A Literature Survey of Software Analytics

IN4334 2018 TU Delft

2018-10-16

Contents

Chapter 1

Preamble

The book you see in front of you is the outcome of an eight week seminar run by the Software Engineering Research Group (SERG) at TU Delft. We have split up the novel area of Software Analytics into several sub topics. Every chapter addresses one such sub-topic of Software Analytics and is the outcome of a systematic literature review a laborious team of 3-4 students performed.

With this book, we hope to structure the new field of Software Analytics and show how it is related to many long existing research fields.

The IN4334 – Software Analytics class of 2018

1.1 License

Chapter 2

A contemporary view on Software Analytics

2.1 What is Software Analytics?

2.2 A list of Software Analytics Sub-Topics

Chapter 3

Testing Analytics

3.1 Motivation

Testing is an important aspect in software engineering, as it forms the first line of defence against the introduction of software faults Pinto et al. [**?**]. However, in practice it seems that not all developers test actively. In this chapter we will survey on the use of testing and the tools that make this possible. We will also look into the future development of tools that is done or required in order to improve testing practices in real-world applications. Testing is not the holy grail for completely removing all bugs from a program but it can decrease the chances for a user to encounter a bug. We believe that extra research is needed to ease the life of developers by making testing more efficient, easier to maintain and more effective. Therefore, we wanted to write a survey on the testing behavior, current practices and future developments of testing. In order to perform our survey, we formulated three Research Questions (RQs):

- **RQ1** How do developers currently test?
- **RQ2** What state of the art technologies are being used?
- **RQ3** What future developments can be expected?

In this chapter we will first elaborate on the research protocol that was used in order to find papers and extract information for the survey. Second, the actual findings for each of the research questions will be explained.

3.2 Research Protocol

For this chapter, Kitchenham's survey method [**?**] was applied. For this method, a protocol has to be specified. This protocol is defined for the research questions given above. Below the inclusion and exclusion criteria are given, which helped finding the rightful papers. After these criteria, the actual search for papers is described. The papers that were found are listed and after they are tested against the criteria that are given. The data that is extracted from these papers are list afterward. Some papers that were left out will be listed and the reasons for leaving them out will be given to make clear why some papers do not meet the required desire.

Each of the papers found was tested using our inclusion and exclusion criteria. These criteria were introduced to make sure the papers have the information required to answer the RQs while also being relevant with respect to their quality and age. Below a list of inclusion and exclusion criteria is given. In general, for all criteria, the exclusion criteria take precedence over inclusion criteria. The following inclusion and exclusion criteria were used:

- Papers published before 2008 are excluded from the research, unless a reference/citation is used for an unchanged concept.
- Papers referring to less than 15 other papers, excluding self-references, are excluded from the research.
- Selected papers should have an abstract, introduction and conclusion section.
- Papers stating the developers' testing behavior are included.
- Papers stating the developers' problems related to testing are included.
- Papers stating the technologies, related to testing analytics, which developers use are included.
- Papers writing about the expected advantage of current findings in testing analytics are included.
- Papers with recommendations for future development in the software testing field are included.

The papers used in this chapter were found by using a given initial seed of papers (query defined below as 'Initial Paper Seed'). From this initial seed of papers we used the keywords used by those papers to construct queries. Additionally, the references ('referenced by') and the citations ('cited in') of the papers were used to find papers. The query row of the tables describing the references, as found below, indicates how a paper was found. For queries the default search sites were Scopus,[1] Google Scholar[2] and Springer.[3]

The keywords used to construct queries in order to find papers were: software, test*, analytics, test-suite, evolution, software development, computer science, software engineering, risk-driven, survey software testing

The table below describes for each paper, which Query resulted in which paper being found. Each of the papers is categorized with a corresponding research question. In the table below, the categories per paper were added based on their general topic. These broad topics will be assigned to a corresponding research question. Categorizations are based on the bullet points extracted from each paper. These bullet points can be found in the appendix of this chapter in section *'Extracted paper information'*.

Category	Reference	Query	Relevant to
Co-evolution	Greiler et al. [57]	In 'cited by' of "Understanding myths and realities of test-suite evolution" on Scopus	RQ2, RQ3
Co-evolution	Hurdugaci and Zaidman [65]	Keywords: Maintain developer tests, 'cited by' in "Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining" on IEEE	RQ2
Co-evolution	Marsavina et al. [86]	Google Scholar keywords: Maintain developer tests, in 'cited by' of "Aiding Software Developers to Maintain Developer Tests" on IEEE	RQ1
Co-evolution	Zaidman et al. [124]	Initial Paper Seed	RQ1
Production evolution	Eick et al. [51]	Referenced by: [80]	Discarded
Production evolution	Leung and Lui [80]	Initial Paper Seed	RQ3
Risk-driven testing	Atifi et al. [6]	In 'cited by' of "Risk-driven software testing and reliability"	RQ2, RQ3
Risk-driven testing	Hemmati and Sharifi [62]	In 'cited by' of "Test case analytics: Mining test case traces to improve risk-driven testing"	RQ3
Risk-driven testing	Noor and Hemmati [95]	Initial Paper Seed	RQ2, RQ3
Risk-driven testing	Schneidewind [109]	Scopus query: risk-driven testing	RQ3
Risk-driven testing	Vernotte et al. [121]	Scopus query: "risk-driven" AND testing	RQ2, RQ3

