package cannot be installed without the other package being dis-configured. The Breaks will usually be used as it will expose a bug or interacts poorly with a specific version of another package. The **Conflicts** is a stronger restriction and will not allow the packages to be unpacked on the same system. The Conflicts can be caused by the 2 packages using the same file or similar, which can have been fixed with a later version.

The different relationships help the package manager to install all the packages available, and other package managers have merged a few of the relation types or does not include

the Breaks or Conflicts relations. The dependencies for a software system is an easy way to re-use others software system for functionality. The dependencies are equally a possible security risk, as the dependencies of a software can include security issues, which was otherwise mitigated in the design or implementation of the system. The dependencies can thus be dangerous to use, if the package include vulnerabilities that would create a back door in the system without mitigating actions. The CVE register can be a place to look at for vulnerabilities, but if the system is small and seldom used, the possibility of unknown vulnerabilities can be a risk for the developed system. The system can mitigate a vulnerability by looking into the design and make restrictions in the interface between the packages. Another vulnerability can be created, if the package is used in an unintended situation or construct, which the package was not intended for. The risk of implementing a package library should be known, when deciding on implementing a functionality or re-using another packages for the functionality.

2.9 Team

Open source projects teams are mostly different from software companies with managers utilising project management theory to create a productive environment for the teams. The organisation of open source can vary greatly as described in section 2.1.3 with Vendors typically being software developing companies, and thus Vendor oriented organisations develop software similar to general software companies. Contributor oriented organisations cooperate more remotely with a Project Core in charge of distributing labour with the rest of the contributors. The Core of the Project is typical the most experienced developers, knowledgeable on the project and might have been the founder of the OSS Project. The Core would be the more experienced and the individuals could ask for assistance with a task. The Core will create tasks, which are either assigned or chosen by the contributors themselves, and the tasks are created to work toward the overall goal of the project.

Looking into Project Management People have an exceptional importance for projects, and here is a great difference from a remote software project to team working in collaboration to complete the tasks. Teams and groups work together and communicate in order to create the best solution and discuss the best architecture and design to fulfill the requirements for the system. The tasks in the open source software project will be assigned to an individual, which is then in charge of coming up with a design solution and implementation. The implementation and design is then accepted or changes are required for the solution to be of acceptable quality. The close collaboration between colleagues result in a better outcome with input from the team. Working in teams or

groups, depending on the dynamic of the team, can be a great asset for companies, but the organisation of an OSS Project work as individuals. Theories on team phases are often used when creating teams, and the different phases are called *Forming, Storming, Norming, Performing and Adjourning*. The processes are used to create a well functioning team along with choosing the team members' personality types to create the best performing team.

Teams are usually chosen by the manager based on personality types to create a team with all necessary skills for the task. Personality tests are often a standard part in the candidate selection, when hiring an individual for a job. Many different personality tests are available for companies to use, and the well known *Belbin*[20] test will give an indication of strengths and weaknesses working on a project. In open source projects teams are not utilized to work on tasks, instead contributors are assigned tasks to work on. All team benefits are thus lost for most open source projects.

Working remotely is a key factor for an oss project, since all or nearly all the members of the organisation live apart and are not able to meet regularly to discuss the project. The organisation will need a guideline of how the information is distributed to all members, since any member might be in need of finding information on the design and implementation of a specific component. The communication guideline will need to include a system for storing documentation correctly, storing information on the line of communication in the organisation and who completed the different tasks in the project.

Open source projects are lacking in a few general concepts, but in case of motivation the contributors can often be stronger. The motivation in an open source project is mostly based on the a personal interest in the project or the product. The interest can be in the product, and its usefulness in every day lives for the contributor or even organisations. The contributor is often motivated by developing skills of excellence from the experience with software development with more experienced developers. The experience gained can be a great asset for the contributor in becoming a better developer. The contributors are mostly developing in their spare time by interest of Software development or is already a software developer but want more experience for their profession. The motivation of the contributors are based on the individual, and a contributor can be busy with many other parts in their life and only use little time on developing oss. Vendor hired developers can be motivated by the experience gained in a project, but is mostly from interest in the industry and the salary paid by the employer. A developer can be motivated similarly to a contributor, but often not to the same extend.

The difference between professional developers and contributors are greatest in the ways of collaboration, where the developers have each others experience closely for a great solution and greater collaboration. The contributors thus can be more motivated but

this is purely based on the contributor as an individual, but the active can be very motivated by the purpose of a project.

2.10 Summary

Trustworthiness in Software can be based on many aspects of Software engineering, but the recurrent aspect of trustworthiness seems to be security. The security of software and open source software as well is easily surveyed by the Vulnerabilities of the system, which by the CVE is managed by Mitre with easy access to the Vulnerabilities discovered in a system. Mitre have created a general way of reporting and managing vulnerabilities in a system, while CVSS managed by FIRST is made for creating matrices for rating the vulnerability severity. Trustworthiness can be rated by other factors seen in figure 2.7 or in appendix A.

Open source software Projects are not managed in one fashion, but can be very different from one another depending on the ownership of the project, and with the ownership the way the project is organised will be quite different. The inner workings of a project will differ from a Vendor software solution to a project with only contributions from spare time developers. In a contributor oriented project a number of people decides the direction of the project compared to the more general industry decisions made by a manager just as for a Vendor oriented project.

As open source software projects are mostly done by contributors individually, many of the team theories on project management are not as relevant for OSS Projects. This does not mean that the team perspective is invalid for OSS project. The contributing developers' experience and skill in software development is still a great indicator for the overall quality of a software product. The developer with greater experience will deliver a better software design and implementation to the project. A key position in an OSS project is the individual in charge of accepting the contributions, and he will most likely have a higher standard with more experience, which results in a better product.

The product of an OSS project can easily reuse other software to gain more features by adding a dependency. The dependency can also be a vulnerability into an otherwise secure system, since the dependency can allow unintentional access into the system. Dependencies are quite common for OSS projects to extend the requirements of the project. The dependencies will have to be thoroughly examined if the dependency is of a good quality and if specific restrictions have to be made for the implementation.

The OSS project code can be rated using several matrices to find the best complexity and maintainability measurement. The different matrices does have both pros and cons for a

neutral rating in terms of the different programming language, although many projects have a combination of programming languages. Data concerning comments in the code indicates the level of understandability, as the code is thus easier to comprehend for developers without prior experience with the code. The maintainability is influenced by the understandability for the developers, who have to maintain, extend or update a previous implementation.

Chapter 3

Analysis

Trustworthiness in software as found in the State of the art section 2, is not simple to determine for all relevant aspects of software engineering. The trustworthiness will have to be based on different aspects, which of time constraints cannot all be completely analysed and implemented into a software trustworthiness analysis. The metrics will have to be chosen and analysed in order to determine the relevance and usefulness in terms of trustworthiness.

3.1 Metrics decision

Metrics have many possibilities for measuring *trustworthiness*, but a few metrics are mentioned in most papers, where the top contender is *security*. As previously mentioned security has been the only parameter of trustworthiness in the early years, then articles started suggesting other aspects of software engineering. Security is then a must have when considering trustworthiness. Security with others are mentioned below with reasons for being chosen or discarded.

Security is an essential part of trustworthiness, where an insecure system would not be trusted by any user. An insecure system would at worst expose their information to any with interest. With most systems today security is essential, otherwise the media would create a storm of bad press for the people behind the system. Security and privacy is a great concern for most people in the modern day with all the information shared on the Internet. A simple example of security risk is the Heartbleed vulnerability, which showed the alertness with security vulnerabilities in larger system, where the repair was made quickly for major systems. The users would otherwise stop using the services if they knew none of their contents were secure and their accounts were hacked with ease.

Security is a problem when users are only authorized to see certain parts of the system, which is the case for much of the Internet and many services connected to the Internet. Security can also be the data handled in a piece of software like customers data and orders for a company, where the data is not directly connected to the Internet, although most computers today are connected to the Internet to communicate with other computers. Security is thus a concern in all software and without security the trust would not exist to a system or a piece of software. Security is thus a must in any metric for evaluating trustworthiness.

Maintainability is linked with security in the way that for the vulnerabilities to be easily and quickly fixed, the source code needs to be highly maintainable. With Heartbleed the the implementation of OpenSSL needed to be fixed to re-secure the system, and with high maintainability the fix would be easier for the developers to develop. Easily maintained code does not directly interfere with trustworthy systems, but security, complexity and availability is connected to maintainability for fixing potential issues. Complexity is closely connected to maintainability, since high complexity will result in low maintainability. A well known software principle called KISS ("Keep It Simple, Stupid") is used to keep software simple, since simpler software will work better and be more secure. Maintainability is not the largest factor in trustworthiness, but does play a part in developing a trustworthy software system.

The *team contribution* is not directly influenced by trustworthiness. The environment software created is influenced by the collaboration and communication between the contributors to the project. The organisation management does play a large part in an successful or failing project. A well functioning team can be a resource for a project, where Project Management have a large set of processes and models to improve the team work in projects. The groups working on open source project mainly work individually and thus will their experience and skill be the most valuable to the general project. System design and requirements are essential parts in any system, where the quality of the design is increased greatly by an experienced software developer. The developers with greater experience will thus develop better solutions and expect the less experienced to contribute with equal quality of software. The core of the project will influence the overall quality of software and revise the other contributors' work. The experience and skills of developers are hard to evaluate, and their contribution individually will be difficult to assess. The contributing parties can be scored by the portfolio of projects by the quality of software. The overall team can be evaluated by the quality of all the developers contributing to the project, although their portfolio can give an indication, it might not give the entire story of the developer. The developer can have contributed to Non Open Source projects or only have created great designs from the revisions by other developers. The contributor might equally not have played a major part in the

development of the project, but still from the portfolio be given credit of the project's success.

The *design* and *requirements* of a project are hard to quantify for projects in general. Different projects will have a design fit for the solution to specific requirements, and an overall good design cannot be quantified without using requirements and purpose of the project. The type of project usually plays an important part of quantifying the project as successful in terms of their design and requirements. The quantification of the design and requirements are difficult to determine in general for all possible systems. A model checker can be made to check the requirement of the implementation's success in fulfilling the requirement, but creating general models for all thinkable requirements would be not just another Thesis but an entire new comprehensive project.

Other aspects like *performance*, *availability*, *usability* and *configuration management* are all specific to the OSS product and are difficult to quantify too. The performance is specific to the individual task and current technology with what performance is acceptable to the user at the moment of use. Performance and availability in general might be possible to quantify for product types, but not all kind of software products have performance and availability as a requirement, although most do to some extend.

In general many of these aspects of software engineering, are hard to quantify automatically by a trustworthiness metric. Most aspects are based on the type and environment of the product and creating an independent quantification of one of these aspects could be quite comprehensive. The aspects would probably be enough work for another Master Thesis or research project, and all these aspects will thus not be covered in this Thesis.

3.2 The metrics

The different metrics are created based on the State of the art section to find accurate metrics to describe the trustworthiness of an OSS project. The different metrics will be explained below with arguments for the choice of the metric and the calculation of the individual score.

3.2.1 Aggregated Security score

The Aggregated Security score has several different metrics to describe the trustworthiness of the project and its dependencies by examining the vulnerabilities discovered. The Aggregated Security score is a metric describing the security aspect of the trustworthiness evaluation by looking at vulnerabilities using the data from CVE and CVSS'

metrics. The Aggregated Security score is aggregated with the score being described as the lowest score found with the project itself and all its dependencies. The Aggregated Security score was developed by Christina García[21] as a Dependency score and has not been changed, although a few issues with the Dependency score will be described. The Dependency score has not been changed as the development of new metrics was prioritised. The script have been redeveloped to have a single program for the entire evaluation and in a more academically used programming language.

The number of CVE ids represents the number of vulnerabilities found, and at any point a low number of vulnerabilities is preferred. The criticality of a vulnerability is rated by the organisation FIRST by CVSS on a 0 - 10 scale. On the scale 10 is a vulnerability with critical severity and 0 has a low severity level. A large number of vulnerabilities with low severity is preferred to a few critical vulnerabilities, as the low critical vulnerabilities might not gain access to much information in the system, where the critical vulnerabilities can gain access to the entire system.

The Aggregated Security score consists of the scores for the vulnerabilities and severity and are weighted as seen in equation 3.1.

$$aggregated\_security\_score = 0.2 \cdot vulnerability\_score + 0.8 \cdot severity\_score \quad (3.1)$$

The vulnerability indicates the general amount of CVEs and the project's ability to create secure software with less vulnerabilities. The vulnerability score calculation is found in equation 3.2. The severity score is an indication of how the severity levels of vulnerabilities are progressing for the project, and the calculations are found in equation 3.6. The weight between the vulnerability score and severity score is decided as the criticality of the vulnerabilities is more important than the amount of vulnerabilities overall. The amount of CVEs might not explain much, whereas the severity of the vulnerabilities in the system will indicate greatly how secure the project is. The severity should thus be a significant higher factor in the weighting for a score of trustworthiness.

$$vulnerability\_score = grade\_ncve \cdot grade\_tcve \quad (3.2)$$

The vulnerability score is examining development of the vulnerabilities for a project. The vulnerability score is made up by the average number documented each year for the project and is represented by the factor ncve, which is graded depending on the average amount of vulnerabilities. The ncve calculations can be found in equation 3.3. tcve represents a grade based on the trend of the amount of CVEs documented each year, the

calculations can be found in equation 3.5. The CVEs are all treated equally equally and with no mind of the severity. The vulnerability score indicates how the project's quality is progressing.

$$grade\_ncve = \begin{cases} \text{User evaluation} & \text{if } ncve <= 5 \\ 0.7 & \text{if } 5 < ncve <= 20 \\ 0.9 & \text{if } 20 < ncve <= 70 \\ 1.0 & \text{if } 70 < ncve \end{cases} \tag{3.3}$$

The **ncve** is the average value of CVEs found annually from the project started or after the MITRE started documenting the CVEs in 1999. The CVEs are simply evaluated as one entity and the mean value is calculated for the vulnerabilities found annually. The amount of CVEs do discriminate against larger projects, because a project with 20 million lines of code are more likely to produce a larger number of vulnerabilities compared to a project in the tens of thousands lines of code. This can be an issue with the evaluation being biased against larger projects receiving a worse score mainly caused by the size of the project. Several solutions are possible with either finding a value comparing the size of the project or separating projects into size intervals for comparison with other projects with similar size. Similar projects are possible to compare such as the most popular web browsers Edge, Chrome and Firefox, but projects with different in size are more harder to compare.

The User evaluation uses user data to survey the user state of the project. The data is taken from OpenHub, which have extensive data on a large set of OSS Projects. OpenHub looks through the OSS projects to find data on the programming languages used and contains data on all the different contributors found. The data on the project is for the project available on OpenHub, while a large number of smaller Linux OSS project is not present. The evaluation is based on the numbers of users and contributors found on OpenHub. The contributors can be an unclaimed committer id or account holder depending on the committer id is claimed or unclaimed. The unclaimed committer ids are contributors, who are not active or use OpenHub, but a page on OpenHub is still created with information of their contributions only from the various projects source code, and the account holders are active users on OpenHub and contains additional information on these user. The number of unclaimed committers are significantly larger than for account holders, since OpenHub is just used and known to few compared to the numbers of contributors to a large set of OSS projects.

$$aggregated\_security\_score_{\text{User evaluation}} = \begin{cases} 0 & \text{if } project\ not\ found \\ 0 & \text{if } 500 < users \\ 0 & \text{if } users < 500 \text{ and } 15 < contributors \\ 10 & \text{if } users < 500 \text{ and } contributors < 15 \end{cases} \tag{3.4}$$

The idea of the user evaluation is that project with few vulnerabilities discovered annually can be for different reasons. The projects with few CVEs can annually be either great at developing secure software almost without creating vulnerabilities, or the project can be of a size where the vulnerabilities are not discovered by the small amount of users and contributors. These situations are thus handled by the user evaluation with projects containing more than 500 users or 15 contributors are deemed as secure, while projects with less are deemed insecure. The projects with few users and contributors are rated as untrustworthy, because of the uncertainty in regards to the project situation. Rating a project to be trustworthy that does not exist in OpenHub, is giving the project a benefit of the doubt. Although it is not possible to see the difference between projects with few CVEs and many users and the project that is unavailable in OpenHub, which can cause confusion between the scores. The score of the unavailable projects could be a Null value or value with neutral trustworthiness just not being a completely trustworthy score without a doubt.

The next part of the vulnerability score is the CVE trend, which is a simple linear regression of the CVEs for every year. The evaluation does not focus on the amount of CVEs but the trends, which is indicated by the slope in a linear regression.

$$grade\_tcve = \begin{cases} 4 & \text{if } a < -0.2 \\ 7 & \text{if } -0.2 < a < 0.2 \\ 10 & \text{if } 0.2 < a \end{cases} \tag{3.5}$$

The grade is then calculated with the intervals of the slope from the equation 3.5. As previously explained the preferred trend is the decrease of CVEs for a project, which means the project is not creating as many vulnerabilities in their product. The 7 score is given for a stable progression in terms of vulnerabilities, which is set to be within a range from 20 % decrease to 20 % increase annually. This is a quite large margin to set for medium and large projects, that allows over time is allowing an up to 20 % increase and still be considered stable. Projects of significant size would +20% change would be a tremendous change, whereas for smaller projects a 20% change would be more closely

to a stable trend. The decrease in vulnerabilities will result in a better score as a low score is indicating less severity.

