Master Project Proposal

Núria Bruch Tàrrega

January 2020

1 Project details

Project title: Measuring the degree of library dependency

Student Name: Núria Bruch Tàrrega
nuria.bruch@gmail.com
Student id: 12783145

Host organization: Software Improvement Group (SIG)
Fred. Roeskestraat 115-123, 1076 EE Amsterdam
www.softwareimprovementgroup.com

Contact person: Lodewijk Bergmans
Senior Researcher
l.bergmans@sig.eu

2 Project summary

Nowadays, many open-source libraries are used by developers to be able to reuse the features that these libraries implement. This implies that a significant number of projects depend on open-source as a concept [16]. Maintaining these dependencies and managing the vulnerabilities or problems that these can cause is not a trivial task.

Furthermore, it is a critical task. For example, some vulnerabilities can become security problems that can have a negative impact in terms of integrity, privacy or availability [26]. This may make it necessary to replace the library being used, to prevent these problems from spreading to the project.

Replacing a library dependency could be a very costly process. For instance, it could involve determining which modules of your project are affected, and which functionalities of the library are being used and need replacement. As well as which is the best way to replace them (i.e. using another library or developing them in-house). In addition, similar problems may arise if a library becomes closed-source.

Currently, most of the package management systems include dependency management, but these are only listing which dependencies exist in each project [11]. However, a more detailed risk evaluation of the open-source dependencies is missing and could be useful for many projects. It would be beneficial to analyze the coupling between the project and the libraries that are being used, as well as the effort that replacing a library would involve. It would provide the developers with a more extensive evaluation to be considered when assessing the risk of the project.

This project is carried out in collaboration with the company *Software Improvement Group (SIG)*, and it is motivated by the *FASTEN* project [1]. The objective of this project is to improve the quality of open-source development environments to make them more secure and reliable. For this reason, the FASTEN project aims to analyze the software library dependencies that the projects have, in more detail.

According to the problem described above, we specify the following research questions:

- **RQ1:** How can we measure the degree of source code dependency between a software project and the open-source libraries it uses?
- **RQ2:** How can we measure the effort required to replace the use of an open-source library in a given software project?

[1] `https://www.fasten-project.eu/`

3 Problem analysis

Therefore, as stated with the research questions in section 2, the primary goals of this project are: Measuring the degree of source code dependency between a project and the libraries it uses, as well as measure the effort needed to replace these libraries.

3.1 Measuring the degree of source code library dependency

To the best of our knowledge, there are no papers about measuring the degree of dependency between two separate projects. However, it is true that the degree of dependency between two classes or modules of the same project has already been measured many times, using coupling metrics [29].

Therefore, we propose re-using the already existing coupling metrics, meant to measure the coupling between units of the *same* project and adapt them to measure coupling *between* projects.

According to Poshyvanyk and Marcus in [22], there are five main groups of coupling metrics:

- **Structural coupling metrics:** Measured directly from static source code analysis. Largely studied by the related literature.
- **Dynamic coupling measures:** Measured using dynamic code analysis. "Introduced as the refinement to existing coupling measures due to gaps in addressing polymorphism, dynamic binding, and the presence of unused code by static structural coupling measures" [22].
- **Evolutionary and Logical coupling:** According to [30], evolutionary coupling can: "tell us which parts of the system are coupled by common changes or cochanges."
- **Coupling measures based on information entropy approach:** Coupling metrics based on the information-theory approach, such as the metrics proposed by Allen and Khoshgoftaar in [1].
- **Coupling metrics for specific types of software applications:** Specialized coupling metrics for certain types of projects, such as knowledge-based systems or aspect-oriented approach.

The last two categories are much more domain-specific. Moreover, the evolutionary coupling is not possible to be applied in our context since it is very likely that the separate projects are not going to evolve at the same time, given that they are not developed by the same team. Therefore, the research of this project, owing to the time limitation, is going to be centered on the structural metrics.

These metrics are going to be proposed as a first step to measure the degree of library dependency. These metrics can be extended and calculated more accurately by adding dynamic coupling metrics in case time permits or in future work.

3.1.1 Structural coupling metrics

There are many structural coupling metrics, each one measuring a different type of coupling from a different perspective, depending on the purpose for which the metrics are needed. To decide which metrics can be applied to answer RQ1, we have compared all the metrics according to the papers from Briand et al. [4], Poshyvanyk et al. [22], and Harrison et al. [9]. Based on the contents of these papers, we have created Table 3.