[1] https://www.scopus.com/
[2] https://scholar.google.com/
[3] https://www.springer.com

Category	Reference	Query	Relevant to
Test evolution	Bevan et al. [21]	Referenced by: [98]	Discarded
Test evolution	Mirzaaghaei et al. [92]	Google Scholar query: test-suite evolution	RQ2, RQ3
Test evolution	Pinto et al. [98]	Initial Paper Seed	RQ1
Test evolution	Pinto et al. [97]	Referenced by: [98]	RQ1
Test generation	Bowring and Hegler [29]	Springer: Reverse search on "Automatically generating maintainable regression unit tests for programs"	RQ2, RQ3
Test generation	Dulz [47]	Scopus query: "software development" AND Computer Science AND Software Engineering	RQ2
Test generation	Robinson et al. [105]	Referenced by [92]	RQ2
Test generation	Shamshiri et al. [110]	Google Scholar query: Automatically generating unit tests	RQ3
Testing practices	Beller et al. [20]	In 'cited by' of "Understanding myths and realities of test-suite evolution".	RQ1
Testing practices	Beller et al. [16]	Initial Paper Seed	RQ1
Testing practices	Garousi and Zhi [55]	Google Scholar query: Survey software testing	RQ1
Testing practices	Moiz [93]	Springer query: software testing	RQ3

3.3 Results

In this section the research questions will be answered. To answer these questions, information from the relevant papers are aggregated. The answers to each research questions are summarized in the conclusion.

3.3.1 (RQ1) How do developers currently test?

To answer RQ1, "How do developers currently test?", we first outline general test practices, then discuss the co-evolution of test and production code and finally, look into the use of Test Driven Development among developers.

3.3.1.1 How do we test?

For the quality of code, test coverage is a popular metric. Alternatives are, for example, acceptance tests, the number of defects in the last week, or defects per Line of Code (LOC) [55]. However, code coverage might not be the best indicator for the extensiveness of testing. For example, according to Beller et al. [17] a code coverage of 75% can possibly be reached with only spending less than a tenth of the total development time on testing. Another concern of using test coverage as a metric is the concept of treating the metric [28] , where developers try to uplift the value of code coverage by hitting many lines with only a few test cases. Marsavina et al. [86] observed that test cases were rarely updated when changes related to attributes or methods in the production code were made. Possible explanations for this are that these changes were not significant or the tests were too simple and were likely to pass. This also fits with the findings of Romano et al. [107], where they claim that "[d]evelopers write quick-and-dirty production code to pass the tests, do not update their tests often, and ignore refactoring."

Besides older tests rarely being updated for changed code, even new tests do not necessarily have the purpose of validating new production code lines. Pinto et al. [98] observed that a significant number of new tests

that are added, were not necessarily added to cover new code but rather to exercise the changed parts of the code after the program is modified. This finding fits with the observation of Marsavina et al. [86], who found that test cases are created or deleted in order to address the modified branches whenever numerous condition related changes are conducted in the production code base. Older production code lines, therefore, may stay untouched by any test cases. Lines uncovered by any traditional code coverage tool should be indicated and signaled to the developer. Therefore, developers should be aware of the fact that they did not cover some lines of their production code with any tests. It seems to be a deliberate action by most developers to not cover older production code lines. These lines might be 'too hard' to test, other lines may be easier to test, or developers do not seem to see the relevance of testing these uncovered lines of code. However, the most commonly used coverage metrics are branch coverage and conditional coverage [55]. As both branch coverage and conditional coverage require multiple different conditions for if-statements, it may possibly be that the absolute number of missed lines of production code by tests is very low but rather the number of missed conditions is higher.

3.3.1.2 Co-evolution

In a case study conducted by Zaidman et al. [124], there was no evidence found for an increased activity of testing before a release.

However, the study detected periods of increased test writing activity. These increased activities of writing test cases were found to be after longer periods of writing production code [124]. With a longer timespan of not writing tests, it can be concluded for these cases that the production code and test code do not gracefully co-evolve [124] [86].

3.3.1.3 Test-Driven Development (TDD)

We found different definitions for TDD across multiple studies. According to Zaidman et al. [124], evidence of TDD was found where test code was committed alongside production code, meaning that the methodology of TDD is used when production code was written before the respective test code. This is in contrast with the originally proposed constraint by Beck [14], where a line of production code should only be written after a failing automated test was written in advance. The confusion for the definition of TDD can also be traced back by the finding of Beller et al. [20], where programmers who claim they practice TDD neither follow it strictly nor practice it for all of their modification. A survey conveyed by Garousi and Zhi [55] on 196 respondents (amongst them managers and developers) indicated that with a ratio of 3:1 use Test-last development and Test-driven development respectively. This found ratio is in contrast with the numbers found by Beller et al. [20]; only 1.7% of the observed developers seemed to follow the strict TDD definition, where most of these developers only practice this strict definition in less than 20% of their time. However, it is important to mention that the survey done by Garousi and Zhi [55] only surveyed the subjects, which allows the confusion for the definition of TDD to play a major role in the results found.

3.3.2 (RQ2) What state of the art technologies are being used?

We will cover two research fields regarding testing analytics: test evolution and generation, and risk-driven testing.