An issue with calculating the trend as the only indicator is that looking at a project in rapid growth will have an increasing number of vulnerabilities if the project is developing with the same standard as before. A project in rapid growth is meant that many users and contributors join the project, and the amount of code is growing quickly. The same can occur with a project that is stagnating and not producing much code for any reason. A stagnating project will not have a lot of vulnerabilities surfacing, as most vulnerabilities are already found and a small amount of code is produced. A fix to this could be to compare the growth of the project source code with the trend of vulnerabilities to create a more neutral metric.

The severity score is created from data based upon the same CVE data, but separating vulnerabilities into severity categories. The categories can be found in table 2.2, which is the categories used by FIRST to separate the CVEs into severity levels. The severity score, similar to the Aggregated Security score, ranges from 0 to 10 with 10 being a critical score for the vulnerability. The higher the severity score the higher severity of the vulnerability. The severity score includes the trends of the severity categories and booleans for high and critical percentage of severe vulnerabilities.

$$\begin{aligned} severity\_score = &(0.6 \cdot average\_critical\_percentage\_great + \\ &0.4 \cdot average\_high\_percentage\_great) + 0.45 \cdot trend\_critical + \\ &0.3 \cdot trend\_high + 0.1 \cdot trend\_medium + 0.05 \cdot trend\_low \end{aligned} \tag{3.6}$$

The average percentage of great variables are a boolean value indicating either the critical or high percentage of vulnerabilities is greater than 25% on average over the project's lifetime. The trend is calculated by the same grade as equation 3.5. The trend grade is more significant for the more severe vulnerabilities as they are a greater risk.

The trends of the critical vulnerabilities are the most significant values in the severity score and thus the Aggregated Security score, because these are the vulnerabilities with high enough severity to ruin users' trust in the system. Thus the critical vulnerabilities should be a score, which causes the system to be less trustworty. On FIRST's scale the known Heartbleed was only given a CVSS score of 5.0, which is categorised as a medium score. Both high and critical CVSS scored vulnerabilities are significantly worse than Heartbleed, which was fixed in no time since the users' authorization information was at risk. Significantly worse vulnerabilities expose IT systems a great deal more, but

medium scored vulnerabilities can be enough to cause users not to trust the system. The worst vulnerabilities will need to be fixed quickly as the project's system is exposed. The status of a vulnerability is not available for all vulnerabilities, but some have text describing a solution to fix the vulnerability. In the National Vulnerability Database[13] some vulnerabilities contain information regarding the implicated versions, although not all contains this information about the software fix.

Information such as the information from Firebug utilised in Vulture project would be very helpful both for other developers using the same software libraries but also for the software libraries to know and fix the issue if the library is at fault. The origin of the vulnerability would be important information in most OSS projects, which could help other projects to improve their security. The information is most likely not available since most non-OSS will not disclose whom was at fault and what caused the vulnerability. Vulture would have been a benefit for many developers and this kind of information would be good, when developing software to know the risk from specific libraries.

The Aggregated Security score will look through all the dependencies of the project and find the library with the highest score in severity. The score with the highest severity is either a dependency or the project will be the Aggregated Security score of the entire project. The idea is based on the weakest link in the chain is often, where an adversary will try to attack the system.

The Aggregated Security score is the opposite of the trustworthiness and the trustworthiness score for the dependency score is thus:

$$Trustworthiness\_security\_score = 10 - aggregated\_security\_score \tag{3.7}$$

The choice of the vulnerability and severity score to range from 0 being low severity to 10 being critical, is based on the measuring of severity for vulnerabilities, and the scoring of CVEs by CVSS metrics. In terms of trustworthiness the score would make the most sense for the scale to be opposite with still 10 being the most trustworthy OSS system, and this is the reason for the confusion between Aggregated Security score and trustworthiness score.

Aggregated Security score limits

The Aggregated Security score ranges from 0 to 10, and the score 0 is given just in case the ncve is less than 5 annually, which is assigned based on the user evaluation in equation 3.4. The user evaluation assigns a score of 0, which means 0 is the lower range,

but what is the real range of the Aggregated Security score without including the user evaluation. The highest possible grade is 10, which is given in the situation where the different variables have the following combinations:

Variable	Upper limits	Lower limits	User evaluation
Trend low	10	4	X
Trend medium	10	4	X
Trend high	10	4	X
Trend Critical	10	4	X
Avg high	1	0	X
Avg Critical	1	0	X
Severity score	10	3.6	X
Grade tcve	10	4	X
Grade ncve	1	0.7	X
Vulnerability score	10	2.8	X
Aggregated Security score	10	3.44	0

TABLE 3.1: The limits for the scale within the code security metric *Aggregated Security score*. The values reachable by the metric is used to describe the trustworthiness in the project

The values are calculated using the above equations to find the lower and upper limits of the Aggregated Security score. The User evaluation is made with a project with less than 5 CVEs annually will either receive a score of 10 or 0 depending on the users and contributions, which can be seen in equation 3.4. The actual range of the Aggregated Security score is thus $[3.44, 10]$, of course specific values between 3.44 to 10 are not obtainable since the grades are quite specific and not a range, but combined with the scores a thorough investigation into the range would be quite time consuming. The range is weirdly only the upper 2/3s of the scale, which could be changed by change a few scores to accomplish this. The score is kept as is, since a change would have to be well investigated how best design the security score, and the time was rather used for further development of new metrics.

3.2.2 Maintainability Score

The maintainability of source code is an important aspect in order for anybody with knowledge of the system or in possession of the documentation can maintain. The readability and thus maintainability is part of the requirements for an OSS project to be

accepted as Open Source, since the code should be easy to understand and read for anybody. The maintainability in connection to trustworthiness is important because OSS product can have contributors, who spend their spare time in other projects or other hobbies, then another developer will need to be able to maintain the code of another developer. Maintainability does not have the same influence on trustworthiness as security with the Aggregated Security score, but it is still relevant. Maintainability is important with the discovery of a vulnerability, since the contributor or another contributor will have to find the source of the vulnerability in the code and quickly find a solution to the issue. Code with worse maintainability will generally take longer to fix, since the developers will use longer time examining the code for errors and find the cause of the vulnerability.

Various metrics are possible to indicate the maintainability and other properties in source code. The simplest metric to calculate is the lines of code to lines of comment ratio, which describes the ratio of comments. The more comments in the code the better documented and descriptive the code is in general. The comments will describe the implementation, while the ideas and algorithms are described within the documentation. The comment to code ratio is the simplest and most descriptive in terms of understandability (and thus maintainability) in source code. The Halstead formula is able to describe maintainability, but academics are debating if the different formulas do in fact indicate, what he claims. The maintainability measure might not give the indication he claims, and is thus a possibility to measure maintainability. The Halstead formulas are significantly more difficult to calculate compared to the lines of code and comments. Another relevant metric is the ABC metric, but this was discarded similarly as Halstead formulas, because of the calculation of the source code. The ABC metric does give a great indication of complexity and size, but with the lines of comment to lines of code ratio is simple and gives a good indication of the understandability.

The maintainability factor is not as important compared to other metrics in OSS project's trustworthiness, and thus a simple solution is preferable. OpenHub contains data gathered from OSS projects with specific data on the source code and the information about lines of code and lines of comment are part of this data. The information is thus easily obtainable on most known and major projects. What is then a good ratio for lines of code to lines of comments, since too many lines of comments will have very little functionality. Giving all the information in the source code instead of writing documentation is not the best solution, but generally the more comments the better understandability. In table 3.2 source code information is given on common OSS projects.

The comments to code ratio is calculated by the following formula:

$$comments\_code\_ratio = \frac{lines_{comments}}{lines_{code} + lines_{comments}}$$

Project	Lines of code	Lines of comments	comments/code ratio
Mozilla Firebug	492,181	350,235	41.6%
Apache Subversion	660,711	208,243	24.0%
MySQL	2,862,087	692,663	19.5%
Ubuntu	911,004	187,691	17.1%
Linux Kernel	18,963,973	3,872,008	17.0%
Mozilla Firefox	14,045,424	2,825,225	16.8%
Keepass	121,090	24,347	16.7%
Chromium	14,945,618	2,752,467	15.6%
Python	1,030,242	184890	15.2%
PHP	3,617,916	587,629	14.0%
Git	774,674	96,554	11.1%
Apache HTTP server	1,832,007	210,141	10.3%
neat project	23708	1695	6.67%

TABLE 3.2: Mostly known projects with information on how many lines of codes and comments were written in the current product as of December 2016. The range of the ratio lying mostly from 10% to 20% for common OSS projects, which are well functioning and is maintained and ongoing projects.

The maintainability score seems to be in the range from 10% to 20% for most projects with a few outliers. The lines of code and lines of comments can be affected by the different programming languages, since the expressiveness is different for the various programming languages, especially when having large systems. The expressiveness does vary and the amount of comment lines will thus count more with more compact programming languages. The comments does tend to express the language and the targeted audience is the people with knowledge about the programming language and thus the comments tend to be explaining why and what the code does. The maintainability will calculate a score on the scale from 1 to 10 similar to the score created for Aggregated Security score with a high score indicating high maintainability. The score will show the code to comments ratio to be within a range with both good and bad maintainable projects. The metric can be found in the equation 3.8.

$$Maintainability\_score = \frac{comment\_code\_ratio - 0.1}{0.25 - 0.1} \cdot 10 \tag{3.8}$$

The score will rate a higher percentage than 25% a score of 10 and lower score than 10% a score of 0. Although a score of 5% is a score of 1 and not 0, but this is the benefit of the doubt with the scale from 1 to 10. Although most projects lies within a range from 10% to 20%, the exceptionally and bad still lie within 5 additional percents in the top and bottom of the ordinary range. The scale should take both the exceptional as the top of the class and the worse than average to be in the scale. The scale could range higher than the top of the range being 25%, as Firebug have 41% comments in their code, but 25% being the top is caused by average being from 15% to 20%, which should not get a bad score for being average but a mediocre score instead. A score range being higher would result in a lower score to the average maintainable projects and comments above 25% might not be an advantage.

3.2.3 Team score

As described in section 2.9 about the team behind the OSS projects, which is actually not working as a team, but rather as a group of individuals who work towards the same goal. The core team might work like a group and a few of available group processes from Project Management, can be used on this group. The Core of OSS projects will need to communicate often and make the decision on the direction of the project, which might make it relevant. For most the Project Management theory is not as relevant to OSS projects as the individual contributors' experience and their portfolio.

As the design is difficult to grade on a scale for any type of OSS project, the contributors experience and indirectly their skill to design great solutions, can be graded by their experience. The grading of the contributors' experience is not the most exact metric, but the experience is definitely important to the overall project. Especially the contributors spend a great part of their spare time developing OSS. The contribution to a system is not equal over all the developers, where the Core group will be accepting all the designs and implementation into the system, while the novice contributor in the project might just contribute with a few commits and other more active contributors will contribute with hundreds or event thousands of commits. The expert contributor's contributions are thus far more valuable to the project and equally his experience from previous projects. The experience from developing industry software or closed software can be difficult to collect into a profile for each contributor. The experience in the industry will have to be public knowledge to incorporate in a metric, which industry software and the developers in the projects rarely is.

The experience of a developer in OSS projects are found by looking at the quality of the other projects the contributor have contributed to. The score of a contributor can thus be calculated by the following equation (3.9).-

$$contributor\_score = \frac{commits_{contributor\ in\ project} \cdot (10 - aggregated\_security\_score_{project})}{total\_commits_{contributor}} \tag{3.9}$$

The contributor is measured by the scores of the projects he has contributed to, which are weighted by the amounts of commits in each of the projects the contributor have contributed to. The contributor score is thus calculated by iterating over all the contributor's projects and dividing by the total commits of the contributor. The Aggregated Security score is a severity score and will have to be flipped to indicate trustworthiness instead of vulnerability severity. The Aggregated Security score will only be based on the project itself and not all the dependencies, since the contributor have not contributed to the dependencies and also to limit the already massive amounts of projects to be scored. An issue with the contributor score could be, that several of the projects, the contributor have contributed to, are small and will receive a score of either 0 or 10 by the user evaluation. The contributor score will thus be either very high or low depending on the projects.

The Team score name was given based on the the team even though the team might not collaborate as a team, but the team experience and skill is still a representation of the team. The contribution score thus indicates the overall experience of the contributors in the project. The contribution score is the equation (3.10).

$$contribution\_score = \frac{contributor\_commits_{project} \cdot contributor\_score}{total\_commits_{project}} \tag{3.10}$$

The team score for the overall project will give an indication of the contributors experience in their projects. The score will be weighted with the most contributing developers counting more towards the score, since they will have both more decision power and have developed more of the overall system. The decision power comes from the projects voting on what direction and what is most important for the project, where the democratic structure might differ, but with the developers with most activity and work put into the system should have more power in the project. The developers with the most contributions will be more likely to be part of the Core group of a project with more essential tasks being assigned to the most active contributor.

Calculating the data for contribution score can be a quite heavy calculation with for example Mozilla Firefox having 4,108 contributors working on many other projects than Firefox. I have looked into a smaller project called neat project, which is working to create an alternative to the transport layer in the OSI-model. Neat project only has 22 developers, which is a bit more testable as a project, although even these 22 developers have participated in 158 different OSS projects. Calculating the scores of 158 project will by all means take a bit of time and restrictions will be necessary especially when working with larger projects and even with the smaller ones as well.

3.2.4 Trustworthiness score

The trustworthiness score is simply a combination of all the scores before mentioned. The trustworthiness score is the measurement of how trustworthy the OSS project is, which is calculated by the equation 3.11.

$$\begin{aligned} trustworthiness\_score = 0.65 \cdot trustworthiness\_security\_score \\ + 0.25 \cdot team\_score + 0.1 \cdot maintainability\_score \end{aligned} \tag{3.11}$$

The weight of the variables are decided based on the security being the biggest concern, when deciding to trust a software product. The team score is based on a larger set of projects, which the contributors have contributed to with bigger and smaller contributions. The security of the project and the security and dependency are the biggest indicator, while the contributors' experience from other contributions. The maintainability is a smaller indicator of the trustworthiness of a system, but is a strong indicator of the understandability and the ease of fixing discovered vulnerabilities. The weights are decided based on various data, and 2/3 being influenced by the Aggregated Security score is the clear majority. The

The weights of the trustworthiness were based from experiments with different projects trustworthiness score. The score from the trustworthy aggregated security score (high score for secure projects) was for larger projects often 0, and the security metric was decided to influence 2/3s of the score, while the other metrics share the last 1/3. The experiment showed 3/4 was too much for the security metric and less showed that it had too little influence. The score was thus decided, although with new metrics the influence of the different metrics should be changed, and the influence of the security score can thus be lessened.

The scores in the Trustworthiness score will often be similar with the team score, since the Team score will have contributors with most commits in the project, being active

in a limited amount of projects and mostly in the evaluated project. The Team score will thus be highly influenced by the main project's Aggregated Security score for many projects.

3.3 Information sources

To create a trustworthiness metric the need of data regarding various OSS projects is essential for completing a trustworthiness score of the projects. Open Source information are not hard to find, but creating software to process all the information, is a great task in itself. The processing of the data can occupy plenty of people along with keeping the information updated, and the process is often time consuming with a large data set. Developing all these processes, could occupy many academics and is not possible within the time of a thesis. Finding reliable sources with the necessary information is thus a key aspect for being able to create different metrics to evaluate trustworthiness. The sources will have to contain a large data set on the topic and the individual projects. The relevant sources for the project is CVE information contained in the National Vulnerability Database, dependency information from Debian Package Manager, and Open Source project information processed and available by OpenHub.com[3].

3.3.1 National Vulnerability Database

The National Vulnerability Database[13] is a section under National Institute for Standards and Technology, which is a US governmental institution issuing standards and in this case manages data on discovered vulnerabilities and security measurement. The NVD contains information on vulnerabilities and the security measurements. The NVD utilises the information from Mitre's CVE register and thus already contains information on a large set of vulnerabilities in software. The NVD is linked with the CVE register for the American Government to have all the information along with vulnerabilities reported to NIST as well, which are equally given an CVE id. The NVD also contains the information by FIRST and the CVSS score along with the information on general security measurements. The NVD seems to contain all the relevant information gathered by different sources in a single place, and the NVD as part of NIST would be considered a trustworthy source of information.

The CVE and CVSS information is thoroughly described in the State of the Art section (2.5), which is the main information used from the NVD. The NVD contains all the information, where both MITRE and FIRST have a piece of the information to create

the big picture. The information is available at different places, but the NVD data is simple to access, and all the information is available in one place.

3.3.2 Debian Package Manager

The Debian Package Manager (dpkg) contains information on a large set of available software products developed for the Linux distribution Debian. The package manager is able to install the software product on the operating system on its own and keep it updated. The Debian Package Manager is used for the Linux distribution Ubuntu, which is one of the most popular Linux distributions. The package manager not only installs the software product, but also the dependencies of the product to install and configure the software correctly. The dependency information is created by the software developers behind the individual project for dpkg to understand and install the correct dependencies for the project. Naturally, dpkg does only work on Linux and even restricted to a limited number of Linux distributions using the dpkg for installing packages. With Windows 10, Microsoft allowed the use of Linux software in a Virtual Machine running a small version of Ubuntu within Windows 10, and with this it is possible to run dpkg from Windows. Microsoft has gotten more fond of Open Source and Linux over the last years and this is a gesture that shows their continuous involvement in the Linux world.