The empty cells in Table 3 mean that the characteristic is unspecified for the metric in the papers. The *Types of connections* assigned in Table 1 as numbers correspond to the descriptions specified in Table 1.

In the same way, the column *Counting connections* has values from A to F, the meaning of which can be found in Table 2. The content of both tables, more detailed descriptions, as well as a complete comparison between each type of connection and each way of counting connections, can be found in [4].

#	Client Item	Server Item	Description
1	attribute *a* of a class *c*	class *d*, d != c	class *d* is the type of *a*
2	method *m* of a class *c*	class *d*, d != c	class *d* is the type of a parameter of *m*, or the return type of *m*
3	method *m* of a class *c*	class *d*, d != c	class *d* is the type of a local variable of *m*
4	method *m* of a class *c*	class *d*, d != c	class *d* is the type of a parameter of a method invoked by *m*
5	method *m* of a class *c*	attribute *a* of a class *d*, d != c	*m* references *a*
6	method *m* of a class *c*	method *m'* of a class *d*, d != c	*m* invokes *m'*
7	class *c*	class *d*, d != c	high-level relationships between classes, such as *uses* or *consists-of*

Table 1: Types of connections, obtained from [4]

To interpret the metrics suggested by Briand et al [3] (from IFCAIC to DMMEC in Table 3), it is necessary to know they are named according to three variables: relationship, locus, and type. The name of each metric is composed in the following way, the initials of the relationship, the initials of the type of interaction, and the initials of the locus. These initials are described below.

There are three types of relationships, listed below, which can be used to determine the coupling of a class *c*. All the definitions have been obtained from [3].

- Inheritance (A, D): Interactions from a class to its antecessors or descendents, depending on the locus.

Counting connections	Level	Description
A	Method or attribute	count individual connections
B	Method or attribute	count the number of distinct items at the other end of the connections
C	Class	add up the number of connections counted as in A) for each method or attribute of the class
D	Class	add up the number of connections counted as in B) for each method or attribute of the class
E	Class	count the number of distinct items at the end of connections starting from or ending in methods or attributes of the class
F	Class	for a class c, count the number of other classes to which there is at least one connection

Table 2: Counting connections, obtained from [4]

- Friendship (F, IF): Extension for C++, interactions from class to all the classes declared as friends or the classes that declare it their friend (inverse friends), depending on the locus.
- Other (O): interaction with classes that do not have an inheritance or friendship relationship.

The three different types of interaction descibed by Briand et al. [3], are the following:

- Class-Atribute (CA): "There is a class-attribute (CA-)interaction from class c to class d, if an attribute of class c is of type class d."
- Class-Method (CM): "There is a class-method (CM-) interaction from class c to class d, if a newly defined method of class c has a parameter of type class d."
- Method-Method (MM): "There is a method-method (MM-)interaction from class c to class d, if a method implemented at class c statically invokes a method of class d (newly defined or overriding), or receives a pointer to such a method."

Finally, the two types of locus are:

- Export from a class (EC): "Change flows away from a class", related to the descendants (D) and the friends (F).
- Import to a class (IC): "Change flows towards a class", related to the antecesors (A) and the inverse friends (IF).

Metric	Inheritance	Import or export	Types of connection	Domain of measure	Counting connections	Indirect coupling
CBO	both	both	5, 6	class	F	no
CBO'	no	both	5, 6	class	F	no
RCF_α	both	import	6	class	E	depends
RFC	both	import	6	class	E	no
RFC'	both	import	6	class	E	yes
MPC	both	import	6	class	C	no
DAC	both	import	1	class	C	no
DAC'	both	import	1	class	D	no
COF	no	both	5, 6	system	F	no
ICP	both	import	6	method, class, set	A, C	no
IH-ICP	only	import	6	method, class, set	A, C	no
NIH-ICP	no	import	6	method, class, set	A, C	no
IFCAIC	no	import	1	class		no
ACAIC	only	import	1	class		no
OCAIC	no	import	1	class		no
FCAEC	no	export	1	class		no
DCAEC	only	export	1	class		no
OCAEC	no	export	1	class		no
IFCMIC	no	import	2	class	C	no
ACMIC	only	import	2	class	C	no
OCMIC	no	import	2, 6	class	C	no
FCMEC	no	export		class	C	no
DCMEC	only	export		class	C	no
OCMEC	no	export		class	C	no
OMMIC	no	import		class	C	no
IFMMIC	no	import	6	class	C	no
AMMIC	only	import	6	class	C	no
OMMEC	no	export	6	class	C	no
FMMEC	no	export	6	class	C	no
DMMEC	only	export	6	class	C	no

Table 3: Coupling metrics comparison

Now that the metrics have been explained with the tables and the description of the metrics by Briand et al., we have to select the candidates to be used in the project.