3.3.2.1 Test Evolution and Generation

Pinto et al. [98] found the investigation of automated test repairing is not a promising research avenue, as these techniques would require manual guidance which could end up being similar to traditional refactoring tools. Nonetheless, more research is performed in this field since then. An approach for automatically repairing and generating test cases during software evolution is proposed by Mirzaaghaei et al. [92]. This

approach uses information available in existing test cases, defines a set of heuristics to repair test cases invalidated by changes in the software, and generate new test cases for evolved software. This properly repairs 90% of the compilation errors addressed and covers the same amount of instructions. The results show that the approach can effectively maintain evolving test suites and perform well compared to competing approaches.

While full automated test suite generation can not replace human testing entirely yet, Bowring and Hegler [29] introduced a tool that generates the templates for tests, which guarantees compilation, supports exception handling and finds a suitable location for the test. Developers still need to fix the test oracles themselves, but the template is there. The technique looks at the context in order to decide what template to use. Robinson et al. [105] created a regression unit tests generation tool. It is a suite of techniques for enhancing an existing unit test generation system. The authors performed experiments using an industrial system. The generated tests from these experiments achieved good coverage and mutation kill score, were readable by the product developers and required few edits as the system under test evolved. Dulz [47] found that by directly adjusting specific probability values in the usage profile of a Markov chain usage model, it is relatively easy to generate abstract test suites for different user classes and test purposes in an automated approach. By using proper tools, such as the TestUS Testplayer, even less experienced test engineers will be able to efficiently generate abstract test cases and to graphically assess quality characteristics of different test suites. Hurdugaci and Zaidman [65] introduces TestNForce (Visual Studio only), a tool to help developers identify unit tests that need to be altered and executed after code change.

3.3.2.2 Risk-driven Testing

The paper by Vernotte et al. [121] introduces and reports on an original tool-supported, risk-driven security testing process called Pattern-driven and Model-based Vulnerability Testing. This fully automated testing process, relying on risk-driven strategies and Model-Based Testing (MBT) techniques, aims to improve the capability of detection of various Web application vulnerabilities, in particular SQL injections, Cross-Site Scripting, and Cross-Site Request Forgery. An empirical evaluation shows that this novel process is appropriate for automatically generating and executing risk-driven vulnerability test cases and is promising to be deployed for large-scale Web applications.

A new risk measure is defined by Noor and Hemmati [95], which assigns a risk factor to a test case if it is similar to a failing test case from history. The new risk measure is by far more effective in identifying failing test cases compared to the traditional risk measure. Using this method for identifying test cases with a high risk factor, these test cases can for example be ran in the background while developing code, to find faults earlier. Furthermore, prioritizing these tests while running the entire test-suite could make the suite detect failing tests earlier and the developer can start fixing the faulty code right away.

3.3.3 (RQ3) What Future Developments Can Be Expected?

This section will elaborate on which future developments can be expected in the field of software analytics.

3.3.3.1 Co-Evolution and Test Generation

For understanding how test- and production code co-evolve and how tests can be generated to support developers, studies have been conducted [86, 98, 124]. Additionally a tool has been made in order to analyze and, consequently, better understand test-suite evolution [97]. For the time being the practical implications of this subtopic have mainly been sought in the repairing and generation of tests.

According to Pinto et al. [98] test repairs occur often enough to justify the development and research for automated repair techniques. M. Mirzaaghaei et al. [92] argue that evolving test cases is an expensive and time-consuming activity, for which automated approaches reduce the pressure on developers. Shamshiri et al. [110] argue that automated generation of unit tests does not end up generating realistic tests and that

the effectivity of developers writing manual tests is equal to developers using automatically generated tests. Therefore, they call for the use of more realistic tests. This suggests that automated test generation is still a topic of future interest, which will likely be researched in order to find a way to generate realistic tests.

3.3.3.2 Risk-driven Testing

Risk-driven testing is an area of recent attention. Researchers have been looking for methods that can either detect potential risks within the same project [95] [62] [121] or that can detect risks based on models carried over from one project to another [80] [6]. These techniques have been implementing history based prediction approaches.

In the future, we can expect more interest and research into risk-driven testing as allocating testing activities effectively will remain important due to testing efforts and developer time being expensive. This area will likely stay in its research phase for the next couple of years as effective measures for risk prediction are still being researched. This goes for measures within the same project and cross-project prediction. Given that the currently researched techniques regard history based implementations, it is likely that these techniques will be subject to further research later on.

3.3.3.3 Testing Practices

Research of several papers [55] [16] [20] has indicated that testing of any form is not as widely practiced as the status quo suggests. How the current state of the practice will change depends on various developments within the field. Tools will be created to assist the developer in writing quality code and tests, such as TestEvoHound as suggested by M. Greiler. [57]. As automated test generation becomes more effective this may reduce the need for developers to spend a lot of time on writing and maintaining tests. With the development of risk-driven testing, developers may also be able to focus on the parts that are likely to be the most important to address, which could lead to better time allocation. The status quo for how much time is to be expected to be spent on testing may also change, given automated test repair and generation techniques become effective and accessible.

3.4 Conclusion

In this chapter, three different research questions about software testing analytics were answered. (RQ1) How do developers currently test? (RQ2) What state of the art technologies are being used? (RQ3) What future developments can be expected?