3.3.3 OpenHub

OpenHub is a platform with data on a large selection of Open Source Software Projects. OpenHub contains detailed information from the source code and the version controls used by the Open Source Projects. OpenHub is an Open Source project helping users to find information on Free and Open Source Software. The users are able to compare the different Open Source projects and find all kinds of information relevant to specific users.

The data available through the website is of a fine quality with only a few problems. The website is fine, but an api would have been an even better alternative as with an api of the project the information is easier accessible. The api is developed and called Ohloh Api from their previous name, but not all the data from the website is available. The website would be the solution for gathering the needed information for the current metrics, but for the future the api might be an even better solution when all data is available, if this will happen.

The information available at OpenHub can be categorized into following categories *Source Code data*, *Version control data* and *Vulnerabilities*. The source code data contains information on the language distribution of the project with information of the lines of code, comments and blanks. The source code is analysed by OpenHub themselves to make the data available, and they regularly re-analyse the different projects to update the project with current information. OpenHub can thus give information on the development of the project over time. The version control data will give information on individual contributor's contribution to the project. OpenHub creates profiles on the contributors to get an overview of what project the contributors have been contributing to. Each project will show all the developers, who have contributed to the project with information on the individual commit. The project also show the contributors with the most commits to the project, who is the most active and has invested most time in the project. Each developer has a profile for their contribution with statistics of what was committed. A general profile equally exist with information on the developers contributions to other projects and their activity level over the time from first commit to the most recent commit.

OpenHub contains 2 kind of profiles either for accounts or unclaimed committer ids, where the difference is user aliases claimed on OpenHub and unclaimed aliases. Both profile types contain similar information on what projects the developer has contributed to, but the account holders can possible claim more ids and will most likely have more projects. Looking into both profiles the unclaimed committer id is more clear and easier to see the big picture from, while the accounts are more detailed, but have mostly the same information overall. The share of unclaimed committer ids are naturally in a significantly larger number compared to the amount of accounts, since the committers are aliases found in all the projects and the accounts will be people signed up for OpenHub. The amount of users on OpenHub is 31,777 and unclaimed committers ids are 945,635, which equally shows the difference[1].

The information contained on vulnerabilities are detailed with the CVEs grouped by the version of the software and an indicator about the security of the project. The security of their projects are measured from the amount of vulnerabilities and based on the recent and larger versions of the software, which might not have been found yet. The vulnerabilities are in a list for each version of the software product, and most larger projects do seem to be insecure. The security evaluation is interesting, but they do not go into details about the metric used. The vulnerabilities are available at OpenHub, but these are not available using Ohloh either, and web scraping all these vulnerabilities would create an immense task.

[1] As of the time of writing (10-01-2017) these were the amount of users and committers on OpenHub

The measurement of security does seem to be fine, but it is hard to judge when all the data is not available on their web site. The vulnerabilities are possible to get from OpenHub and this might be useful with the data on visioning of the software grouped. The data on severity is only available from the grouping and not the specific CVSS score or any indicator except a low, medium and high grouping. Using the OpenHub data would also ensure that a project is not found to have any vulnerabilities in case of different naming with CVE and OpenHub. The projects in OpenHub seems to be the most known OSS projects, since none of the smaller Linux libraries are available. The missing Linux libraries can either be because of OpenHub grouping all these smaller Libraries as part of the Linux Kernel and simply a product of them. Otherwise the projects are missing entirely, and are missing because nobody have created them with OpenHub and the information is thus unavailable. Many of the dependencies of the OSS projects will then not be available to OpenHub and information on these projects will thus be absent for a trustworthiness metric. In general the search mechanism at OpenHub does not seem to be very successful in their search, and many queries will give no result or complete different projects entirely. Searching for a specific name is mostly correct, but searching with the name being a little misspelled will not even find the project and mostly not give any results for a query. This is of course not a great feature, but with most searches being quite precise, the search problems at OpenHub will be limited.

3.4 Software

The software product will need to developed with similarities to a well known process in software development and in a language well known in the academic world. The goal of the software is to be the start of a trustworthiness score development, which will not end with this project. The software product should thus be easily available and expandable for other academics working with security and trustworthiness in software.

3.4.1 Development process

The software process to be used for the software development will be similar to Agile development, although Agile development would not quite make sense for a single person project. The meetings are thus lonesome, but the stages in the process are very useful for software development in small to medium sized projects. The agile development process is a repeated iteration of 4 processes, which are found below.

1. Analysis

2. Design
3. Implementation
4. Testing

The 4 stages are the default in most software development, but for Agile development the 4 stages are repeated for each task or assignment in the project. Each large component is split into smaller tasks, which have to be accomplished in order for the component to work as intended. Each task does then use the 4 stages to *analyse, design, implement* and *test* a solution to the task, which is then added to the other tasks, and in the end a fully functioning component is then implemented. The tasks can at times need to be done sequentially for the process to work. Teams working with Agile development have a planned cycle called sprints, which can vary but are often a week and at the end of the week the entire team is meeting to be updated with progress on the individual tasks. The developers will discuss the relation between the components and a new task will be assigned, and the next sprint has started. The strength of the agile process is the tasks and smaller components will be fully functioning and can be tested, while other parts of the project is still under development. Unfortunately, projects in the industry can often end up being delayed. The unfinished project will have the most important components finished and functioning, if the project is well managed, even with the project being delayed. Agile development is great for a certain size, but with huge projects an Agile process can be hard to maintain, when trying to find all the correct requirements. Huge projects can be national projects similar to Deutsche Bahn's software infrastructure or military projects.

The Agile development is great for small projects, which ensures all the implementations are tested and working correctly. The testing is important, since software developers often think that all possibilities are handled correctly, although they will often find a problem or a result being incorrect in some way. The testing will thus ensure that the software is working correctly and others can use these to check the implementation.

3.4.2 Programming language

The programming language is not always the simplest decision and is often made for the reason of usage and preferences by the team. The common programming languages are then probably going to stay as the most common, until a significant different programming language gains ground. For example is the programming language C often used either for performance or hardware integration, while Java is used for the Virtual Machine to create a more versatile and cross-platform product. Perl is well known for

the ease in data mining, but has not gained much popularity with developers otherwise. Perl is not the most used in either the industry or academics, which tend to use other more popular choices.

A good choice for the programming language would be Python, which is popular both for academics and developers in aspects of data mining, data manipulation and machine learning. Python is a high level programming language and good for many aspects of computer science with a large amount of developers developing libraries for many utilities. Python would be a fine choice with many developers, and the academics in Computer Security would be able to use the code and implement new metrics for trustworthiness. Python currently exists in two different versions 2 and 3, where 2 has more libraries integrated as not all are continued into version 3. Version 3 is thus the version, continued into the future with new libraries being developed. Python 3 is thus the obvious choice for a programming language of a trustworthiness system.

Other alternatives do exists like Ruby, Java or C#, which are all fine programming languages with useful features for a trustworthiness system. Ruby unfortunately does not have all the libraries Python has, and machine learning could be useful in trustworthiness down the line. Java and C# would be fine solutions as well, but Python is stronger and libraries in data mining and machine learning are kept up to date and well functioning. Java does have a large overhead, when run and can sometimes operate slowly compared to other programming languages, and with a large set of data, Java might not be the best solution after all. C# does not have the libraries supported quite to the same level as the available Python libraries.

A useful library for finding CVE information called cve-search is similarly a Python3 script, and Python3 would be needed in the project all together. A user of the software will thus not need to install additional programming languages except the main programming language for the entire project.

3.4.3 Structure

The software will have to be structured quite differently from the product of García with components handling certain tasks and gathering the information for calculating the scores from section 3.2. Most components will have the task to find information or use a certain library, which are all used to score the project on trustworthiness. These components will be developed similar to interface, but as Python does not use interface it will just be scripts or classes calling each other, where the functions will return the value for the original class. The components will work as interfaces between the components,

where functions will call each other and utilise other functions to give the intended response.

The components will work independently from each other and have individual tests to ensure all the components work correctly. The tests will be UnitTests to test the functions for a correct and valid response, which in general is called white box testing. Tests are also used during the development of the system to ensure the functions response correctly given all possible arguments and still respond with a valid response. The Null response in case of a bad request or if no results were found by the request. Using tests to guarantee the functionality is called test driven development, where tests often can be created before the actual implementation for the developer to fulfill during the development. Test driven development is very popular and a good developing method, as the developer is able to test all functionality in case an implementation interferes with a previously implemented part of the system. The UnitTests can be used for external users to validate that the overall system works as intended, and users will be able to find the system which are not correctly setup on their system.

3.4.4 Libaries

To access the different data sources, a few libraries will have to be found and in the end utilized. The Vulnerability data is available at NVD, and a library called cve-search is developed to search the nvd register and find information on projects and their CVEs. The cve-search library does take time to configure, but the configuration is simple. It is simple to use and is able to make all the necessary information available for the security metric. to find the dependencies of the project a Linux library called apt-rdepends is able to find the dependencies of the different projects. To access the data from OpenHub a web scraper will be needed in order to easily iterate over the HTML pages using the tags to find the information needed.

3.4.4.1 cve-search

Cve-search[22] is a python programmed library, which utilises a MongoDB database to store all the information on CVEs. Cve-search is an OSS project helping people easily access and look up information on projects or the individual CVEs with simple command line interface. The library uses Python3 to run, and the source code is not meant to be run as classes or interfaces, but is just scripts for the user to find information on vulnerabilities in the terminal. The scripts simply writes out all the information found or is able to visualize specific information, if you use additional time to configure the

project correctly. As the project is written in Python with a MongoDB database, the library is able to run cross-platform and is not restricted to a single OS.

The configuration of the library is explained for most parts on GitHub with all the source code. The configuration can take a few hours, since all the information from the NVD on vulnerabilties will have to be populated into the database. An issue with a few calls was noticed, when switching operating system from Linux to MacOS, the output format was chanced and instead of using JSON a simple printout was made. The difference can thus be more greater, than would be expected and using a single OS might ease the work with software utilising the library.

3.4.4.2 Web scraper

Web scrapers are created for mark-up languages to be easier to index compared to the page being a single string. Web scrapers parse the HTML to find the information needed on the web page. Available Web Scrapers for Python3 are limited with only a few well functioning libraries. The most used one is BeatifulSoup, which parses the web page and creates an index of the web page. BeautifulSoup needs to be given the specific HTML code, and a library called RoboBrowser is developed on top of BeautifulSoup in order to surf the web more easily and create requests for the mark-up code. The parsing of web pages is not a fast process and can take a little time with many pages to browse over. The web scrapers are useful to browse the web, but a limitation is the scraping only works as long as the HTML is not changed. Updating a web page's HTML and completely restructuring, is rarely done except with a complete redesign. The web scraping function will thus need to change in this case.

Alternatives to BeautifulSoup is quite limited in Python3, while WWW:Mechanize is alternative if used in Python2.

3.4.4.3 apt-rdepends

Apt-rdepends is one of the few command line tools available to find the dependencies of a project. Dependencies are only relevant when discussing OSS, since the installation of Non-Open Source products will install and control everything and not give information about what is used to accomplish the tasks. Apt-rdepends uses the dkpg description to find the relation to other projects and returns the result as a string. The apt-rdepends is simple as all the information is given by the dpkg. Unfortunately alternatives to apt-rdepends does not quite exist currently, as this is only interesting for OSS and most of

this software is installed by executable script, which means no formal register is made for the dependencies of OSS projects.

A limitation is that apt-rdepends only works with Linux distributions and thus will only be useful for giving dependencies of a project with the Linux libraries. The dependencies in other Operating Systems could probably be found with package managers in the OSs, but Windows and Macintosh do not use a package manager at default. Windows have executable files, but Macintosh have available package managers in order to install different software libraries. The apt-rdepends does what it is designed for, but the limitation of the operating system should be apparent to the user of the library.

3.4.4.4 Restrictions

The restriction of the Operating System is significant as the dependencies will only be available for Linux OSS projects. The Linux community is the most OSS oriented, and most of the software available in Linux will be one of the types OSS projects. Linux will thus be a good choice for the operating system with everything being Open Source already. It would be nice with the product of this Thesis to work on other operating systems, but this restriction is something the users will have to live with.

Other than apt-rdepends the other libraries are available and can work in the other operating systems. The implementation might have to change a little, if trying the software product on another operating system, as differences can be present unexpectedly and each functionality will need to be tested.

3.5 Summary

The metrics for creating a trustworthiness score will be to consider Security, Maintainability and the Team contributions. The scores will give insight into the quality and security of the overall project, and thus the likelihood a user would trust the system. The Security is rated by the discovered vulnerabilities with information from CVE and CVSS data, and how the projects overall security is over the time the project have been active. The security is a major part in the overall trustworthiness and is the largest concern for users. The Security metric is called Aggregated Security score, which rates the severity of the vulnerabilities found over time, and the overall vulnerability of the system by looking at the set of vulnerabilities annually. The Maintainability will tell how well structured and easily understandable the source code is, which means the contributors will have an easier time fixing potential vulnerabilities. The Maintainability is

rated by the amount of comments written in the source code, since the more comments the easier the code is to understand. The last metric is the Team Contribution, which will look into the contributors projects and these projects' quality. The contributions by the developer into the various projects will enrich his experience and over time improve his skill to create better software. To attain a score of the contributor the metric uses the overall quality of the project, he has contributed to.

The software to be created will be based on components to fulfill specific roles in the overall metric depending on the data to gather or the library to implement. The process will be a process similar to Agile development with analysis, design, implementation and tests. These are key part of the software development, and these will be conducted with each task to be developed in the software. The test development is important for ensuring the correct functionality is intact at all times. These tests can be used to see for other developers, if everything is configured and still working in case of further development. The programming language to be used is Python version 3, which is excellent at data mining and data manipulation. These attributes are essential for finding the correct information and organising the information to be used in the trustworthiness metric. The libraries to use are cve-search with information from NVD, RoboBrowser for information on OpenHub and apt-rdepends to find information on what dependencies a project consists of. The libraries are a key components for the project to be able to find the information to rate the trustworthiness.

Chapter 4

Design

The design of the software is quite simple with different classes created to fulfill the tasks. The overall design is simple as the product is a small project. The design is created to be one single component called OSSProject, which is an Open Source Software Project, which will be the project evaluated. The different metrics will be implementing with the link to the information sources needed to conduct the evaluation of the individual metric.

Python has created software design differences compared to other programming languages, which should be known in order to understand the design and the source code. In Python a class does not necessary need to be a class, but can be a collection of scripts, which in other programming languages would have been designed as a class. Python is a high level programming languages and have made some changes compared to others to simplify the structure of software design.

4.1 Metrics

The metrics each describe an aspect of the overall trustworthiness, which is a combination of 3 scores *Aggregated Security score*, *Team score* and *Maintainability score*. How the aspects add up can be found in the diagram 4.1.

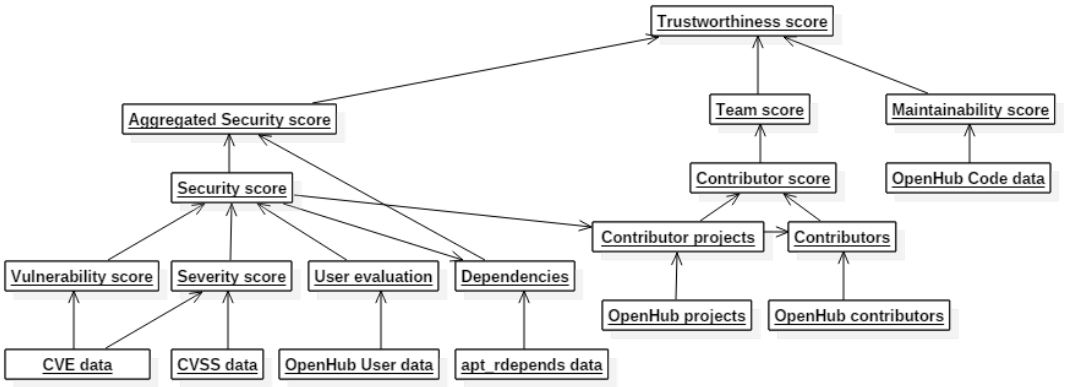


FIGURE 4.1: The relation between the metrics can be seen in the diagram, and how the metrics combined will describe the trustworthiness of the software. The different metrics will have different information sources in order to find the relevant information. For larger version see appendix C

The 3 metrics are each a combination of different information, which can be found as the nodes of the metric structural diagram. The Maintainability score is the simplest with only using data on the project from OpenHub, which is then easily calculated based on the formula 3.8. The Aggregated Security score is a little more complicated with information from all the information sources. The Security score uses the Vulnerability score and Severity score to calculate most projects using only the data regarding CVE vulnerabilities and CVSS score information. The User evaluation is used for smaller projects or projects with a small number of vulnerabilities, which is then evaluated on the reason for this small number with the information on users and contributors in OpenHub. The dependencies are found using apt-rdepends to find the projects dependencies, and these dependencies are equally evaluated based on the Security score. The Aggregated security score is then finding the most untrustworthy score from the project or its dependencies. The Aggregated Security score is thus an evaluation of the weakest link within the entire project.