In Table 3, there are some metrics that incorporate inheritance to calculate coupling and some that do not. Since we want to take into account each usage of the libraries in the projects, it is significant to measure both, inheritance and non-inheritance. Therefore, the metrics CBO', COF, IH-ICP and NIH-ICP will not be used for this project. The metrics by Briand et al. either only take the inheritance into account or not at all, but they can be combined to consider every type of dependency.

Furthermore, to take the *transitive* dependencies into account, it is important to count transitive coupling as well. In this manner, the metric RCF_α is truly interesting in this context.

Finally, the metrics that are being considered for this project, taking into account the criteria explained above, are the following:

- CBO
- RCF_α
- MPC
- DAC, DAC'
- ICP
- Metrics by Briand et al. (only for one of the locus)

3.2 Measuring the effort needed to replace a library

The typical way of estimating effort is by the means of models such as COCOMO II [24], which calculate the effort to generate new code. However, for the second research question, it is necessary to find a way to calculate the effort to modify the existing code instead of developing new code. In [14] and [2], Kama et al. present a tool for software change effort estimation, based on COCOMO II.

This tool, COCHCOMO, combines static and dynamic software analysis to calculate the effort to modify a part of the code. It takes the classes affected by the change into account, both directly and indirectly.

However, COCHCOMO is based on the requirements of the system. Given a change in a requirement, it estimates the effort needed to carry out the change, considering the classes affected by the requirement. The calculation process used in COCHCOMO could be adapted to base it on the replacement a library, instead of basing it on changes in the requirements.

It is also important to consider some new parameters added to this effort calculation. For instance, if the library is deemed to be replaced by another library (or multiple ones) or the features used from the library will be implemented in-house.

Plus, in [14] the methodology contemplates the state of each of the classes (i.e. it is finished, in development or not started). It will be examined if this consideration is applicable in the case of this project or not.

3.3 Proof-of-Concept

To calculate the metrics as a proof-of-concept, it is necessary to create a dependency graph of the project. It has to be a call-level graph since we require a fine-grained analysis of the source code dependencies. Moreover, it has to represent the software ecosystem of the project considering the various versions of the libraries it uses. This way, it will avoid creating false-positives with library versions the project does not really use.

Most of the literature about modeling software ecosystems, use the nodes of the dependency graph to represent libraries or versions of these [6, 10, 15]. They are using a package-based approach. The problem with this approach is that, by using this node definition, it is not possible to determine which parts of the project are affected by a particular dependency. The same happens with measuring this dependency. Furthermore, a package-based graph could define transitive dependencies that do not really exist (see Figure 1 for an example).

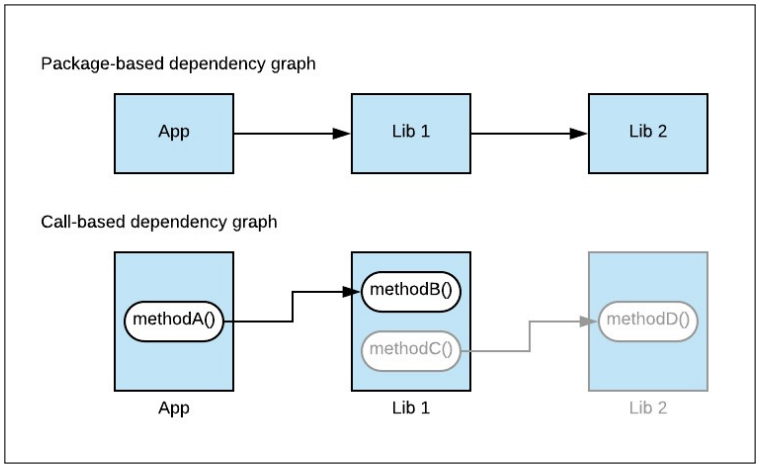

Figure 1: Comparison package-based and call-based dependency graphs.

Nevertheless, some papers do implement a call-based approach and therefore, could be used in this project.