Regarding the current testing practices of developers (RQ1), we found that developers do not seem to update their tests very often and when they do, it is because of a changed condition in production code lines. Furthermore, older uncovered production code lines are not likely to be covered in the end. Developers, thus, seem to ignore indications of their code coverage tools or do not seem to use any code coverage tool at all. Furthermore, developers do not seem to put a lot of effort into making sure the co-evolution of their production- and test code is done gracefully. They do, on the other hand, make sure their test code compiles when production code classes have been removed. However, testing is mostly done in longer periods of increased testing. The methodology of TDD also seems to be a confusing term for developers, as there is not enough clear guidance in the implementation of it. The actual ratio of TLD and TDD is, therefore, unknown but can be guessed with great certainty to be much lower for TDD than for TDD.

The current state of the art in testing analytics (RQ2) consists of research in co-evolution and generation of tests, and risk-driven testing. Approaches are proposed for automatically repairing and generating test cases during software evolution. While fully automated test suite generation is not there yet, a tool is introduced that generates the templates for tests, which guarantees compilation, supports exception handling and finds a suitable location for the test. In the field of risk-driven testing, new risk measures are defined which make

prioritizing certain high-risk tests able while running the entire test-suite, which could make the suite detect failing tests earlier.

For future developments (RQ3), further research can be expected on the front of automated test generation. Even with some discussion regarding the effectiveness of test generation, the field currently agrees that conducting research in order to find, especially, realistic ways of generating tests is worthwhile. We also found that risk-driven testing has been given more attention in the form of research recently. This subtopic is still in its research phase. It can be expected that research on the front of history based risk prediction methods will continue.

Chapter 4

Build analytics

4.1 Motivation

Ideally, when building a project from source code to executable, the process should be fast and without any errors. Unfortunately, this is not always the case and automated builds results notify developers of compile errors, missing dependencies, broken functionality and many other problems. This chapter is aimed to give an overview of the effort made in build analytics field and Continuous Integration (CI) as an increasingly common development practice in many projects.

4.2 Research Questions

- **RQ1** What is the current state of the art in the field of build analytics?
- **RQ2** What is the current state of practice in the field of build analytics?
- **RQ3** What future research can we expect in the field of build analytics?

4.3 Research protocol

Using the initial seed consisting of [22], [18], [103], [19], [96], [127], [122] and [63] we used references to find new papers to analyze. Moreover, we used academical search engines like *GoogleScholar* to perform a keyword based search for other relevant build analytics domain papers. The keywords used were: build analytics, machine learning, build time, prediction, continuous integration, build failures, active learning, build errors, mining, software repositories, open-source software.

4.4 Answers

Through this we found the following papers

4.5 Summary of papers

4.5.1 [22]

Initial Seed

This is a US patent grant for a method of predicting software build errors. This patent is owned by Microsoft. Using logistic regression a prediction can be made on the probability of a build failing. Using this method build errors can be better anticipated, which decreases the time until the build works again.

4.5.2 [18]

Initial Seed

This paper explores data from Travis CI[1] on a large scale by analyzing 2,640,825 build logs of Java and Ruby builds. It uses TravisTorrent as a data source. It is found that the number one reason for failing builds it test failure. It also explores differences in testing between Java and Ruby.

4.5.3 [103]

Initial Seed

A stuy on the build results of 14 open source software Java projects. It is similar to [18], albeit on a smaller scale. It does go more in depth on the result and changes over time.

4.5.4 [19]

Initial Seed

This paper introduces TravisTorrent, a dataset containing analyzed builds from more than 1,000 projects. This data is freely downloadable from the internet. It uses GHTorrent to link the information from travis to commits on GitHub.

4.5.5 [96]

Initial Seed

This paper is a survey amongst Travis CI users. It found that users are not sure whether a job failure represents a failure or not, that inadequate testing is the most common (technical) reason for build breakage and that people feel that there is a false sense of confidence when blindly trusing tests.

4.5.6 [127]

Initial Seed

This paper analyzed approximately 160,000 projects written in seven different programming languages. It notes that adoption of CI is often part of a reorganization. It collected information on the differences before and after adoption of CI. There is also a survey amongst developers to learn about their experiences in adopting Travis CI.

4.5.7 [122]

Initial Seed

This paper analyzes what factors have impact on abandonment of Travis. They find that increased build complexity reduces the chance of abandonment, but larger projects abandon at a higher rate and that a

[1] See https://travis-ci.org

project's language has significant but varying effect. A surprising result is that metrics of configuration attempts and knowledge dispersion in the project do not affect the rate of abandonment.

4.5.8 [63]

Initial Seed

This paper explores which CI system developers use, how developers use CI and why developers use CI. For this it analyzes data from Github, Travis CI and it conducts a developer survey. It finds that projects using CI release twice as often, accept pull requests faster and have developers who are less worried about breaking the build.

4.5.9 [120]

References [18]

This paper discusses the difference in failures on continuous integration between open source software (OSS) and industrial software projects. For this 349 Java OSS projects and 418 project from ING Nederland, a financial organization.

Using cluser analysis it was observed that both kinds of projects share similar build failures, but in other cases very different patterns emerge.

4.5.10 [60]

References [19]

This paper uses TravisTorrent ([19]) to show that 22% of code commits include changes in build script files to keep the build working or to fix the build.

In the paper a tool is proposed to automatically fix build failures based on previous changes.

4.5.11 [119]

References [18], [103]

This paper proposes a tool called BART to help developers fix build errors. This tool eliminates the need to browse error logs which can be very long by generating a summary of the failure with useful information.

4.5.12 [125]

Referenced by [119]

This paper studies the usage of static analysis tools in 20 Java open source software projects hosted on GitHub and using Travic CI as continuous integration infrastructure. There is investigated which tools are being used, what types of issues make the build fail or raise warnings and how is responded to broken builds.