The Team score is based on the information from Openhub regarding the project's contributor and their projects of contribution. The contributors will be grade on the projects, which they have contributed to. The projects are found from the contributor's projects on OpenHub, and the projects are scored by the Security score. The Contributor score is a combination of the contributor's contributions to the projects and from the Security score of the project. The contributor score is calculated from the formula 3.9. The Team score is then calculated from all these contributors to contribute to the overall trustworthiness valuation.

4.2 Interactions between components

The different sequence diagrams shows how the scores are calculated for the final trustworthiness score. The sequence diagrams are split into the different scenarios of the scores, which are done sequentially by each other and will be presented in this fashion. The different scores are calculated as the function is called in the OSSProject class, and it is possible to simply call for the dependency, maintainability or team score individually without all the other scores being calculated.

4.2.1 Security score

The Security score consists of 2 different scenarios to algorithm, which depends on the annual numbers of vulnerabilities in the project. The first is for an annual average above 5 CVEs, which is found in figure 4.2. The second is for the CVEs less than 5 annually, which will result in a user evaluation found in figure 4.3. The regular Security score is based on the Vulnerability score and Severity score. The user evaluation assigns a score based on the amount of users and contributors with a score of either 0 or 10 depending on the project's situation.

The regular Security score is for annual CVE count above 5, which will then proceed as followed in figure 4.2.

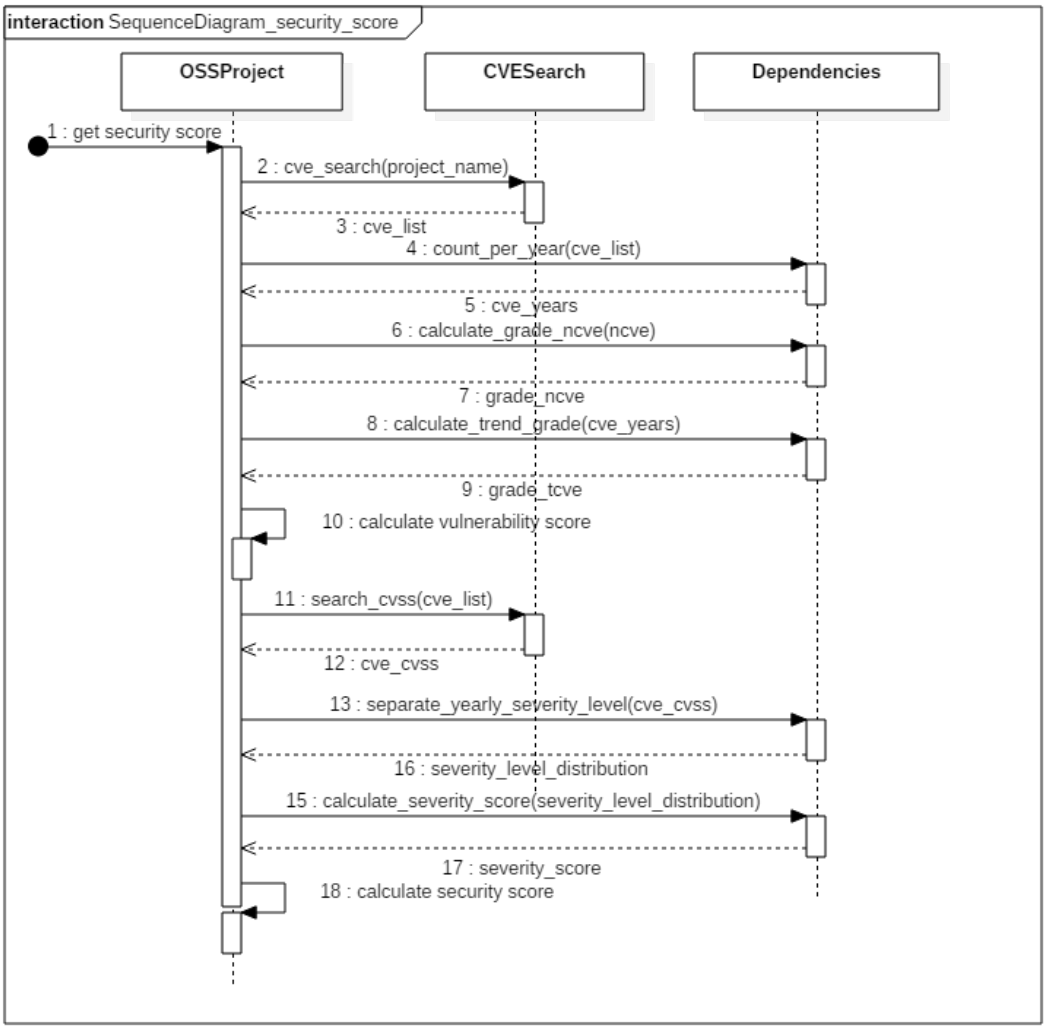


FIGURE 4.2: The sequence diagram for calculation of the Aggregated Security score with the calculation of the vulnerability score in steps 2-10 and the severity score in steps 11-17.

The beginning of the process will evaluate the information from CVEs by looking at the Vulnerability score, which focuses on the general trend and numbers CVEs, and the vulnerability score is conducted in the steps 2-10 in the sequence diagram. In step 2 the CVESearch class finds the CVE ids as a list, which only contains the information about the CVEs' id without any related information. The CVEs are separated into a Map with all the count of CVE vulnerabilities annually in step 4, which is used both in calculating the average, and the overall trend of all the vulnerabilities. The grade for the annual average (ncve) and the trend (tcve) are then found by the Dependencies class in step 6 and 8, which returns 2 individual grades. The 2 grades from ncve and tcve is then combined and calculated to the vulnerability score in step 10.

The severity score is calculated by the Dependencies class for most parts, but organised and controlled by the OSSProject. The information needed for the severity score would be the CVSS scores of the previously found CVE ids for the vulnerability score. These scores are found in step 11 and 12 by the CVESearch class, which returns a Map with the CVEs and their individual CVSS score. The CVEs have to be separated by years and severity level to calculate the severity score, which is done in step 13 by the Dependencies class. The return in step 14 will be a Map with the year as key and another map for the different severity levels and the count. The Severity score is then calculated by the Dependencies class based on the data delivered by the same class. The interaction back and forth is caused by the Dependencies class only containing scripts. The idea is for the OSSProject to control and contain all the data, which is used for the calculation and can easily be changed if others will change the control flow of the program at a later point.

As seen in the class diagram figure 5.1, the Dependencies was created as a script instead of a class, since the information to return at any point is simple. The data is always homogeneous with being a list of scores and ids. The data can thus easily be contained within the OSSProject without being confusing and unorganised. On the other hand the function interactions can be significantly decreased in case of Dependencies being a class, but since this is not a problem the Dependencies will work just as well for a scripts instead of a class.

The sequence diagram for the user evaluation is when the annual average is less than 5, which means that the overall vulnerabilities found in the software product are low. The low vulnerability discovery can be caused by little activity from other developers as users in the project, or that the product is of a high quality level of software. The user evaluation is decided from equation 3.4. The control flow of the software is similar to the Security score control flow in figure 4.2, but the difference is that before the Vulnerability score calculations is initiated, the user evaluation will evaluate the Security score.

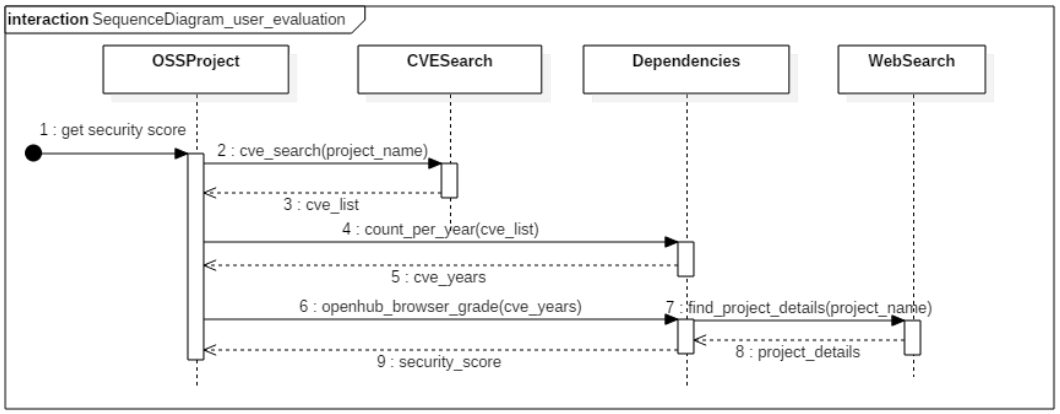


FIGURE 4.3: The User evaluation sequence diagram is simpler than the Security score sequence diagram with the larger part not being evaluated for the User evaluation. The User evaluation is used for the CVE annual average being less than 5, and will thus be calculated based on the user and contributor numbers of the project.

The sequence diagram has the same initial steps with getting CVEs and creating an annual count for the project's CVEs. The difference is the ncve is less than 5 and thus is the user and contributor information found in step 6. The Dependencies class find the information on the project's user and contributors by calling a function in WebSearch, which the project is then evaluated on. The Security score is thus assigned based on the user evaluation formula 3.4 by the Dependencies and returned to the main class.

4.2.2 Maintainability score

The maintainability score is simpler to calculate and currently the simplest metric, since it is just based on the information available from OpenHub's data. The sequence diagram shows that OSSProject only gets the source code information and calculates the score from this information.

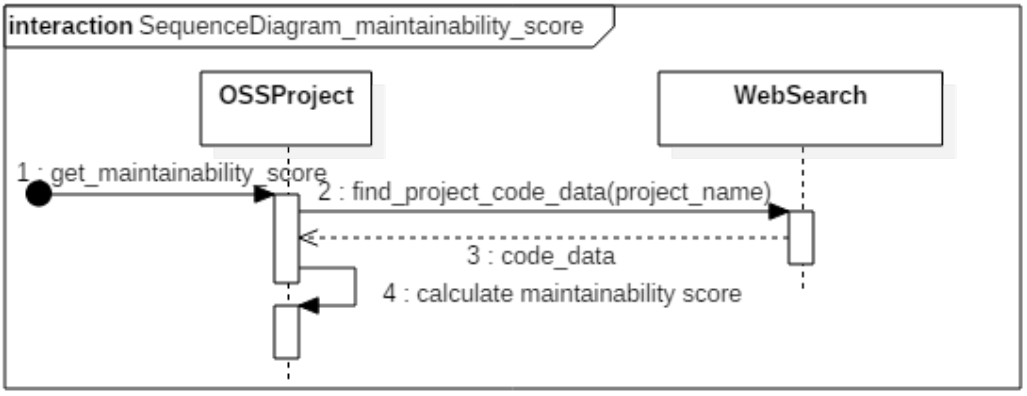


FIGURE 4.4: The Maintainability score sequence diagram is simple with WebSearch delivering the data on the source code to OSSProject, which is then in charge of evaluating the Maintainability score.

The code data is returned to the OSSProject class as a Map with all the relevant data and used to calculate to the Maintainability score. The simplicity of the score is a strength as the score is easy to calculate with OpenHub information from a single web page, and time is not used to find the information from the source code itself.

4.2.3 Team score

The Team score is a heavier in calculations, since a larger set of data needs to be obtained for the calculation. The sequence diagram is simple as only few functions are called for the team score to be evaluated, but the different functions will process smaller or larger data sets depending on the project to evaluate.

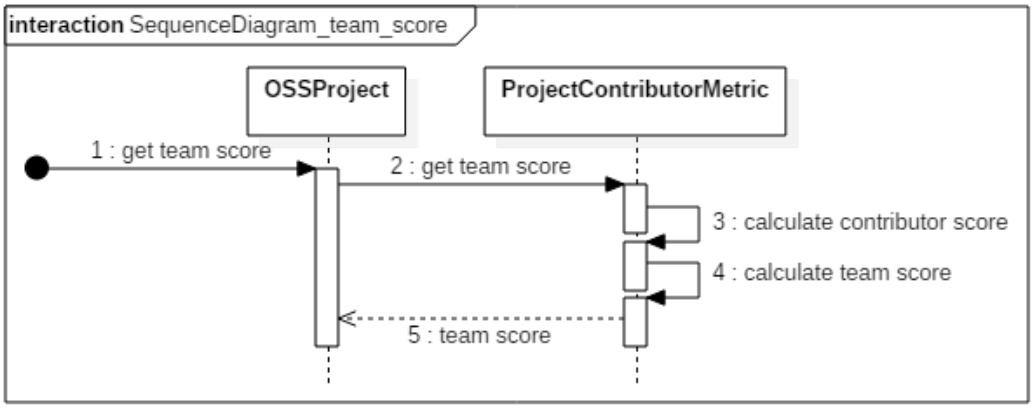


FIGURE 4.5: The Team score is mostly calculated and evaluated by the ProjectContributorMetric class, which calculates the contributor score based on the projects of the contributors. The Team score is calculated based on the coontributor scores and returned to the OSSProject.

The scores are evaluated by the formulas 3.9 and 3.10, which will be iterated over all the contributors' projects and the contributors. The evaluation of the team score is performed by the ProjectContributorMetric class, which contain a matrix for all the projects the contributor have contributed to. The step 3 will iterate over all the found contributors to find all the projects contributed to and evaluate these projects by the Security score, and these projects can be large set of data. The Team score is then calculated by iterating overall the contributors and scoring the teams' experience and skills from other projects.

The class ProjectContributorMetric is able to contain information in the class instead of just returning the data to the OSSProject, which is the case for Dependencies in the Security score. The Dependencies could have been made as a class instead of being a script, but this was decided as the information is significantly simpler and the OSSProject can easily contain the data without being unorganised. In case the Security score should be expanded to more information, a reevaluation should be made for the dependencies to become a class. The PCM class has to keep the data on contributor, projects and the link between them with contributions. The data is saved and organised in the class for the project to use at any time.

4.3 Summary

The design of the software is simple with few classes with each their responsibility in finding and evaluating the data on the Open Source Software project. The software

design consist of a single component using other third party applications to accomplish the trustworthiness score. The component is controlled by the class OSSProject, which then assigns tasks and controls the flow of the trustworthiness evaluation. The other classes in the class diagram assists in their way for OSSProjects to accomplish this evaluation of the project.

The control flows of the different metrics can be seen in the sequence diagrams, which are more or less complicated based on the task at hand. The control flow is controlled by OSSProject to distribute the tasks to the other class and scripts. The class is capable of accomplishing more complicated tasks and containing the data, and the scripts simply returns the data from a given call without storing any data. The team score evaluated by the ProjectContributorMetric is containing a large data set in order to do the calculations, and all the data is easier contained in the class instead of returning all the data back to OSSProject as Dependencies does, but this is the difference between classes and scripts. The Maintainability score is easily calculated by the few function calls in figure 4.4. The Aggregated Security score is not so complicated to calculate, and the data set is sent from Dependencies to OSSProject to evaluate the overall security of the project and its dependencies.

Chapter 5

Implementation

In this section of the report the implementation will be described for the structure, connectivity with other components and the individual classes. The structure will be described based on the implementation of the third party libraries and classes containing large amount of data. The classes are described by the individual functions and how they are implemented.

5.1 Structure

The project consists of two classes and 4 helping scripts, which are all part of the OSSProject component, where the scripts will find information by implementing third party components. The two classes OSSProject and ProjectContributorMetric are two objects, which interacts with each other to perform the given task, while the 4 scripts or function libraries helps to perform certain tasks and provide the data to the 2 objects. The class diagram can be found in a larger version in appendix B.

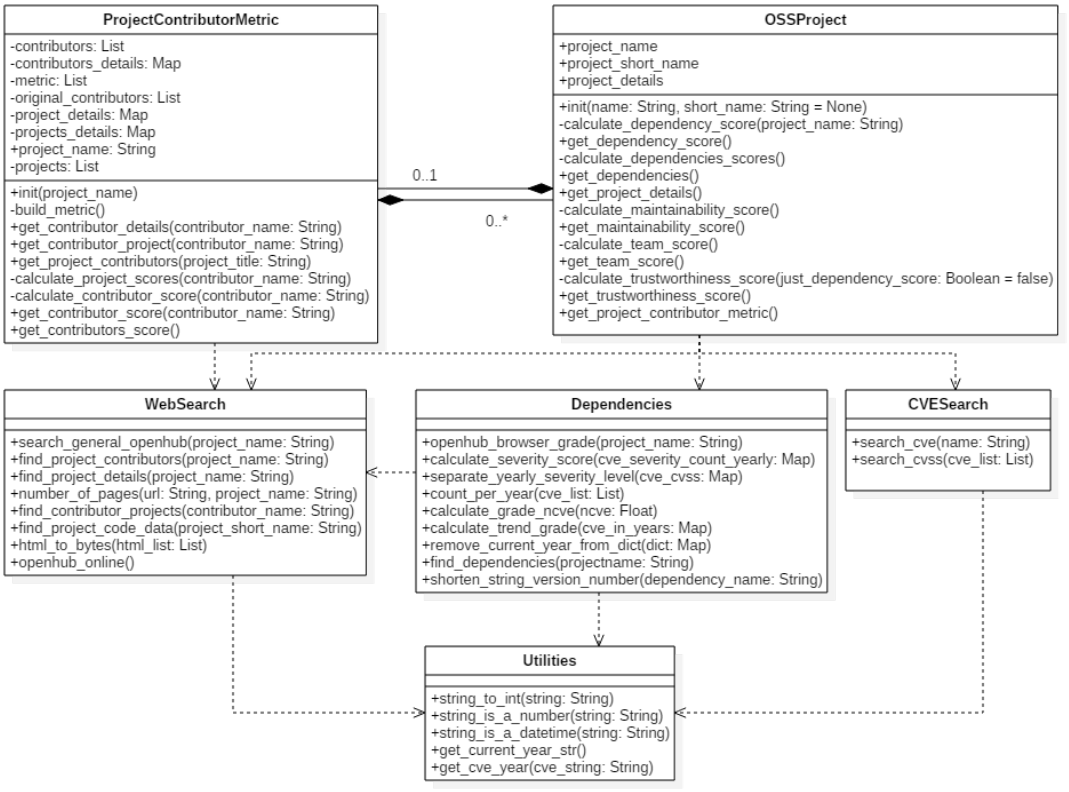


FIGURE 5.1: The OSSProject component consists of the classes in this diagram(larger version in Appendix B). The main class of the component is the OSSProject class, which is in charge of all the functionality and outsource the tasks to create the trustworthiness metric. WebSearch searches web pages and in currently only OpenHub.net, dependencies finds dependencies of the OSSProject, CVESearch finds the CVE and CVSS information, Utilities contains helping functions and ProjectContributorMetric is a Matrix containing all the information about contributor and their contribution to different projects.