Hejderup et al. in [12], propose the creation of a call-level dependency network, in which it can be determined which parts of the project depend on which library. Furthermore, this paper employs a different process to resolve dependencies, taking into account the evolution over time of the versions of the libraries on which the projects depend. Therefore, the analysis of the dependency grade and replacement effort would not be conducted exclusively per library, but also per version of each library.

Hejderup et al. [11], propose an approach to create a call-based dependency network, PRÄZI. This approach, as the previous one, takes the versioning of the different libraries into account. It also considers that a certain project can depend on various versions of a library at the same time. In the paper, they implement it to create the

dependency network of an entire software ecosystem (Crates.io). Nevertheless, this approach could be utilized to construct the network of a given project.

3.4 Validation of the metrics

Ultimately, the metrics used in this project will have to be validated. There is not a unique way to validate metrics which is globally accepted and used. Therefore, various approaches will be adopted to validate the metrics as holistically as possible.

In the paper [25], Srinivasan et al. explain that there are two fundamental approaches for metric validation: *theoretically* and *empirically*. Therefore, to provide a check of validity that is as broad as possible, a mixture of these two approaches will be used during this project.

For the theoretical validity of the coupling metrics, we will validate that all of them fulfill the *Mathematical Properties of Measures for Coupling*, described in [25].

Furthermore, Meneely et al. [18] describe a set of 47 validation criteria for metrics. The time limit of the project does not allow for a complete validation of all the metrics. Consequently, a sub-set of these criteria will be used to validate the metrics. This will be done for both the coupling and effort metrics.

The empirical validation of the metrics will be carried out by the means of case studies. For instance, the predicted effort corresponds to the real required effort, within a certain error window.

4 Research method

The main research method that is going to be employed during this project is *Technical Action Research* (TAR) [27].

This research method is artifact-based, which means that the first step is to produce the artifact meant to be used in certain situations envisioned by the researcher. The testing of this artifact, to see if it is effective in these situations, is done through a number of iterations. First, under ideal conditions, and then changing the experiments step by step to reach a real-world situation. These iterations, in the context of this project, are going to be limited due to the existing deadlines. However, there is the option of continuing with this part of the work in the future.

In addition, the research will also include controlled experiments. These experiments will be conducted as the empirical part of the validation of the coupling metrics. Additionally, the empirical validation of the effort estimation is going to be achieved by the means of case studies. In these case studies, the estimated effort will be compared with the real effort that the change required.

Some of the most difficult parts of this research are the validation and adaptation of the coupling metrics.

First, the validation since there is not a unique way to do it, but rather a wide variety of criteria and approaches. Therefore, conducting a complete validation is a complex task as it is not possible to validate every aspect of the metric.

In addition, even though the metrics that are going to be used to measure coupling between the projects and the libraries already exist, they are going to be adapted. These changes of the metrics could be complex for some of them. However, there are many metrics that could be used, and the ones that can not be adapted could serve as an indication of the types of dependencies that could exist when adapting other metrics.

After the formal definition and theoretical validation of the metrics for both, the coupling measurement and the estimation effort, to replace a library, a Proof-of-Concept (PoC) will be made. With this PoC, the metrics will be calculated for the projects given to the PoC. Once the PoC is ready, the empirical validation will be carried out for each of the metrics, by conducting controlled experiments and case studies as elaborated above.

The methodology for validating the metrics chosen during this research will be divided into two phases: a first phase of theoretical validation and a second one of empirical validation [25]. These validations will be carried out during the phases *Creation of the theoretical model* and *Evaluation of the project* respectively (see Section 7).

Literature survey During the literature survey for this proposal we mainly utilized *Google Scholar*. The resources found were documented with the following structure: First, by the overarching area of research, in this case: Dependency management, dependency/coupling metrics, effort estimation, and metrics validation.

As for the metrics validation category, we have conducted a brief excerpt of the main points or useful information extracted from each of the papers. Finally, the papers of coupling metrics are used to extract the characteristics of each of the proposed metrics to filter out the non-applicable metrics (see Table 3). Besides, the effort estimation papers were used as a survey of the major models in order to compare them.

See section 9 for more detail.