4.5.13 [8]

Google Scholar search term `Github "Continuous Integration"`, papers from 2018

This paper analyses 93 GitHub projects before and after adoption of Travis CI. It finds only one non-negligible effect, an increasing merge ratio, meaning that more merging commits in relation to all commits

after a project started using Travis CI. But the paper also shows that this effect can be seen on projects not adopting CI. It shows the importance of having a proper dataset with as little bias as possible.

4.6 What is the current state of the art in the field of build analytics?

The current state-of-the-art in build analytics domain refers to the use of machine learning techniques to increase the productivity when using Continuos Integration (CI), to generate contraints on the configuration of the CI that could improve build success rate and to predict build failures even for newer projects with less training data available. Beside the papers from the initial seed, we will discuss the following state-of-the-art aproaches papers:

4.6.1 [24]

This paper aims to find a balance between the frequency of integration and developers productivity. They proposed models able to predict the build time of a job taking advantage of data from TravisTorrent. Their research is also slighty addressing the problem of optimal build time. Their method consists of selecting using different strategies to select the relevant features from the 56 features presented in TravisTorrent build records and applying a set of both linear and non-linear algorithm for predicting the time of a build. They evaluate the models performance using Root Mean Square Error (RMSE) and R-Squared and obtained for some models like Extreme-Gradient-Boosting(XGBOOST) a very high R-Squared around 80%, which shows that their model was able to capture the variation of build time over multiple projects. The main downfall of this paper is the testing size of only 10000 records of the 1,846,396 available data due to computational limits resulted probably from the usage of R machine learning packages, instead of python with TensorFlow. Their research could be useful on one hand for software developers and project managers for a better time management scheme and one the other hand for other researchers that may improve their proposed models.

4.6.2 [108]

The paper presents a tool VeriCI capable of checking the errors in CI configurations files before the developer push a commit and without needing to wait for the build result. Even if there are some other papers that achieve even higher accuracy in prediction of build failures, this paper is unique by not using metadata in the learning process like number of commits, code churn and so on. The authors rely on the actual user programs and configuration scripts, fact that make the identification of the error cause possible. Their approach consists of the following steps: give a formal description to the CI build process, extract the right code features and train self-explainable decision trees. VeriCI achieve 83% accuracy of predicting build failure on real data from GitHub projects and 30-48% of time the error justification provided by the tool matched the actual error cause. Even if VeriCI is capable of locate and give a reason for the expected failure, the false positive rate is quite high, therefore the authors proposed as a future work the analysis of the cost impact that a high rate of false positive has and also deploying the tool in large scale of CI enviroments.

4.6.3 [94]

This paper is posted only as a cover so far. It is the most recent paper of this survey, with the poster being published in June 2018. The paper addresses the problem of build failure prediction in CI environment for newer projects with less data available. It is using already trained models from other project with more data available and combined them by the means of active learning in order to find which of that models generalized better from the problem in hand and to update the models weights accordingly. It is also aimed to cut the expense that CI introduce by reducing the label data necessarily for training. Even if the method

seems promising, the results presented in the poster shows an F-Measure (harmonic average of recall and precision) of around 40% that could be better improved.

4.7 What is the current state of practice in the field of build analytics?

In this section, I will examine scientific papers to analyse the current trend of build analytics in the software development industry.

4.7.1 [53]

In this paper, Martin talks about the current state of the software industry in terms of Continuous Integration (CI) and comments on the practises required to implement CI effectively. He talks about his experience working for a large English electronics company where the development of a project took two years and the integration process took several months. Integration is a long and unpredictable process. Martin suggested this approach and that the two most common reactions he got were: "it can't work (here)" or "doing it won't make much difference". He expresses that most engineers don't know how simple the process can be of setting the CI framework up. In this way, we get a glimpse into the practises popular within the industry regarding build analytics.

4.7.2 [63]

This paper examines the usage, costs and benefits of Continuous Integration. A survey conducted in open-source projects indicated that 40% of all projects used CI. Of the projects that used CI, 90% used Travis for their CI services. They also determine that the more popular projects use CI but there is no correlation between the popularity of language and usage of CI. It also observes that the median project introduces CI a year into development. The paper claims that CI is widely used in practise nowadays and CI adoption rates will increase even further in the future.

4.7.3 [103]

Version Control Systems (VCS) such as GitHub, and hosted build automation platforms such as Travis, have made Continuous Integration is widely available for projects of every size. This paper suggests that CI is widely used and has improved the quality of processes and developed software itself. However, the article suggests that there is little known about the variety and frequency of errors that cause builds to fail. It suggests that developers should eliminate flaky tests and address common issues regularly such as broken interaction with repositories to keep the build system healthy.

4.7.4 [113]

This paper defines CI as a key element in agile software development and testing environment. It also uses Marin Fowler's practises of CI (as discussed previously) and expresses the importance of CI in the software industry.

4.8 What future research can we expect in the field of build analytics?

Future research in build analytics branches in a couple of different topics. [96] proposes to focus on getting a better understanding of the users and why they might choose to abandon an automatic build platform.

According to [8] future work could look into more perspectives when analyzing commit data, for instance partitioning commits by developer. It also notes the importance of more qualitative research.