The OSSProject class is the main class of the project, which assigns tasks for the other classes to handle. The OSSProject uses the classes WebSearch, Dependency and CVESearch to find information to calculate the dependency and maintainability score of the project. OSSProject has a composite relation to ProjectContributorMetric (PCM), since OSSProject will contain a PCM object to find and organise all the information regarding project contributors. The PCM object is not an association, since the PCM is part of the project and would not be relevant on its own. The PCM object will find all the contributors in a project and all the projects, which the contributors have contributed to, to evaluate the team's experience . The PCM object contains a large amount of information on the contributors, which are available through the OSSProject. The PCM equally has a composite relation to OSSProject as every project evaluated on the security metric and assigned a Security score, will create an OSSProject object in order to make this evaluation of all the OSSProject contributed to by the contributors to the original project. The Security score is calculated based on the original project and not

using the dependencies of the project, since this would make the calculation significantly heavier.

The classes WebSearch, Dependencies and CVESearch would have been classes in other programming languages, but objects can be created in Python without being a class and be a script instead. The classes are in fact more like scripts, which will run and simply return a result based on the function. The Dependencies script will consists of script manipulating the data on the dependencies and assist the OSSProject to organise the CVE and CVSS data and calculate the Security score. Dependencies use the WebSearch class to find the amount of users and contributors for the User evaluation. CVESearch is simple and have 2 functions to find CVEs and their CVSS scores for each CVE. The WebSeach uses the RoboBrowser component to find data on OpenHub, which have been used for the different metrics to be calculated. The last scripting class is Utilities, which contain functions used by the other scripts to handle data.

5.2 Components

The components collaborating to score a project based on trustworthiness, are OSSProject, apr-rdepends, RoboBrowser and cve-search, where some components are third party application used to find relevant data. The main component of the project is the OSSProject, which takes charge of all the information and creates the rating based on the information provided by the external sources.

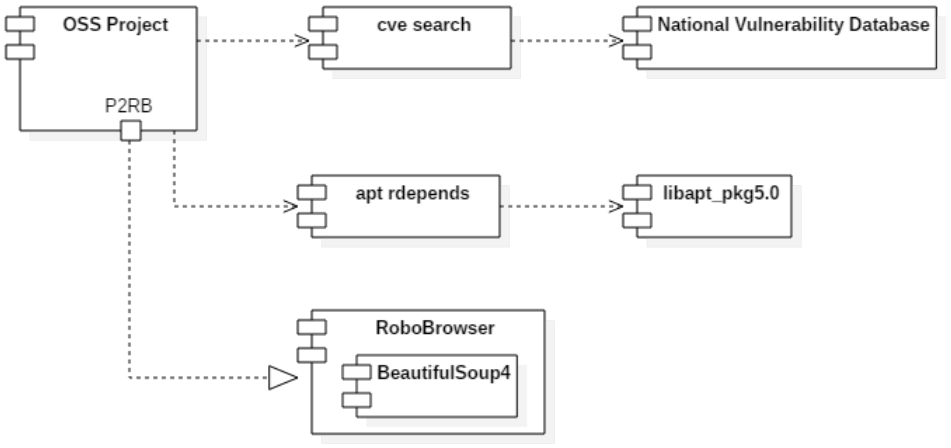


FIGURE 5.2: The Component diagram shows the projects connectivity to third party software libraries. The different libraries are used for finding information on the OSSProject to score the project on trustworthiness. The libraries are used and realised by different means to use the libraries as intended.

The all components are connected to the main component OSSProject with different relations to accomplish the tasks of the components. The components cve-search and apt-rdepends are used by a command line library called subprocess, since the components are intended to be used from the command line and not realised as a Python object. The apt-rdepends is created to be a command line tool in Linux and not much information is available on this component, but the component is written in Perl and depends on the library libapt-pkg5.0. Libapt-pkg5.0 is used as a library for high level package managers to manage software packages. Cve-search is written in Python and is chosen to be a command line tool as well, although a Python class would have been nice to include and easier to manage when using Python code. The idea was instead for everybody to use it as a tool to find relevant information about CVEs. The library uses information from National Vulnerability Database to find and save the information in a MongoDB database.

The RoboBrowser is a library used in Python to scrape websites for data, and RoboBrowser is using another Library called BeautifulSoup in order to parse the HTML into a beautiful soup of information. The BeautifulSoup Library is able to create an overview of HTML in an organised fashion, which RoboBrowser uses along with requesting the HTML from a website. RoboBrowser is used to request and find information from the website OpenHub and in a few special cases BeautifulSoup is used on its own, although RoboBrowser might have been a possible implementation but would require more work. The RoboBrowser is realised as an object in Python to search through the website and look for the relevant data to the project. The relevant data in this project is data on specific OSS Projects about the source code or the contributors.

The component diagram is simple and the project only uses a few libraries to accomplish the task at hand to evaluate a project on trustworthiness.

5.3 Classes

The classes can be found in the class diagram figure 5.1, and the implementation considerations will be described in the following section. The main class in the project is OSSProject, since it is the class object, which can be used in other projects where a trustworthiness score is necessary. The OSSProject will delegate the tasks in between the classes to find all the necessary information to conduct the trustworthiness score for a project.

5.3.1 OSSProject

The OSSProject is the main class for the entire project and have a few public functions for the data to be returned. The OSSProject does not calculate anything that has not been asked of the project by a get method, and the calculation is performed as the request is made. The get methods can be seen in the diagram below in figure 5.3.

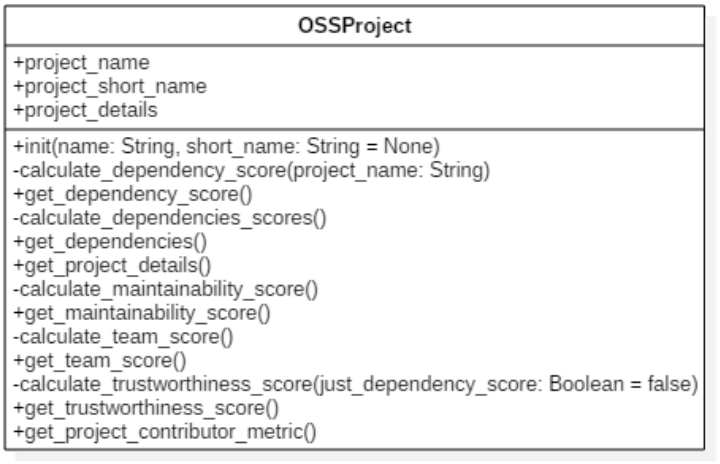


FIGURE 5.3: The OSSProject class is the main class controlling the actions taken and tasks performed. The class will calculate the different scores from the metrics based on the data and scores received from the other classes.

The get methods are either a score or a set of data, which the calculation was based upon. The scores are calculated as the get method is called and only the necessary score is calculated for the method, which means all the other irrelevant scores or data will not be calculated before the data is needed. The data is thus not available, if the necessary score using this data have been run. The data is the data available at the moment the function is called and can be Null (None in Python) in case the data is unavailable. The get trustworthiness method will thus calculate all the scores for the overall trustworthiness to be evaluated.

The methods for calculating Maintainability score, Trustworthiness score and most of Security score is implemented in the OSSProject class, while the calculation of the team score is calculated by the ProjectContributorMetric. The Security score for the project is calculated by the delivered data from Dependencies, and the control flow can be found in the sequence diagram 4.2. The overall Security score with all the dependencies will be described after the initiate dependency metric is explained. The only calculations done for OSSProject on the Security score is the simple calculations with all the data received, while more complicated calculations are assigned to other classes. The data used to calculate the vulnerability score is a list of CVE ids and a list with the count

of ids into the year the vulnerability was discovered, in order to calculate the variables in formula 3.2. The calculations for the annual average and the trend is performed by the Numpy library, which is a mathematics library. The trend is calculated in the Dependencies class, and the annual average is calculated by Numpy from the Map with counts per year using the mean function. The Severity score of the Security score is mostly calculated by the class Dependencies, since the Severity score is a collection of different trends and other data from the CVSS scores and the severity level. The Severity score will thus be discussed in section 5.3.2, as it is calculated by the Dependencies class. The overall Dependencies score is then calculated by the data from Vulnerability score and Severity score.

The general Dependencies score for the project uses the calculation of the project itself and all the dependencies used in the project, which are using the same metric for the original project to calculate the Security score. The dependencies of a project is all the libraries used both for the project but equally for all the dependencies projects, and the complete numbers of projects are all the projects used for the package manager to have the software installed and configured. These dependencies are found by calling a function in the Dependencies class. The dependencies are all having their Security score calculated, and the highest security score is then used as the overall score for the project. The highest Security score or the lowest trustworthiness score is used for the project as this is the weakest link in the project and thus easiest for a threat source to attack to gain access to the system. The apt-rdepends finds all these dependencies in Linux, and the names can vary from the projects in the CVE database. The names project can exists in the CVE register but the name might not be the same as the one used the apt-rdepends for Linux package managers. The project will thus not with guarantee be found.

The Maintainability score is simply calculated with the data provided by OpenHub with amounts of lines of code and comments for the calculation. The data is accessed by the WebSearch class, which will provide the data on the project. The score is determined by the formula 3.8. The calculations are thus quite simple and the data collecting will be described in for the WebSearch class in section 5.3.3. The Trustworhthiness score is simply calculated by the formula 3.11, which is a simple calculation of the already calculated scores for Security score, Maintainability score and Team score. The scores are calculated and in case these are not calculated the calculation will be initiated before the Trustworthiness score is calculated.

5.3.2 Dependencies

The Dependencies class is created to assists the OSSProject in the calculation of Dependencies score. The class uses the library apt-rdepends to find the Linux dependencies for the project, and it is the projects link to this library. The Dependencies class handles the data for OSSProject, when the data is related to the dependencies and the security metric. The class' available functions and thus the data can be seen in the class diagram in figure 5.4.

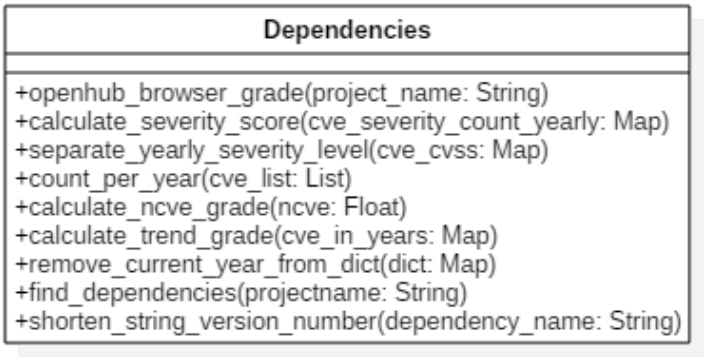


FIGURE 5.4: The Dependency class handles the data about CVE and CVSS scores to calculate the Security score with all the information about dependencies of the project. The Dependencies class does handle all the metric related to the security.

The dependencies functions is assisting the main Dependency calculations for achieving the Security score of the project. The class cooperates interact with OSSProject quite a bit for the OSSProject class to create the evaluation of the Security score. The functions are used for manipulating CVE data, finding the dependencies, grading an aspect of the Security score or utilities for CVE data and dependency data.

The functions for manipulating data on vulnerabilities are *count per year* and *separate yearly severity level*, which are used in the Security score of OSSProject. The count per year function is used to separate a list of CVEs into a list with the counts of vulnerabilities each year, which is used for the evaluation of the Vulnerability score to be calculated on tcve and ncve. The result is a Map with the years of found vulnerabilities. The separate yearly severity level is the function used in the Severity score for separating the CVEs first into years and then into their level of severity, which will be a resulting Map with the years and a Map with the levels of severity.

The utilities functions for CVE data are *remove current year from dict* and *shorten string version number*, which are only used inside this class and thus not moved to the Utilities class. The *remove current year from dict* is simple and just removes the current year

from the Map or Dictionary object, which is used both in the count per year function and calculate severity score. The removal of the current year is done, since the current year does not contain all the vulnerabilities as the year is not concluded yet. The year will thus only contain the discovered and will depending on the date contain a fraction of the year and this can cause a trend or average to be lowered and might even cause a stable trend to be decreasing instead. The *shorten string version number* was created to shorten the version number of the dependencies to find a project it might be related to. The naming of the Linux libraries at times contain several version numbers and these were removed to find the project it might belong to, but this function is not currently used as removing the numbers of a project might result in the project being a completely different project and not related to the original dependency project.

For the Security score the function *find dependencies* is used to find the dependencies in Linux by the apt-rdepends. The apt-rdepends is a library installed in Linux and simple need a name to find the dependencies, and the dependencies are for the project listed with the relation to the project by the possibilities presented in section 3.3.2. The different dependencies are then found by searching the result for Depends or Pre-Depends to find the relevant libraries, and the other relation types in the package manager is not relevant as the dependency is not necessary.

The functions for calculating a grade for the Vulnerability score is *calculate trend grade* and *calculate ncve grade*. The *trend grade* will be using a library called Numpy, which is a general mathematics library for Python. The trend is calculated by using the Numpy library to calculate the Linear Regression of the annual count of CVEs and use the slope to decide, how the development of vulnerabilities are for the project. The slope is then given a grade based on the formula 3.5 to determine the trend grade for the overall project and for the individual severity levels in the severity calculations. The *ncve grade* is calculated from the annually average and given a grade based on the formula 3.3. The *openhub browser grade* function will use information from the project to do the user evaluation from formula 3.4. The information is found by the WebSearch class on users and contributors, and the grade is then assigned based on the situation with users of the project.

The *calculate severity score* is used to calculate the severity score, when the CVEs are separated into severity level and year. The severity score is then calculated based on the formula 3.6. The calculations firstly ensures that enough data is presented to conduct this score, since a trend with only one year will not be able to present a trend. For the projects with a single year of development will thus be evaluated as insecure and untrustworthy, since the users and contributors does not have had enough time to find all the possible vulnerabilities within the project. Next the project is examined to ensure that if a year

after the first evaluated year is not present in the Map a Map with 0 vulnerabilities will be stored for this year, which is especially important for smaller project. The smaller projects with a year without vulnerabilities would otherwise get a false calculation and a wrong trend for their vulnerabilities, if only the years with discovered vulnerabilities are presented. The next part will create lists for the trends to be calculated and eventually the severity score.

5.3.3 WebSearch

The WebSearch class is the implementation of the web scrapers libraries to find the relevant information on websites and in this project OpenHub is the only website used to find the relevant information. The libraries used are RoboBrowser to the extend it can be used and BeautifulSoup for the rest of the website to be accessible. The functions in the class is presented in the figure 5.5.

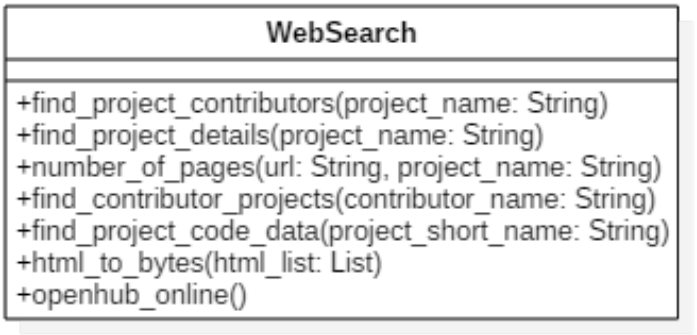


FIGURE 5.5: The WebSearch class is implemented for scraping websites to provide data from OpenHub for evaluating different scores. The WebSearch finds information about the projects source code and contributors.