Once the literature survey is finalized, we expect the following input:

- A clear overview of the existing coupling metrics, as well as a comparison between them, in order to have the necessary knowledge to determine if a metric is applicable to the situation considered in this project, and how could it be adapted.
- The different existing models to estimate effort. An overview of the necessary steps and the data to estimate effort and the differences between the models.
- The various approaches to validate metrics, and the different characteristics of the methodologies. Also, the awareness of which methodologies and criteria could be applied to the proposed metrics.
- The existing options to create a call-level dependency graph for the PoC.

Timeline The timeline of this project will be divided in five main parts. A description of each one of these parts can be found in Section 7, as well as a timeline in the form of a Gantt diagram, in Figure 2.

5 Expected results of the project

Currently, there are three expected results of this project:

Theoretical description of the set of metrics: The first result is the description of a set of metrics to be used to answer the research questions. These are the coupling metrics to calculate the degree of dependency between a project and its libraries, as well as the estimation of effort needed to replace the libraries. Each metric will include a formal definition as well as a set of validation criteria that these metrics will have to fulfill to be considered valid.

Implementation of the proof-of-concept (PoC): The second result will be the software implementation of the previously created theoretical model. This PoC will be meant to be used as a prototype to calculate the metrics proposed, to use the results for the evaluation and validation of the metrics.

Validation of the metrics: Once the PoC is ready, the validation of the metrics will be conducted. There will be experiments for the coupling metrics, as well as case studies for the effort estimation. The third result is expected to be a report with the conclusions of the validation, the theoretical and the empirical ones.

6 Required expertise

In this section, the required expertise to carry out this project is discussed. In Table 4, the necessary skills are listed. Each skill is associated with a *Required proficiency*, which represents the level of knowledge required for the project and the *Current proficiency*. These levels of knowledge are assigned according to the following proficiency scale:

1. Fundamental
2. Basic
3. Intermediate
4. Advanced
5. Expert

7 Timeline

This project will be divided into several phases:

1. **Learning:** This initial phase consists of acquiring the necessary knowledge to obtain the level of competence required to carry out the project (see section 6).

Skill	Required proficiency	Current proficiency
Java programming	Expert	Intermediate
Static code analysis	Advanced	Basic
Software metrics consideration and design	Intermediate - Advanced	Basic
Software metrics validation	Intermediate - Advanced	Basic
Open-source implications	Intermediate	Intermediate
Software architecture design	Advanced	Intermediate

Table 4: Necessary skills with required and current proficiency

2. **Creation of the theoretical model:** In this phase, we will define the metrics to be used to calculate the coupling of a project with the libraries it uses. Also, the model to estimate the effort to replace a library will be defined.

   In addition, the validation criteria that are going to be used for each of the metrics will be described, and the theoretical validation of these metrics will be carried out.

3. **Development of the Proof-of-Concept (PoC):** This third phase consists of implementing the PoC to calculate the coupling metrics and to implement the effort estimation model.

4. **Evaluation of the project:** In this phase, the PoC will be used in real projects. The results obtained for each one of the metrics, as well as the model of effort estimation, will serve as the data to conduct the empirical validation. This validation will be done by means of controlled experiments and case studies.

5. **Thesis writing:** This phase consists of writing down the process of each of the previous phases for the creation of the documentation of the project.

It should be noted that the last phase: *Thesis writing*, will not be carried out in a sequential manner with the other phases, but will be done in parallel with the rest of the phases.

During phases 2, 3, and 4, the three expected results of the project will be developed (see section 5).

8 Risks

Project goals are too ambitious. If the project goals are underestimated, it could cause a problem in terms of project completion. To minimize the possibility of this risk becoming reality, it is critical to clearly specify the goals of the project. To be able to react if the risk is not prevented, some extra time has been not included in the planned timeline. First, more hours could be invested as overtime and weeks 34 and 35 can also be added to the project, while the final deadline would still not be missed.

Figure 2: Gantt Chart

Project goals are too vague. This risk is also related to the definition of the project objectives. As with the previous risk, to prevent it, it is important to dedicate the necessary time and effort to specify the objectives. However, if this situation is detected, the development of the project will be stopped to reassess and specify the objectives in more detail. In this way, it will be avoided that this risk leads to major problems in later stages of the project.

The information about the project is unavailable/incomplete/too difficult. In order for this risk not to lead to severe problems during the project, it is important to gather information on the topic as soon as possible. If it is noticed that there is not enough information or the project is overly complex, efforts will be focused on the part of the project that is most feasible. In this way, the project would be continued; although the result would not be as complete as planned, it would serve as a starting point for future continuation.