Chapter 5

Bug Prediction

5.1 Motivation

Minimizing the number of bugs in software is an effort central to software engineering - faulty code fails to fullfill the purpose it was written for, its impact ranges from sligthly embarrassing to disastrous and dangerous, and last but not least - fixing it costs time and money. Resources in a software development lifecycle are almost always limited and therefore should be allocated to where they are needed most - in order to avoid bugs, they should be focused on the most fault-prone areas of the project. Being able to predict where such areas might be would allow more development and testing efforts to be allocated on the right places.

However, as noted in [50], reliably predicting which parts of source code are the most fault-prone is one of the holy-grails of software engineering. Thus it is not surprising that bug-prediction continues to garner a widespread research interest in software analytics, now equipped with the ever-expanding toolbox of data-mining and machine learning techniques. In this survey we investigate the current efforts in bug-prediction in the light of the advances in software analytics methods and focus our attention on answering the following research questions:

- **RQ1** What is the current state of the art in bug prediction? More specifically, we aim to answer the following:
  - What software or other metrics does bug prediction rely on and how good are they?
  - What kind prediction models are predominantly used?
  - How are bug prediction models and results validated and evaluated?
- **RQ2** What is the current state of practice in bug prediction?
  - Are bug prediction techniques applied in practice and if so, how?
  - Are the current developments in the field able to provide actionable tools for developers?
- **RQ3** What are some of the open challenges and directions for future research?

5.2 Research protocol

We started by studying the initial 6 seed papers which were selected based on domain knowledge:

- [58]
- [32]
- [5]
- [49]
- [59]
- [81]

Our searches were based on the following elements:

1. Keyword search using search engines (Scopus, ACM Digital Library, IEEE Explorer). The search query was constructed so that the paper title had to contain the phrase bug prediction, but also the other more general variants used in literature: *bug/defect/fault prediction*. The title also had to contain at least one of following keywords: *metrics*, *models*, *validation*, *evaluation*, *developers*. To remain within the bug prediction field we required *software* to appear in the abstract.
2. Filtering search results by publication date. We excluded papers older than 10 years; that is, published before 2008.
3. Filtering by the number of citations. We selected papers with 10 or more citations in order to focus on the ones that already have some visibility within the field.
4. Exploring other impactful publications by the same authors.

Table 1. Papers found by investigating the authors of other papers.

Starting point	Type	Result
[49]	is author of	[50]
[32]	is author of	[31] [33]

5.3 Answers

Chapter 6

Ecosystem Analytics

6.1 Motivation

In the modern day and age, the majority of software products make use of external software or libraries to use the functionality (for example parsing JSON) of these products, without having to develop this functionality itself. Moreover, multiple languages, such as Python and Rust, provide package managers (pip[1] and Cargo[2] respectively) which can be used to easily manage this third-party functionality, as well as distribute it.

In parallel to this, the popularity of creating open source projects is on the rise as well. On platforms such as GitHub[3], it is easy and quick to create a new software project, which can be developed, reviewed and used by the whole community. This development leads to more libraries being developed and being available for public use.

Because of these two developments, further inspection of the dependency relations between projects leads to a graph-like structure of software projects, where the nodes are the projects and the edges represent a dependency between two software projects. This structure is known as a *software ecosystem*. As stated by [91], a *software ecosystem* is "a collection of software products that have some given degree of symbiotic relationships." Another, similar definition is given by [82]: "A software ecosystem is a collection of software projects which are developed and co-evolve in the same environment." [90] extends this definition, "by explicitly considering the communities involved (e.g. user and developer communities) as being part of the software ecosystem." [112] opposes the overall notion of calling this structure a software ecosystem: "It is inadvisable to describe the free software community, or any human community, as an ecosystem, because that word implies the absence of ethical judgment."

Although [112] thinks that the term software ecosystem itself is incorrect, it does not necessarily disagree with the definition of the term. The definition which will be used in this chapter is the definition of [90], since it captures the essence of the other two definitions, while adding the notion of the human communities alongside as well.

By performing analysis on these software ecosystems, the aim is to generate meaningful insights. These insights can then be used to improve the efficiency and effectivity of the software development process, as well as to learn to identify and inform about potential problems. For example, a warning could be displayed if a dependency has a security vulnerability.

The field of research on software ecosystems, *ecosystem analytics*, focuses on performing such analysis. This chapter discovers what the current progress is in this field of research through a literature survey. This discovery is not limited to the theoretical perspective, but will uncover practical implications as well as the

[1] https://pypi.org/project/pip/
[2] https://crates.io/
[3] https://github.com:

open challenges of the field. In order to describe each covered aspect, we have formulated three research questions:

- **RQ1**: What is the current state of the art in software analytics for ecosystem analytics?
- **RQ2**: What are the practical implications from the state of the art?
- **RQ3**: What are the open challenges in ecosystem analytics, for which future research is required?

Each of these research questions will be answered using recent papers written in this field of research.

This chapter is structured as follows. First, the research protocol is described in detail. This includes decisions on which papers are included in the review. After this, the research questions are answered using the previously stated set of papers.

6.2 Research Protocol

In order to select literature to answer the research questions given in the previous section, the survey method suggested by [75] is used. This method creates a systematic way to select a set of papers, which is relevant to the research question(s).

The search strategy, as described by [75], are usually iterative and benefit from consultations with experts in the field, amongst other things. Our search strategy can be split in three different types:

- the initial seed, given by an expert in the field, MSc. Joseph Hejderup
- a search using a digital search engine, namely Google Scholar[4]
- a selection of referenced papers within papers selected before in the above two searches

In order to select literature to answer the research questions given in the previous section, the survey method suggested by [75] is used. This method creates a systematic way to select a set of papers, which is relevant to the research question(s).