The website OpenHub is down for maintainance about 2-4 times a month, which has been an annoyance in this project, and thus a function was made to check if the website was down or functioning. A few utilities are made for the WebSearch class to assist the other functions. These functions are *openhub online*, *html to bytes* and *number of pages*. The *OpenHub online* function checks the website is online, but unfortunately RoboBrowser is very unstable and can vary from a few seconds up to 30 seconds, which means the *OpenHub online* is not really as useful as could have been, but the user of the software will have to make sure the website is not down for maintenance. The *html to bytes* creates a byte array of the html string, since BeautifulSoup needs the information as a byte array or can better handle the information like this. The function is thus used for RoboBrowser to open a website and selecting a div tag and all its content, which will

then be changed from a list of html to byte array. The *number of pages* is used for finding contributors, since the contributor is presented as a set of pages with the contributors. The numbers of pages are found from the page of contributors to iterate through all the contributors. The number of pages is found by the links for selecting the pagenation on the page. The pagenation is simpler found on its own compared to finding in extention to all the contributors and was thus made into a separate function.

RoboBrowser is used by creating an object, which can then open a url. The website is then parsed by an HTML parser and uses BeautifulSoup to iterate through the website, which can then be used to find different tags and iterate through these. The BeautifulSoup has a select method, which finds the content of a div element for example, but unfortunately this content is not searchable, but another BeautifulSoup will have to be created to search the HTML content. The BeautifulSoup part of the RoboBrowser can equally find all the elements of a specific tag like links or div with a specific class or id. The contributors or contributors' projects are thus found by iterating the div elements in which they are contained. Much information on OpenHub can be found by iterating through links, since the links contain information on contributors or projects and linking to more detailed content. RoboBrowser and BeautifulSoup are quite powerful libraries, but the tools use time to parse the HTML and much time is used by these libraries.

The rest of the functions are used to find relevant information on the project, which is used by the other classes to calculate their metric. The information is found by parsing the project or contributor specific pages for the wished information using the BeautifulSoup library. The *project details* are found from a search for projects and using the first project, which is the best match for the search. The information for the project details are available right on the searching page for the number of users, contributors and lines of code, which is used as the basic information for a project. The *project details* are also used to find the project with the best match to a search. Unfortunately the searching can be quite insufficient and unprecise, where a search missing an letter in the end might find another project as a better match, and many smaller projects are missing and thus many dependencies will receive a score of 0 as they are not found in OpenHub. An example is searching for 'firefo' on OpenHub will give no results, but it is quite close to a project with the name 'Firefox'. The searches will have to be spot on to find the projects and the results can be entirely different, which is why the naming on OpenHub are not used to determine the naming for the project.

The names used for the projects are found by using OpenHub the project name, which is the full name, and the short name being the unique url project id used by openhub, which are often the shortest and simplest explanation to the project. The short name are often the one or a close name in apt-rdepends and equally used in the CVE register.

These names have thus been used although the names are not always correct for all these instances, but this was the best names used for searching all the sources. The name is important, but focus was used other places and the naming would have to suffice, since the names were correct for most larger projects but can have projects where they are not working. Especially projects where more elaborate names will have to be used to describe the project like several of MySQL projects, where apt-rdepends would need mysql_server or mysql_client for the specific software product. The names does suffice for most part and works fine with most projects.

The *project contributors* are found by examining the contributors page for all the contributors and using the *number of pages* function. The html is simple to search through as the website uses div tags to separate the contributors and finding the information is easy with the web scraping libraries. The contributors used in the projects are all the contributors found as unclaimed committers, since these are easier searchable compared to the accounts. The unclamined committers are most of OpenHub with less than 3% are accounts, and this is the used implementation and with all the searches done only very generic names will result in actual accountholders. The unclaimed committers are thus found to be enough for the implementation.

The *project of the contributors* are found by searching for each contributor and finding the exact match and on the search page all the project the contributor have contributed to are available and easy to search for. The projects are found with the commits to every project by the committer, which is returned to the requesting class. The *project code data* is found on a page of the project, which contains the lines of comments and code and more information on the language distribution in the project. The information is simply found on the page using the web scraper libraries.

5.3.4 ProjectContributorMetric

The ProjectContributorMetric is a class used to keep track of contributors and their projects. The general structure of the class is a matrix with contributors and their amount of commits to different projects. The matrix is created similar to, when data mining is looking into works used from different sources. The rows of the matrix is thus a list of all the contributors and the columns are all the projects. The matrix is thus filled with numbers, which represent the amount of commits a specific contributor have contributed to a project. The matrix will display a large amount of 0s, since most projects are only contributed to by a few of the contributors. A Map of the contributors and projects are kept to make the matrix searchable for a contributor and what index in the matrix is what contributor or project. The metric is *build* by requesting WebSearch

for the data about contributors and their projects to set up the matrix to contain all this data. In the implementation a limit of commits can be set for contributors and projects by the contributor, which will only use project with of more than the limit or only use contributor with more commits than the limit. These limits can be used to only use the most significant contributors or projects, since a large set of contributors will result in an enormous set of projects, and these data sets can be significantly smaller by setting these limits.

The ProjectContributorMetric class variables and functions can be seen the class diagram in figure 5.6.

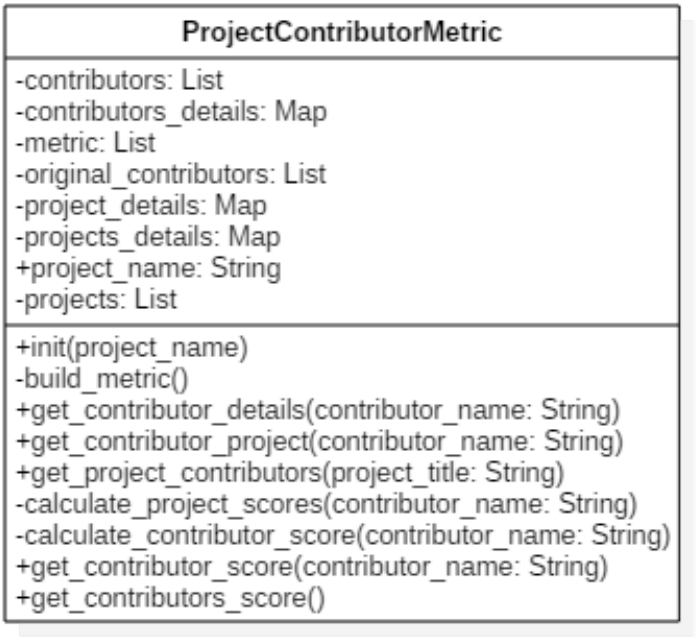


FIGURE 5.6: Caption

Calculating the contributor score will require finding scores of all the projects to find the contributor score from formula 3.9. Is the contributor score of a single contributor relevant or is all the contributors needed, calling the *calculate contributor score* can be either and will only rate the relevant projects in case it is only one contributor of interest. As an example with 220 contributors in the MySQL project have a set of 9592 projects to be rated and with about a project rated each couple of seconds will take about 5 hours to rate the project and all the contributors score. A small limit can be set for contributors and projects, and the limit reduces the run time significantly by reducing the data set. The Team score is quickly calculated if all the projects have been scored and the contributors too, and the Team score is calculated by the formula 3.10.

A few get methods are available, either the contributor's project or all contributors in a project. These get methods are *get contributor projects*, *get contributor details* or *get project contributors*, which will find the relevant data. *get contributor projects* will return the projects with their data based on a single contributor, and the *get project contributor* will result in all the contributors contributed to a specific project within the original project.

5.3.5 CVESearch

The CVESearch class is the connection to the cve-search application, which from the NVD finds the vulnerabilities related to the project. The cve-search is a command line tool, which will simply print the result for a project. The cve-search is used for 2 information in this project the CVE ids and the CVSS scores, which are found using the functions in the class displayed in figure 5.7.

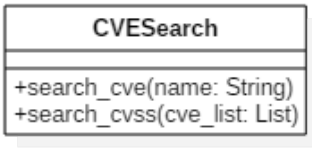


FIGURE 5.7: The CVESearch class is implementing the projects interaction with the cve-search project, which provides CVE and CVSS data from the NVD to the project.

The function search cve searches for the CVEs relevant to the project name and is printed an id per line. The string is then split by the new line character and inserted into a list. The command line used for the cve-search from the root of the folder for searching for the CVEs is:

./bin/search_fulltext.py -q PROJECT_NAME

For the fulltext search the MongoDB database need to be populated with all CVEs from the NVD, which is quite time consuming in the configuration. The NVD contains currently about 90,000 vulnerabilities, which will take approximately an hour to populate the database. The configuration can take a few hours because of this.

The list of CVE ids is then used as the argument for the next function search cvss, which will then find the CVSS scores for the individual CVE id. The search will return different information available in NVD with the information related to the CVE, which contains the same information about the cause, CVSS score and other information, but

the information used is the CVSS score for each CVE. The search is performed using the following syntax.

```
./bin/search.py -c CVE -c CVE2 -c CVE3 ...
```

The search can be done individually for each CVE or can be performed with all the CVEs in a single large command. Each call to the Subprocess library will take about a second, which means a large set of vulnerabilities will take a few minutes to perform. Using a single large command with all CVE ids is time saving although can give other problems. The time saved is significant as the return is still about a second for return of all the results, which is a significant time saved compared to projects with hundreds of vulnerabilities. The result with all CVE ids at once will return each object in a curly bracket in Linux, but with MacOS the data is presented with a line separating the 2 or more objects. The different results caused by the operating system is a problem, since different implementations would be needed for the operating systems, but since Linux was chosen to be the main operating system only a Linux implementation is made. The entities being separated by curly brackets is very similar and a few corrections to the string would form the structure of JSON, which was then used for iterating through the results. The function then returns a Map with the CVE id as key and the float CVSS score as the value of the Map.

The CVESearch implementation is thus only functioning for sure in Linux distributions as of now, since other part of the software is only working with Linux as well. The differences with the operating systems might can be fixed, when an alternative to apt-rdepends is available, but Linux is the working system for now with the project.

5.3.6 Utilities

The Utilities class is a class for assisting the other scripts in generic tasks. The tasks are simple and can all be seen in figure 5.8. Most of these functions are quite simple and not needing much of explanation, since these functions are just to simplify the general algorithm.

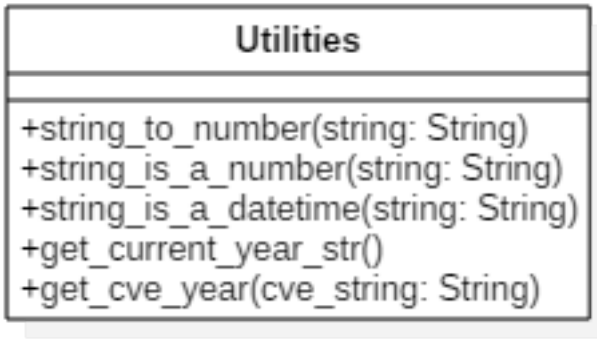


FIGURE 5.8: The class diagram for the Utilities class, which contains assisting functions for the scripts to use for finding specific information from their data. This can be the CVE ids year, help assert the kind of string object and parse to a correct int.

2 of these functions are simple a test to find out if the given string represents either a string or a date, which are the functions string is a number or string is a datetime. The function get current year is a function to return the current year as a string using a DateTime object to find the current year. Another function returns the year of the vulnerability discovery, since the CVE id contain the year of the vulnerability registered with MITRE.

The function calculating string to number is simple, but the function will equally translate a number with a string containing letters for signifying thousand, million and billion. Strings from the website OpenHub shows the lines of codes represented by for example 18M, which will be translated to 18,000,000, which this function does and also translates numbers separated by commas. The function will equally translate a string to a float if the string is essentially a float and not an integer. The function will try to translate anything that it might get from OpenHub of data as a int or float.

5.4 Summary

The different classes works from the OSSProject class that is calling all the other classes to find the information for calculating scores and evaluating a project on trustworthiness. The OSSProject is the object to be used for creating a project object, which can provide the information used to create the calculations. The OSSProject uses ProjectContributorMetric (PCM) to create a matrix for all the contributors and their contribution to different projects. The projects are all evaluated to create the Team score, which will indicate the experience and skill of the contributors contributing to the original project.

The implementation of the different libraries are created in scripts to easily provide the necessary data for the various scores.

The scripts implementations are designed to be as simple as possible and provide the data in an organised fashion for both the metric and an eventual user to get the overview of the data. The scripts simply returns the requested data in Maps for simplicity. The scripts are optimised as much as possible in order to improve the run time of the evaluation, but with much data needing to be gathered, and the time of the program was not a focus for the system. A few more classes such as Contributor might have been useful, but with the system working for all the different projects and handling all possible data, this corner was unfortunately cut and focuses on the overall score instead.

A few issues arise including search results on OpenHub and dealing with the different names used in different systems and libraries. The search result is nothing much to do about, but is weird the OpenHub project have not noticed the bad service. The naming used for OpenHub, apt-rdepends and CVE are not always the same, which can cause a problem with trying to give a single name for all the systems. A solution would be for the user to provide all the names or creating a dictionary with all the name on open source projects, but an automatic solution would be preferred. The automatic solution or a dictionary would be necessary, because the dependencies are found and can not be named by the user. An automated solution is required with all these dependencies and projects in the team score to have names for the different information sources. Unfortunately the issue would take up significant time to solve, and the use of OpenHubs name and unique id was deemed fine for now.

The implementations are made as simple as possible with all the metrics being handles by the classes with information from the sources. The implementation can take time to be calculated with much information needing to be sought out from the sources and be manipulated for the calculations in the end to be made.

Chapter 6

Evaluation

The evaluation will be performed in each individual metric in order to create an evaluation of each, and then the trustworthiness score will be evaluated as a collection of them all. The scores all contribute to the overall trustworthiness tool with a few metrics to use for calculating a score, but an expansion of the metrics is encouraged and possible with other's take on OSS project's trustworthiness. The evaluation will examine the results achieved by the various metrics in order to discuss eventual improvements in the score and in the metrics.

6.1 Aggregated Security Score

The Aggregated Security score will look into the security of the overall projects with all the dependencies used to achieve the product. The dependencies will be equally in importance to the general products, since these dependencies can as well be vulnerable to intrusion into the system. The Security score will be discussed separated and in relation to the dependencies.

6.1.1 Security score

The results of the security score is presented below with all the attributes to the metric represented in an abbreviated form[1]. The formulas can be found in section 3.2.1, if a specific calculation is of interest.

[1]The abbreviations are as follows ncve (average annually cve), tcve (trend cve), vs (Vulnerability score), lt (low trend), mt (medium trend), ht (high trend), ct (critical trend), ah (average high high criticality), ac (average high critial criticality), sev (Severity score), ss (Security score).

Project	ncve	tcve	vs	lt	mt	ht	ct	ah	ac	sev	ss
Apache server	0.9	10	9	7	10	10	10	0	0	8.85	8.88
Atom editor	-	-	-	-	-	-	-	-	-	-	0
Chrome	1	10	10	7	10	10	4	1	0	6.55	7.24
Django	0.7	10	7	7	10	7	7	0	0	6.6	6.68
Docker	-	-	-	-	-	-	-	-	-	-	0
Mozilla Filezilla	-	-	-	-	-	-	-	-	-	-	0
Firefox	1	10	10	7	10	10	7	0	1	8.1	8.48
Keepass2	-	-	-	-	-	-	-	-	-	-	10
MongoDB	-	-	-	-	-	-	-	-	-	-	0
MySQL	0.9	10	9	10	10	4	7	0	0	5.85	6.48
neat-project	-	-	-	-	-	-	-	-	-	-	0
Neo4J	-	-	-	-	-	-	-	-	-	-	0
OpenSSL	0.7	10	7	7	10	10	10	0	0	8.85	8.48
PHP	1	10	10	4	4	4	4	1	0	4	5.2
Python	0.7	10	7	7	7	4	7	0	0	5.4	5.72
Ruby	0.7	10	7	7	10	7	7	0	0	6.6	6.68
Ruby on Rails	0.7	10	7	7	10	7	7	0	0	6.6	6.68
Swift	-	-	-	-	-	-	-	-	-	-	0
tar	0.7	10	7	7	7	7	7	0	0	6.3	6.44
Tor browser	0.7	7	4.9	7	4	7	7	0	0	6	5.78
Ubuntu	0.7	7	4.9	7	7	7	7	0	0	6.3	6.02
Wordpress	1	10	10	10	10	10	7	0	0	7.65	8.12

TABLE 6.1: The security scores results for the selected projects and shows how the majority of the projects being in the range from 5-7, and a few projects given a high severity score mostly because of the small project size.

The results show a data set of open source projects being evaluated by the Trustworthiness of Open Source Software (toss)[23], and the data can be split in 2 with projects either being evaluated by their users or being evaluated by the general security score. The user evaluation will be elaborated in section 6.1.1.1. The other evaluations are made in the range from 3.44 to 10, and for these results the lowest score is 5.2 and highest is 8.8. The scale for security score is based on severity and thus lowest is the most trustworthy projects. The idea was the scale would be equal to the CVSS in the severity levels, but seems to be quite denser compared to the CVSS severity scale.

The severity have a few problems with the scale, since the 2/3s of the scale is in use and should have been more to compare to the severity scale. The constants will have to be changed to cover more of the scale, instead of 0 is only obtainable when considering

user evaluation. The constants are both for the vulnerability score and severity score, which could be changed from the grade choice in the trend evaluation. The constants for evaluation will have to be thoroughly checked and evaluated, which can take significant time and have been spend differently for this project. Another solution could be to transform the limited score to the scale from 0 to 10 instead.