The gap between the current expertise and the required expertise is too large. To mitigate the possibility that this risk becomes reality and results in a grave problem for the holistic completion of the project, during the first phase of the project, time has been allocated for knowledge gathering. This is meant to reduce the gap between the required and current knowledge. In the same way, it could be detected that it is necessary to invest more hours in the training phase. These hours would be spent in the form of overtime, which would not affect the estimated date of completion of the project.

Adaptation and justification of the coupling metrics. During the second phase of the project, the coupling metrics are going to be adapted. One of the risks associated with this task is that the metrics may not be flexible enough to be used in a different context. To prevent this, we selected several metrics that could be used. Therefore, if one of them is not feasible to adapt, we can use the rest. However, if this is the case, the characteristics of the discarded metric and the type of coupling that it measures would be considered when adapting another metric.

Adaptation of the effort to change code calculation. This risk is similar to the previous risk. In this case, we would have to consider an alternative model to estimate the effort to implement in the context studied in this project. To do so, it is important to consult the support from experts in the field of research, as well as the first phase of the project. In this initial phase, it is expected for us to acquire the necessary knowledge about the model to be able to assess if the adaptation is feasible.

Validation of the metrics. Since there is no unique and standardized way to validate metrics, this task entails a risk for the project. The validation will be performed for certain properties and validation criteria for each of the used metrics. However, the time limit does not allow for all of the metrics to be validated using each of the criteria, or conduct too many experiments. Therefore, once the project is finalized, we could affirm that the metrics have been validated, but it will not be a complete validation.

9 Literature survey

For the creation of this proposal, we have conducted research on various topics, listed below. For each of these research activities, the tool that has been mainly used is *Google Scholar*.

- Dependency management
- Coupling metrics
- Code change effort measurement
- Research methods

The papers to be used for this document were selected according to their language (English) and accessibility. Of all the performed searches, we first conducted a selection according to titles and a second filter after reading the abstract and introduction of the papers.

The papers that according to these filters are related and relevant to the topic of this research were completely read and used to create this proposal. For each of the selected papers, the references of the paper were reviewed following the same procedure.

Dependency management This topic has been researched from various facets, one of the most widespread being the vulnerability of libraries. Examples of research on vulnerabilities in libraries and how these are expanded through the projects that utilize them, are [7, 10, 20, 21].

Another aspect of the dependencies between projects that have also been studied are the dynamics of usage between packages and versions of these. For example, Wittern et al. in [28] analyze the dynamics of package use in *npm* and the adoption of new versions of these by the developers. Plus, Mileva et al. analyzed the popularity of library versions, as well as the adoption process of them [19].

Ultimately, there is research related to modeling software ecosystems, such as [6, 10, 15]. However, most of these models are created at a project or project version level.

There are no studies, that we have been capable to find, analyzing the degree of dependence of a project on the libraries that it uses. This fact was perceived by Hejderup et al in [12].

Coupling metrics To obtain the papers describing the coupling metrics to be used in this project, we initially found the papers about measuring dependency (e.g. [5, 22]). Based on these papers, the research was focused on structural coupling metrics, which lead to finding the literature review papers and the definition of the metrics (e.g. [3, 4, 9]).

Finally, we conducted a literature survey focused on how to validate the metrics. Resulting in papers about the diverse approaches and properties of the metrics to take into account when conducting a validation. For instance, [18, 23, 25].

Code change effort measurement To gather information about the techniques to estimate effort, the first step was to research about the general topic effort estimation, yielding papers such as [24]. Next, the research focused on estimating effort to change code, providing as principal result the research by Kama et al. in [13, 14].

Dependency graphs To create the PoC, it is necessary to implement a dependency graph that permits the calculation of the metrics. The first results obtained during the literature survey about dependency graphs (e.g. [15, 17]) used package-based approaches. Therefore, we limited the search to call-based approaches. The two main results of this final search have been [11, 12].

Research methods In this case, the first paper used to define the research method of the project, was given by the university. In [8] a description of each of the main research methods can be found. Based on this paper, the research was focused on papers about the research methods selected to be used during this project (e.g. [27]).

References

[1] Edward B Allen and Taghi M Khoshgoftaar. Measuring coupling and cohesion: An information-theory approach. In *Proceedings Sixth International Software Metrics Symposium (Cat. No. PR00403)*, pages 119–127. IEEE, 1999.