The search strategy, as described by [75], are usually iterative and benefit from consultations with experts in the field, amongst other things. Our search strategy can be split in three different types:

- the initial seed, given by an expert in the field, MSc. Joseph Hejderup
- a search using a digital search engine, namely Google Scholar[5]
- a selection of referenced papers within papers selected before in the above two searches

6.2.1 Initial seed

MSc. Joseph Hejderup has provided us with a total of thirteen papers, as shown in Table 1.

As each of these papers come from an expert in the field, each paper is assumed to be relevant to atleast the field of software ecosystems. Because of this, each of these papers were judged on their relevance to either of the research questions. In Table 1, this relevance judgment is shown in the left column, since a paper is only selected, if the paper is indeed relevant. Table 2 describes the reason for which each particular paper is not selected for the literature survey.

Selected	Author(s)	Title	Year	Keywords
-	[2]	Strong dependencies between software components	2009	
-	[1]	Predicting upgrade failures using dependency analysis	2011	

[4] https://github.com:
[5] https://github.com:

Selected	Author(s)	Title	Year	Keywords
+	[3]	Why do developers use trivial packages? An empirical case study on NPM	2017	JavaScript; Node.js; Code Reuse; Empirical Studies
+	[26]	How to break an api: Cost negotiation and community values in three software ecosystem	2016	Software ecosystems; Dependency management; semantic versioning; Collaboration; Qualitative research
+	[37]	A historical analysis of Debian package incompatibilities	2015	debian, conflict, empirical, analysis, software, evolution, distribution, package, dependency, maintenance
+	[38]	An empirical comparison of developer retention in the RubyGems and NPM software ecosystems	2017	Software ecosystem, Socio-technical interaction, Software evolution, Empirical analysis, Survival analysis
+	[61]	Software Ecosystem Call Graph for Dependency Management	2018	
+	[73]	Structure and evolution of package dependency networks	2017	
+	[78]	Do developers update their library dependencies?	2017	Software reuse, Software maintenance, Security vulnerabilities
-	[90]	Studying Evolving Software Ecosystems based on Ecological Models	2013	Coral Reef, Natural Ecosystem, Open Source Software, Ecological Model, Software Project
+	[101]	Semantic versioning and impact of breaking changes in the Maven repository	2017	Semantic versioning, Breaking changes, Software libraries
+	[104]	How do developers react to API deprecation? The case of a smalltalk ecosystem	2012	Ecosystems, Mining Software Repositories, Empirical Studies
+	[118]	Adding sparkle to social coding: An empirical study of repository badges in the npm ecosystem	2018	

Table: 1. Papers provided by MSc. Joseph Hejderup. The first column describes whether the paper of the row will be used. A '+' means it will be used, a '-' means it will not.

Paper Reference	Reason not selected
[2]	This paper seems to delve more into one software project itself whereas we are more interested in the relationship between different software projects

Paper Reference	Reason not selected
[1]	Similarly to [2], we are more interested in the relationship between different software projects

Paper Reference	Reason not selected
[90]	We were in doubt over this one, it could be useful but we weren't convinced that it was. Since we already had a lot of material we decided to not use this

Table: 2. Papers from the initial seed that were not selected for the literature survey, along with a specification of the reason why this is the case.

6.2.2 Digital Search Engine

The second strategy type which is used to select relevant papers for this literature study, is by a digital search engine. In this literature survey, Google Scholar[6] is used. From the initial seed, common keywords were retrieved and the following queries have been used to search for relevant papers:

- "software ecosystems" AND "empirical analysis" *(2018)*
- "engineering software ecosystems" *(2014)*
- "software ecosystem" AND "empirical" *(2014)*

[6]https://github.com:

- "software ecosystem analytics" *(2014)*
- "software ecosystem" AND "analysis" *(2017)*
- "software ecosystem" AND "empirical" *(2018)*

For each of these queries, the results were first filtered by the publish year. These are described by the italic year after each query above. The papers that are filtered are published earlier than the set publish year. These specific years were chosen since the survey focuses on the state of the art within the ecosystem analytics.

After this filtering, we first determined whether a paper was relevant to the literature survey by examining the title. If it was unclear whether the paper was indeed relevant by only looking at the title, the abstract of the paper was examined closely. On these two criteria, each of the selected papers were judged and ultimately selected. The selected paper using these method can be found in Table 3.

First Author	Title	Year	Keywords	Query Used
[44]	An empirical comparison of dependency network evolution in seven software packaging ecosystems	2018	Software repository mining, Software ecosystem, Package manager, Dependency network, Software evolution	"software ecosystems" AND "empirical analysis"
[46]	Software engineering beyond the project – Sustaining software ecosystems	2014		engineering software ecosystems
[64]	How do developers react to API evolution? A large-scale empirical study	2016	API evolution, API deprecation, Software ecosystem, Empirical study	"software ecosystem" AND "empirical"
[66]	Software Development Analytics for Xen: Why and How	2018	Companies, Ecosystems, Software, Measurement, Object recognition, Monitoring, Virtualization	software ecosystem analytics
[67]	Measuring the Health of Open Source Software Ecosystems: Beyond the Scope of Project Health	2014		"open source software ecosystems"
[77]	An exploratory study on library aging by monitoring client usage in a software ecosystem	2017		"software ecosystem" AND "analysis"
[83]	An empirical analysis of the transition from Python 2 to Python 3	2018	Python programming, Programming language evolution, Compliance	"software ecosystem" AND "empirical"
[85]	Revisiting software ecosystems Research: A longitudinal literature study	2016	Software ecosystems; Longitudinal literature study; Software ecosystem maturity	"Software ecosystems" OR "Dependency management" OR "semantic version"
[102]	Software evolution and maintenance	2014		Software Evolution and Maintenance