6.1.1.1 User evaluation

The user evaluation uses the projects evaluated using the users and contributors from figure 6.1. The user evaluation is found by the formula 3.4 and all the security scores calculated will thus either be 0 or 10. The score of 0 or 10 is mostly used in the user evaluation, since a 10 score will be the equal of the project being the worst possible in terms of security. The data used for the user evaluation is seen in figure 6.2.

Project	Users	Contributors	Security score
Atom editor	23	822	0
Docker	104	532	0
Mozilla Filezilla	1943	1	0
Keepass2	225	2	10
MongoDB	386	106	0
neat-project	0	23	0
Neo4J	20	68	0
Swift	1	307	0

TABLE 6.2: The user evaluation data used to evaluate the projects security score with the annually vulnerability count is below 5. The data shows contributor count often rise above user count for not commonly known projects, and only in very well known projects does the user number rise above the contributor count.

The User information is gathered from OpenHub, where the individual users claim their use of a software project, but the contributors are automatically added from data of their commits. The results thus shows that some projects have more contributors than users for this reason, which is especially common for the small and medium sized projects, whereas larger and well-established projects have a large amount of users. Considering the use of OpenHub is known although not used by all contributors, the user data might not be accurate and a project with 100 users are actually known and well used by many in the open source community. KeePass is a quite known password manager in the open source world, and 225 users are a large number for a small product, which are not used for developing software, but as a software product for the end users.

On the other hand the contributor count being larger than 15 will result in a trustworthy software. The neat-project is quite new and is developed by a group in Norway, and the project is meant to change how the transport layer is for computers. The software is not finished and only the first versions seems to be out for the public. The software can not be known to be secure or trustworthy at this point, but the software gives a score for the project to be undoubtedly secure. The contributors could be set a bit higher, but the future will show if this software will be secure or not.

From the data the user evaluation seems to be working as intended for well known projects, where OpenHub contains data on the projects. A change could be made for the user count to be lower in the range of 100-200, and the contributors limit seems fine with 15. The contributors in a project with 15 people is a group, which should be able to review their own code, if the group focuses and do work on the security to be of importance in the project.

6.1.2 Dependencies

The dependencies uses the security score for evaluating the dependencies and is with the security score of the main project the aggregated security score. The dependencies are found by apt-rdepends for Linux package managers and are thus used for finding the dependencies in Linux software and will be different from other operating systems' executables. The figure 6.3 shows all the dependencies data found and used to create a security score for the project.

Project	Dependencies	0 scores	10 scores	Other scores	AS score
Apache server	108	97	5	6	10
Atom editor	180	167	8	5	10
Chrome	156	146	6	4	10
Django	3	2	0	1	6.68
Docker	15	14	0	1	5.78
Mozilla Filezilla	147	140	4	3	10
Firefox	126	118	5	3	10
Keepass2	81	76	3	2	10
MongoDB	50	44	4	2	10
MySQL	91	74	12	5	10
neat-project	1	1	0	0	0
Neo4J	1	1	0	0	0
OpenSSL	18	14	1	3	10
PHP	111	103	4	4	10
Python	34	30	1	3	10
Ruby	39	33	1	5	10
Ruby on Rails	85	75	4	6	10
Swift	75	69	3	3	10
tar	9	7	0	2	6.44
Tor browser	39	33	3	3	10
Ubuntu	1	0	0	1	6.02
Wordpress	185	168	10	7	10

TABLE 6.3: Dependencies data for open source projects based on Linux dependencies and their distribution of the security score in the dependencies of the projects.

The dependencies for the different projects show the amount of dependencies and how the scores are distributed. As seen in the projects' data many dependencies have scores of 10, based on the user evaluations. Most of the dependencies will have a score of 0, because of either 2 cases where data is not available in the project or the project has a good state of users and contributors. Most of the dependencies receive a 0 value for the reason that the projects are not found in OpenHub and does not have many vulnerabilities discovered. These dependencies with a 0 values can not be with these situations with most being caused by the information being unavailable.

The only projects with very few dependencies does not have a Aggregated Security score less than 10 with most having less than 9 dependencies for the project not to have a 10 score. The projects which in general have more than 18 dependencies in their project, will

thus have at least one 10 score for the project, and thus the project is completely insecure from a security point of view. The view of the weakest link of the chain in the software dependency chain, does make sense but most projects will receive a 10 score from the score. The 10 score does require data from OpenHub and in that case the project has been submitted by contributors in the project. The other scores have been assigned a score by the general security score, which are seen in figure 6.1. The 10 score seems to be significantly common for the Linux dependencies in the security aspect, since only few of the list of well known projects have a score of less than 10. 3 of the 6 with a score other than 10 is caused by no dependencies at all, where neat-project is most likely the only where the software will have to be developed without use of dependencies. Ubuntu does not have dependencies, as it is not part of the package library, since it cannot be installed as a software package in Linux. Neo4J is distributed by an executable even in Linux and does thus not exist in the same package library. The 10 score will be recommended to be changed in a fashion in order for not most projects to be scored with a 10 score.

6.1.3 Entirety

The entirety of the aggregated security score has a few issues to work as a fully functioning security metric for a project. The score is using much data and the idea is good, but a few issues with security score and user evaluation will need for the security metric to be even better.

The security score scale is a problem if it is to use the same severity scale as CVSS, which would be the best recommendation in order not to cause more confusion about the scale. The user evaluation should change the data not available to be another score than entirely trustworthy, since it does actually not describe much other than the project is of no importance and might as well not count toward the security metric. The user evaluation could equally have different user and contribution values to count toward different scores instead of having just a single value to determine completely trustworthy or untrustworthy. The dependencies are actually fine as is, if the problems are changed with the user evaluation.

The entire of the Aggregated security score is actually not of a high quality, which should have been addressed and prioritised more in this project to improve the score. The score did seem like it was of a good quality and did not need more improvements, and the focus on finding other metrics should have included a more thorough examination of the security metric. The missing parts and issues with the metric have unfortunately not been discovered in good time to create a better metric for the project, but more with continuous work with the metric.

6.2 Maintainability score

The maintainability score is quite simple both in the computation seen in figure 3.8. The maintainability results are shown below in the table 6.4, which shows how these well mostly well known open source projects current status in their maintainability with the lines of code to comments ratio.

Project	Code/Comment ratio	Maintainability score
Apache server	10.29	0.19
Atom editor	8.26	0
Chrome	15.55	3.70
Django	18.23	5.49
Docker	10.93	0.62
Mozilla Filezilla	12.98	2.00
Firefox	16.75	4.5
Keepass2	16.74	4.49
MongoDB	21.29	7.53
MySQL	19.49	6.32
neat-project	6.67	0
Neo4J	19.64	6.43
OpenSSL	18.20	5.47
PHP	13.97	2.65
Python	15.22	3.48
Ruby	12.18	1.46
Ruby on Rails	16.07	4.04
Swift	20.08	6.72
tar	11.99	1.32
Tor browser	23.22	8.81
Ubuntu	17.08	4.72
Wordpress	27.49	10

TABLE 6.4: The maintainability score results shows how most of the projects are within the decided range for the scale with only 3 scores being either 0 or 10. The range is found by looking at a large data set of projects and the results shows that this is equally found to be true with these projects.

The maintainability score shows how most of these projects is in the range from 10% to 25% ratio, which is the decided range of the scale. The scores are all within range except 3 projects, where 2 are only a few percent off and the last 3.33 percent off. The scores seem to be fine within this range, but the average of all the scores is only 3. The range

could thus be reevaluated to only being 10-20 percent of the code, which would result in a score with 5.9 in average. To find the best possible score for the range a large data set is needed and it needs to be representable for the entire software industry. I have not been able to find a number from any sources with an exact best practise limit, but with this data the scale does seem to be a little high with 25% as the upper bar. The 25% is thus a good goal for commenting a line for every 4 lines of code and would be necessary for explaining all the actions in the code.

6.3 Team score

The team score is used to describe how the current contributors experience and performance have been with their previous projects. The perfect situation would be to test each contributors performance in their commits, but this is unfortunately not possible. The contributors found are only the contributors currently contributing, which means the contributors with commits during the last 12 months. The contributors are thus evaluated on the projects which they have been part of and committed to. The data from each project is found in table 6.5.

Project	Contributors	Projects	0 scores	10 scores	Other scores	Team score
Apache server	27	205	181	19	4	5.71
Atom editor	822	786	753	30	3	9.89
Chrome	1584	619	565	40	14	5.76
Django	369	508	489	16	3	5.42
Docker	532	1215	1131	63	21	9.91
Filezilla	1	7	7	0	0	10
Firefox	1087	1601	1479	95	27	5.14
Keepass2	2	1	0	1	0	0
MongoDB	106	118	111	4	3	9.96
MySQL	126	314	285	20	9	4.62
neat-project	23	22	18	3	1	9.87
Neo4J	68	1510	1446	53	11	9.5
OpenSSL	126	223	207	11	5	3.02
PHP	170	2776	2607	131	38	5.49
Python	47	518	472	36	10	9.08
Ruby	42	228	209	16	3	7.29
Ruby on Rails	554	1826	1740	74	12	6.29
Swift	307	287	267	16	4	9.9
tar	4	34	27	3	4	5.63
Tor browser	80	293	261	22	10	5.75
Ubuntu	25	1507	1348	130	29	7.79
Wordpress	34	96	88	8	0	3.49

TABLE 6.5: The overall results for the team score on various projects using a limit of 10. The results shows the different projects are evaluated by their contributors and projects and how the projects are distributed with their scores.

The results show the same issue with most scores being evaluated either 0 or 10 by the user evaluation, which might be an issue with a naming convention that does not match for all the different sources used. The projects are evaluated with a limit on projects and contributors to 10 commits, which will be further explained in the section 6.3.1. The contributors are weighted by their activity by commits in the project, which means a contributor might count for 10% of a huge project with a massive amount of commits. The novices in the projects might only have a few commits and not really count for much, and especially setting a limit is important with larger projects as the contributors with few commits does not play a big part in the project. The amount of evaluators are quite varying by the smaller and bigger projects. The contributors found does have different

amount of activity and contributions to projects. A contributor in neat project have about 130 different projects that he contributed to, while other contributors will only have contributed to a few and might not show up in the table if they have not contributed more than 10 commits to a project. With most of these projects the team score have a significant amount of projects to go through, but with Filezilla, KeePass and Tar only few contributors are actually contributing currently. These projects are thus not being developed much further currently, and especially KeePass that the 2 contributors with only experience in this project.

The average score for these projects are 6.8 as a score and the score seems to be distributed over most of the scores, but these projects are mostly quite successful and thus should the score be sort of high with these projects compared to an average over all projects in OpenHub. The projects are not too high as a limit is set, but would otherwise be about 10 times bigger. The projects should be in the top half of the spectrum with many 0 scores, since the 0 score will result in a score closer to 10 since the scores are security score and thus opposed to the team score. The team score from these data does seem to give a fine expression of how the projects' contributors are skilled. The weighting of the score seems fair by the most active contributors being considered more in the score and thus will have a more important role in the project.

6.3.1 Setting limits

Setting the limit is to limit the run time and the amount of projects that have to be iterated and scored for the team score to be calculated. A few different projects are shown in table 6.6. The limits set for the projects, means that contributors in the project is not considered if they have committed less than 10 commits, and the projects are not considered from a contributor if the contributor have committed less than 10 commits.

Project	Contributors	Projects	Team score
OpenSSL	362	2320	3.40
OpenSSL (10 limit)	30	223	3.01
Tor (no limit)	510	2114	5.81
Tor (10 limit)	190	293	5.75
Tor (25 limit)	140	151	5.72
Tor (50 limit)	108	117	5.68
Ubuntu	771	10055	7.96
Ubuntu (10 limit)	316	1507	7.79

TABLE 6.6: Team score evaluation results with limitations set on contributors and projects evaluated. The projects are quite different and does thus present a large part of open source projects. The results shows that a limit set on 10 would not deviate the score too much but save significant time on the projects to calculate.

The projects are significantly different in both size and use, which is on purpose to show that the projects, will have similar score even with limits and higher limits than used. The scores are almost the same with a limit set to 10 for both contributors and projects, which have been used for the table 6.5. By setting a limit of 10, the projects needed to be scored can be reduced by about a factor 10. A factor 10 is significant for example with Ubuntu where 1507 projects takes significantly less time compared to 10055 projects. 10000 projects will use about 8 hours of calculations, while 1507 is about an hour spent on calculating, and the team score is a little off with about 4 percent with OpenSSL. Firefox will spend about 2 hours calculating all the projects from the contributors and have 9700 projects to evaluate.

The limit of 10 is set for the team score results shown in this section, as the limit of 10 will for most projects have less than 1 percent from otherwise calculated result. A score being 1 percent off is fine compared to the time saved from calculating many more projects.

6.4 Trustworthiness score

The trustworthiness score shows the overall trustworthiness based on the previously evaluated metrics and is based on the formula 3.11. The overall trustworthiness will only receive a high score if both the aggregated security score is low and team score is high, which can be a problem especially for the aggregated security score for larger projects

with a single score of 10 as dependency. The results of the scores is presented in table 6.7[2].

Project	TAS score	Team score	Maint score	Trust score
Apache server	0	5.71	0.19	1.45
Atom editor	0	9.89	0	2.47
Chrome	0	5.76	3.7	1.81
Django	3.32	5.42	5.49	4.06
Docker	4.22	9.91	0.62	5.28
Filezilla	0	10	2	2.7
Firefox	0	5.14	4.5	1.73
Keepass2	0	0	4.49	0.45
MongoDB	0	9.96	7.53	3.24
MySQL	0	4.62	6.32	1.79
neat-project	10	9.87	0	8.97
Neo4J	10	9.5	6.43	9.52
OpenSSL	0	3.02	5.47	1.3
PHP	0	5.49	2.65	1.64
Python	0	9.08	3.48	2.62
Ruby	0	7.29	1.46	1.97
Ruby on Rails	0	6.29	4.04	1.98
Swift	0	9.9	4.23	3.15
tar	3.56	5.63	1.32	3.85
Tor browser	0	5.75	8.81	2.32
Ubuntu	3.98	7.79	4.72	5.01
Wordpress	0	3.49	10	1.87

TABLE 6.7: The trustworthiness score with the major metrics to show how the result of the trustworthiness metric is. The results are quite low for most major projects while some of the smaller projects do have higher scores, which is mainly caused by the trustworthy aggregated security score.

Most of the data is having low scores in general, which are most likely because the larger projects are not able to receive a score different from 0 as trustworthy aggregated security score, as they will have at least a single score with the value 10 for AS score. The values can thus not result in a score higher than 3.5 with team score and maintainability score being perfect. The only scores higher than 3.5 in trustworthiness score is the 6 projects with few dependencies and thus smaller chance for having dependencies, which

[2]Abbreviations are as follows TAS (trustworthy aggregated security score), Maint score (maintainability score) and Trust score (trustworthiness score)

are untrustworthy. The trustworthiness score's issue is that it discriminates against larger projects, and instead of making the projects comparable it is biased towards smaller projects. Smaller projects are at any point easier to create trustworthy, since the goal and the functionality is significantly smaller, but the quality of the different projects should be comparable even with different sizes.

The overall trustworthiness metric does give a good indication of the metrics chosen for the project. The metrics does work and describes the trustworthiness, albeit the metrics can always be perfected to be more indicating and significant in the scoring.

Chapter 7

Conclusion

The trustworthiness of software is an important aspect of software development, since users would otherwise find an alternative to the software product. The trustworthiness from the users can be influenced by different attributes or a combination of these attributes. Trustworthiness is difficult to find a metric to describe, since trustworthiness is caused by many aspects of software engineering, and unfortunately have a solution not been found for the complete assessment of trustworthiness.

The metrics that have been analysed and examined with this project is maintainability, security and team capability. The maintainability of a software product is important for both further development and maintaining a high quality of software. The maintainability metric is the simplest metric in this project with only little details and can be extended with further information on the complexity of the source code. The security metric have some work to become neutral and compatible with CVSS severity levels for easier understanding of the resulting overall security score. The security metric is of great importance in trustworthiness with previously being the only consideration, when judging the trustworthiness in a software project. The team capabilities are important for open source software projects, since most projects are separated groups working toward a common goal with only little interactions over a distance. The teams are by the team metric evaluated based on their experience and resulting projects, in order to evaluate their capabilities of creating a secure and trustworthy open source project.

The team metric indicates the portfolio of the contributors by looking at the security of their projects. The metric can be improved by looking at less data for calculating the score, since the data mining require a long time to evaluate larger data sets. The metric itself is good with the score of all the contributors projects and can further be improved by looking into the aggregated security score. The security score was determined to be of fine quality, but at the end of the project more work on the metric was determined

to be needed, which would have been done if the issues was found earlier in the process. The score is biased towards smaller projects, and the larger projects are thus evaluated to being less trustworthy by default. The user evaluation can be confusing and could be improved by more complex measured instead for a completely trustworthy or completely untrustworthy. The security score is although a good score for scoring security in terms of trustworthiness, and with improvements the security score could be more understandable and expressive. The maintainability metric can be improved with more expressive complexity measurements, but the simplicity is equally an advantage of the metric, which would be nice to preserve.