[2] Mehran Halimi Asl and Nazri Kama. A change impact size estimation approach during the software development. In *2013 22nd Australian Software Engineering Conference*, pages 68–77. IEEE, 2013.

[3] Lionel Briand, Prem Devanbu, and Walcelio Melo. An investigation into coupling measures for c++. In *Proceedings of the 19th international conference on Software engineering*, pages 412–421, 1997.

[4] Lionel C. Briand, John W. Daly, and Jurgen K Wust. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on software Engineering*, 25(1):91–121, 1999.

[5] Marcelo Cataldo, Audris Mockus, Jeffrey A Roberts, and James D Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 35(6):864–878, 2009.

[6] Alexandre Decan, Tom Mens, and Maëlick Claes. An empirical comparison of dependency issues in oss packaging ecosystems. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 2–12. IEEE, 2017.

[7] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 181–191. IEEE, 2018.

[8] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[9] Rachel Harrison, Steve Counsell, and Reuben Nithi. Coupling metrics for object-oriented design. In *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No. 98TB100262)*, pages 150–157. IEEE, 1998.

[10] JI Hejderup. In dependencies we trust: How vulnerable are dependencies in software modules?, 2015.

[11] Joseph Hejderup, Moritz Beller, and Georgios Gousios. Prazi: From package-based to precise call-based dependency network analyses, 2018.

[12] Joseph Hejderup, Arie van Deursen, and Georgios Gousios. Software ecosystem call graph for dependency management. In *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, pages 101–104. IEEE, 2018.

[13] Nazri Kama. Integrated change impact analysis approach for the software development phase. *International Journal of Software Engineering and its Applications*, 7(2):293–304, 2013.

[14] Nazri Kama, Sufyan Basri, Mehran Halimi Asl, and Roslina Ibrahim. Cochcomo: A change effort estimation tool for software development phase. In *SoMeT*, pages 1029–1045, 2014.

[15] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. Structure and evolution of package dependency networks. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 102–112. IEEE press, 2017.

[16] Raula Gaikovina Kula, Coen De Roover, Daniel German, Takashi Ishio, and Katsuro Inoue. Visualizing the evolution of systems and their library dependencies. In *2014 Second IEEE Working Conference on Software Visualization*, pages 127–136. IEEE, 2014.

[17] Raula Gaikovina Kula, Coen De Roover, Daniel M German, Takashi Ishio, and Katsuro Inoue. Modeling library dependencies and updates in large software repository universes. *arXiv preprint arXiv:1709.04626*, 2017.

[18] Andrew Meneely, Ben Smith, and Laurie Williams. Validating software metrics: A spectrum of philosophies. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(4):1–28, 2013.

[19] Yana Momchilova Mileva, Valentin Dallmeier, Martin Burger, and Andreas Zeller. Mining trends of library usage. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pages 57–62. ACM, 2009.

[20] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. Vulnerable open source dependencies: Counting those that matter. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 42. ACM, 2018.

[21] Henrik Plate, Serena Elisa Ponta, and Antonino Sabetta. Impact assessment for vulnerabilities in open-source software libraries. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 411–420. IEEE, 2015.

[22] Denys Poshyvanyk and Andrian Marcus. The conceptual coupling metrics for object-oriented systems. In *2006 22nd IEEE International Conference on Software Maintenance*, pages 469–478. IEEE, 2006.

[23] John M Roche. Software metrics and measurement principles. *ACM SIGSOFT Software Engineering Notes*, 19(1):77–85, 1994.

[24] TN Sharma. Analysis of software cost estimation using cocomo ii. *International Journal of Scientific & Engineering Research*, 2(6):1–5, 2011.

[25] KP Srinivasan and T Devi. Software metrics validation methodologies in software engineering. *International Journal of Software Engineering & Applications*, 5(6):87, 2014.

[26] Common Vulnerabilities and Exposures. CVE - Frequently Asked Questions.

[27] Roel Wieringa and Ayşe Moralı. Technical action research as a validation method in information systems design science. In *International Conference on Design Science Research in Information Systems*, pages 220–238. Springer, 2012.

[28] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. A look at the dynamics of the javascript package ecosystem. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 351–361. IEEE, 2016.

[29] Yong Yu, Tong Li, Qing Liu, and Qian Yu. Measurement of software coupling based on structure entropy. In *International Conference on Intelligent Computing and Information Science*, pages 435–439. Springer, 2011.

[30] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005.