The metrics on security, team capabilities and maintainability each describe a few possible attributes of computer science to describe trustworthiness of a software product. The trustworthiness score is expressive and useful in describing if users should trust the software project or not. The attributes are chosen to create a trustworthiness score, and these metrics are in no way all the attributes. Other attributes are possible for others to extend the score with descriptive attributes to trustworthiness. The trustworthiness score is a good intermediate score with the attributes examined until now, but will require more work to be useful for software developers and architects. The open source products of most interest to these stakeholders would be smaller projects, which they can use for their own development.

7.1 Future works

The future works are plenty and I hope other students will find the project interesting and useful in continuing the creating of a trustworthiness metric.

The future work I see immediately ahead is to redesign parts of the Aggregated Security score (previously dependency score) using the information presented in this project, which are possibly a smaller task with the analysis and evaluation of this thesis. The security metric is important for trustworthiness and an improvement would contribute the trustworthiness score a long stride ahead.

Other future work is to find other attributes which are measurable and related to trustworthiness, these new attributes will require a significant workload to both find data and a neutral evaluation of open source software. Ideas for these metrics can be found in this thesis, but new ideas and eyes on the task can equally be a strength in the expansion of a trustworthiness metric.

Appendix A

The 44 Trust principles

The 44 principles from [1] can be found on the following pages. The pages are an exact copy from the article's trust principles.

Appendix 1: Trust Principles

For GPALS software development, specific trust principle statements and associated sets of detailed compliance requirements have been developed. These statements are shown below, but for lack of space, the compliance requirements have been omitted (full trust principle compliance requirements documentation runs to over one-hundred pages). Each trust principle is intended as a requirement on the development process that should increase the trustworthiness of any developed software. Readers interested in the full set of compliance requirements should contact the authors.

A few of the concepts mentioned in the statements below may be unfamiliar to some readers. For example, several references are made to terms such as Computer Software Configuration Items (CSCIs), Computer Software Component (CSC), and Computer Software Unit (CSU). These terms refer to standard software components in the DoD-STD-2167A Military Standard for Software Development. The documents mentioned (e.g., SDD, IDD) are also standard in DoD-STD-2167A. A concept mentioned repeatedly below is the "identified software lifecycle activity". This refers to the software development or maintenance activity relevant to the principle at hand (defined more specifically in the compliance requirements documentation).

Access Control: Identified software lifecycle activity shall be automatically controlled by the software engineering environment with respect to an explicitly defined security policy.

Administration: The software engineering environment, software tools, and the developed software shall be maintained according to explicit administration documentation by qualified individuals.

Auditing: A record of identified software lifecycle activity shall be automatically logged and stored by the software engineering environment in a protected repository.

Configuration Management: A configuration management system shall be established and shall include mechanisms and explicit procedures for configuration identification, configuration accounting, configuration control, and configuration auditing. All configuration items shall be stored in a protected repository that maintains all software versions, software modification requests, and software changes.

Design Documentation: In addition to the Software Design Document (SDD) and Interface Design Document (IDD), the characteristics of the design activity, critical design alternatives considered, and critical design rationales shall be documented.

Design Review: Design peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the software design.

Design Tools: Design Computer-Aided Software Engineering (CASE) tools shall be employed to maintain design/requirements traceability mappings and to generate design documentation.

Design Traceability: All aspects of the design shall be shown to be traceable to the requirements and all requirements shall be shown to be traceable to the design.

Environment and Tool Selection: The software engineering environment and all software tools shall be selected according to an explicit selection policy that considers the trust rating, maturity, documentation, and source code availability of that software.

Environment Integrity: An explicit procedure shall be available for identifying changes in the software engineering environment components and, if required, to restore the integrity associated with that software.

Formal Design Specification: In addition to any informal design specification, the design shall also be specified in a formal specification framework.

Formal Design Verification: A formal verification shall be performed to prove that the formal design specification correctly meets its requirements.

Formal Methods Approach: All formal specification and verification activities shall follow an approach which includes the use of a formal specification and verification toolset, documentation, peer reviews, and traceability mappings.

Formal Requirements Specification: In addition to informal requirements specifications, functional requirements shall also be specified in a formal specification framework.

Formal Source Code Verification: A formal verification shall be performed to prove that a low-level formal specification of the source code correctly meets its requirements.

Identification and Authentication: The initiation of an identified software lifecycle activity shall be done only be an individual that has been identified and authenticated by the software engineering environment.

Intrusion Detection: Audit trail data shall be used to perform periodic and random intrusion detection analysis on the software engineering environment.

Least Privilege: Privileges to perform identified software lifecycle activity shall be allocated and maintained so that a privilege is only given to individuals who require that privilege.

Multi-Person Control: Identified software development activity shall not be completed without the active endorsement and involvement of at least two qualified software developers.

Planning: Detailed plans for all software development activity (including trust principle compliances) shall be described in a Software Development Plan (SDP) and the management of the software development shall follow the approach described in the SDP.

Prototyping Approach: All prototyping that is performed as part of the risk mitigation strategy shall be performed according to an explicitly defined prototype plan that describes the way in which the prototype is designed, developed, tested, documented, and protected.

Prototype Software Reuse: When prototype software is reused in the developed software, then the prototype software shall be sufficiently documented, reviewed, and tested to ensure that the level of trust is commensurate with the developed software.

Requirements Analysis Documentation: In addition to the Software Requirements Specification (SRS) and Interface Requirements Specification (IRS), information useful in understanding the software requirements analysis process and a rationale for all critical requirements analysis decisions shall be documented.

Requirements Analysis Review: Requirements analysis peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the software requirements.

Requirements Analysis Tools: Requirements analysis Computer-Aided Software Engineering (CASE) tools shall be employed that provide for requirements specification, consistency checking, and documentation generation.

Requirements Traceability: All software requirements shall be shown to be traceable to an explicit system requirement or

customer source and all system requirements allocated to a Computer Software Configuration item (CSCI) shall be shown to be traceable to a software requirement.

Reliability Measurement: Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) test and field results shall be used to reduce observed software failure rates to acceptable levels.

Risk Mitigation: All potential risks associated with the software development activity shall be explicitly identified and a risk mitigation strategy shall be documented and complied with throughout the software lifecycle.

Security Policy: All software development activity shall be performed in accordance with an explicitly defined and enforced security policy with respect to all software developers and software development resources.

Shared Knowledge: Each identifiable component of the software development activity, including all aspects of requirements, source code, design, tests, proofs, software tools, methodologies employed and support activity, shall be associated with at least two individuals who are thoroughly familiar with the details, implications, and alternatives considered for that component.

Software Reuse: All reused software shall be subject to an explicit selection policy that considers the trust rating, maturity, documentation, and source code availability of the software.

Source Code Standards: An explicitly defined source code standard that enforces modular, structured programming shall be complied with throughout the coding activity.

Source Code Analysis: All developed code shall be subjected to code analysis using tools and procedures that measure complexity and style.

Source Code Review: Source code peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the source code and Computer Software Unit (CSU) tests.

Source Code Documentation: The source code and characteristics of the software coding activity shall be documented.

Source Code Traceability: All source code shall be shown to be traceable to the design and Computer Software Unit (CSU) tests. The design shall be shown to be traceable to the source code.

Test Documentation: In addition to the Software Test Plan (STP), Software Test Description (STD), and Software Test Report (STR), the characteristics of the Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) test activity shall be documented.

Test Responsibility: The responsibility for Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) testing shall be placed with an independent group or organization not involved with coding or design of the software being tested.

Test Review: Test peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the tests.

Test Strategies: All Computer Software Unit (CSU), Computer Software Component (CSC), and Computer Software Configuration Item (CSCI) test and integration tasks shall include provisions for various testing strategies.

Test Tools: The software engineering environment shall include a testbed for creating, executing, documenting, and analyzing the completeness of all tests.

Test Traceability: All Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) tests shall be shown to be traceable to the software requirements. Both the source code and the software requirements shall be shown to be traceable to the CSC and CSCI tests.

Trusted Distribution: All software shall be transferred from its source to its destination in a way that ensures that the integrity has not been compromised during the transfer.

Trusted Path: An explicit mechanism shall be included in the software engineering environment to ensure that identified software lifecycle activity cannot be intercepted by unauthorized means.

Appendix B

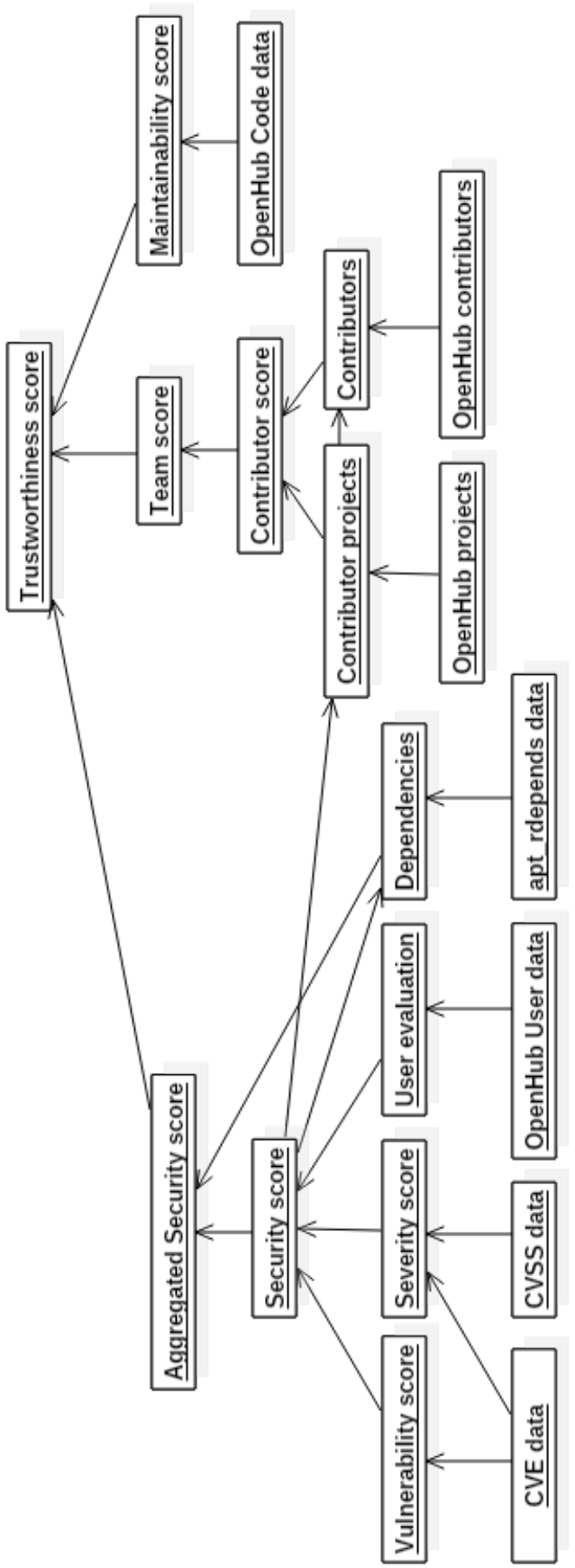
Class diagram

ProjectContributorMetric
-contributors: List
-contributors_details: Map
-metric: List
-original_contributors: List
-project_details: Map
-projects_details: Map
+project_name: String
-projects: List
+init(project_name)
-build_metric()
+get_contributor_details(contributor_name: String)
+get_contributor_project(contributor_name: String)
+get_project_contributors(project_title: String)
-calculate_project_scores(contributor_name: String)
-calculate_contributor_score(contributor_name: String)
+get_contributor_score(contributor_name: String)
+get_contributors_score()
0..1
0..*
OSSProject
+project_name
+project_short_name
+project_details
+init(name: String, short_name: String = None)
-calculate_dependency_score(project_name: String)
+get_dependency_score()
-calculate_dependencies_scores()
+get_dependencies()
+get_project_details()
-calculate_maintainability_score()
+get_maintainability_score()
-calculate_team_score()
+get_team_score()
-calculate_trustworthiness_score(just_dependency_score: Boolean = false)
+get_trustworthiness_score()
+get_project_contributor_metric()
WebSearch
+search_general_openhub(project_name: String)
+find_project_contributors(project_name: String)
+find_project_details(project_name: String)
+number_of_pages(url: String, project_name: String)
+find_contributor_projects(contributor_name: String)
+find_project_code_data(project_short_name: String)
+html_to_bytes(html_list: List)
+openhub_online()
Dependencies
+openhub_browser_grade(project_name: String)
+calculate_severity_score(cve_severity_count_yearly: Map)
+separate_yearly_severity_level(cve_cvss: Map)
+count_per_year(cve_list: List)
+calculate_grade_ncve(ncve: Float)
+calculate_trend_grade(cve_in_years: Map)
+remove_current_year_from_dict(dict: Map)
+find_dependencies(projectname: String)
+shorten_string_version_number(dependency_name: String)
CVESearch
+search_cve(name: String)
+search_cvss(cve_list: List)
Utilities
+string_to_int(string: String)
+string_is_a_number(string: String)
+string_is_a_datetime(string: String)
+get_current_year_str()
+get_cve_year(cve_string: String)

a

Appendix C

Metrics diagram


Trustworthiness score
Maintainability score
OpenHub Code data
Team score
Contributor score
Contributors
OpenHub contributors
Contributor projects
OpenHub projects
Aggregated Security score
Security score
Dependencies
apt_rdepends data
User evaluation
OpenHub User data
Severity score
CVSS data
Vulnerability score
CVE data

Bibliography

[1] John Watson Edwurd Amoroso, Carol Taylor and Jonathan Weiss. A process-oriented methodology for assessing and improving software trustworthiness. *Proceedings of the 2nd Acm Conference*, 2(2):39–50, 1994. URL http://dl.acm.org/citation.cfm?id=191188.

[2] ISO. Information technology - security techniques - information security risk management. Standard, International Organization for Standardization, Geneva, CH, June 2008.

[3] Inc. Black Duck Software. Discover, track and compare open source, 2014-2016. URL https://www.openhub.net/.

[4] OpenSource.com. General information on open source software and the community, 2016. URL https://www.opensource.com.

[5] OpenSource.org. Open source definition (annotated), 2016. URL https://opensource.org/osd-annotated.

[6] OpenSource.com. Four types of open source communities, 2013. URL https://opensource.com/business/13/6/four-types-organizational-structures-within-open-source-communities.

[7] CW Krueger. Software reuse. *Computing Surveys*, 24(2):131–83, 1992. URL https://dl.acm.org/citation.cfm?id=130856.

[8] Mohamed Bishr Andreas Metzger Holger Koennecke Sandro Hartenstein Nazila Gol Mohammadi, Sachar Paulus and Klaus Pohl. An analysis of software quality attributes and their contribution to trustworthiness. *Closer 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science*, 3(3): 542–552, 2013. URL https://www.scopus.com/record/display.uri?eid=2-s2.0-84884474760&origin=inward&txGid=16B7A2698DA668A530CDCB47F6A4675B.wsnAw8kcdt7IPYLO0V48gA%3a.

[9] Beni Suranto. Evaluating trustworthiness of software components. *Teknoin*, 21 (1), March 2015. URL https://www.researchgate.net/publication/281344524_EVALUATING_TRUSTWORTHINESS_OF_SOFTWARE_COMPONENT.

[10] Anne Immonen and Marko Palviainen. Trustworthiness evaluation and testing of open source components. *Proceedings - International Conference on Quality Software*, 2007. URL http://findit.dtu.dk/en/catalog/6438663.

[11] National Institute of Standards and Technology (NIST). Risk management guide for information technology systems. Standard, National Institute of Standards and Technology, Gaithersburg, Maryland, USA, July 2002.

[12] Mitre. About cve, 2016. URL https://cve.mitre.org/about/.

[13] National Vulnerability Database. National vulnerability database, 2016. URL https://nvd.nist.gov/home.cfm.

[14] FIRST. Common vulnerability scoring system v3.0: Specification document, 1995-2015. URL https://www.first.org/cvss/specification-document.

[15] Codenomicon. The heartbleed bug, 2014. URL http://heartbleed.com/.

[16] National Vulnerability Database. Vulnerability summary for cve-2014-0160, 2016. URL https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160.

[17] Thomas; Zeller Andreas Neuhaus, Stephan; Zimmermann. Predicting vulnerable software components. *CCS '07 Proceedings of the 14th ACM conference on Computer and communications security*, 14:12, 2007. URL https://www.researchgate.net/publication/281344524_EVALUATING_TRUSTWORTHINESS_OF_SOFTWARE_COMPONENT.

[18] Hassan Raza Bhatti. Risk management guide for information technology systems. Master thesis, Lulå University of Technology, Department of Computer Science, Electrical and Space Engineering, July 2010.

[19] Debian. Debian policy manual, chapter 7 - declaring relationships between packages, 1997-2016. URL https://www.debian.org/doc/debian-policy/ch-relationships.html.

[20] Harvey Maylor. *Project Management*, chapter 11, pages 242–264. Pearson, 4th edition, 2010.

[21] Cristina García García. Reputation management of an open source software system based on the trustworthiness of its contributions. Technical report, Technical University of Denmark & Universida de Oviedo, May 2015.

[22] cve search. cve-search, January 2017. URL https://github.com/cve-search/cve-search. cve-search repository for GitHub project.

[23] Michael B Nielsen. Trustworthiness of open source software, 2016-17. URL https://github.com/mbnielsen/toss. Github repository for the TOSS project.