First Author	Title	Year	Keywords	Query Used
[115]	Lessons learned from applying social network analysis on an industrial Free/Libre/Open Source Software Ecosystem	2015	Social network analysis Open source Open-coopetition Software ecosystems Business models Homophily Cloud computing OpenStack	"software ecosystem analytics"

Table: 3. Papers selected from searches using Google Scholar. The column "Query Used" describes which of the queries is used to retrieve the paper.

6.2.3 Referenced papers

Finally, a selection of papers has been made by looking at the references found in papers selected using the two methods above. For these papers, the selection process is similar to that of the selected papers using the digital search engine; it is selected when both the title and the abstract are deemed relevant to the research questions. This has led to the papers in Table 4. being selected.

First Author	Title	Year	Keywords	Referenced In
[11]	How the Apache community upgrades dependencies: an evolutionary study	2014	Software Ecosystems · Project dependency upgrades · Mining software repositories	[78]
[25]	Ecosystems in GitHub and a method for ecosystem identification using reference coupling.	2015		[38]
[41]	Measuring Dependency Freshness in Software Systems	2015		[73]
[43]	An empirical comparison of dependency issues in OSS packaging ecosystems	2017		[3], [38], [44]
[45]	Broken Promises - An Empirical Study into Evolution Problems in Java Programs Caused by Library Upgrades	2014		[101]
[84]	Quantifying the transition from Python 2 to 3: an empirical study of Python applications.	2017		[83]
[88]	An empirical study of api stability and adoption in the android ecosystem	2013		[85]

Table: 4. Papers selected which are referenced in previously selected papers. The column "Referenced In" describes in which selected paper the paper is referenced.

6.3 Answers

In this section, an aggregation of information, found in the papers, is presented. Each subsection of this section focuses on one of the three research questions posed in Section 1.

6.3.1 What is the current state of the art in software analytics for ecosystem analytics?

To answer this research question, we examine the explored topics in ecosystem analytics. Moreover, we summarize which research methods, tools and datasets are being used to explore this topics.

The main topic explored in the selected papers are related to the dependencies within the software ecosystem. One of the main subjects related to these dependencies is the subject of breaking changes between different versions of a package. [26] researched the attitude of developers of Eclipse, CRAN and NPM packages towards making breaking changes.

Reference	Explored topic(s)	Research method(s)	Tool(s)	Dataset(s)	Ecosystem(s)	Conclusion
[3]	Empirical study on the use of trivial packages, as well as the reasoning behind this	Quantitative frequency, Survey	-	NPM, GitHub	NPM	Used because it is assumed to be well implemented and tested (only 45% actually has tests) and increases productivity. Quantitative research has shown that 10% of NodeJS uses trivial packages, where 16.8% are trivial packages in NPM

Reference	Explored topic(s)	Research method(s)	Tool(s)	Dataset(s)	Ecosystem(s)	Conclusion
[26]	Attitude of towards breaking changes and how do ecosystems influence this	Interviews	-	-	Eclipse, CRAM, NPM	There are numerous ways of dealing with breaking changes and ecosystems play an essential role in the chosen way.

Reference	Explored topic(s)	Research method(s)	Tool(s)	Dataset(s)	Ecosystem(s)	Conclusion
[44]	Quantative empirical analysis of differences and similarities between the evolution of 7 varying ecosystems	Survival analysis	-	libraries.io	Cargo, CPAN, CRAN, npm, NuGet, Packagist, RubyGems	Package updates, which may cause dependent package failrues, are done on average every few months. Many packages in the analyzed package dependency networks were found to have a high number of transitive reverse dependencies, implying that package failures can affect a large number of other packages in the ecosystem.

Reference	Explored topic(s)	Research method(s)	Tool(s)	Dataset(s)	Ecosystem(s)	Conclusion
[46]	The article provides a holistic understanding of the observed and reported practices as a starting point to device specific support for the development in software ecosystems	Qualitative interview study	-	-	-	The main contribution of this article is the presentation of common features of product development and evolution in four companies. Although size, kind of software and business models differ
[66]	Code review analysis	Virtualization of process	-	Xen Github data	Xen	Analysis of code review has lead to more reviews and a more thoughtful and participary review process. Also providing accomodations for new software developers on OSS by easy access is very important.

6.3.2 What are the practical implications from the state of the art?

Reference	Explored topic(s)	Research method(s)	Dataset(s)	Ecosystem(s)	Conclusion
[64]	Exploratory study aimed at observing API evolution and its impact	Empirical study	3600 distinct systems	Pharo	After API changes, clients need time to react and rarely react at all. Replacements cannot be resolved in a uniform manner throughout the ecosystem. API changes and deprication can present different characteristics.
[78]	An Empirical Study on the Impact of Security Advisories on Library Migration	Empirical study	4,600 GitHub software projects and 2,700 library dependencies	Github, Maven	Currently, developers do not actively update their libraries, leading to security risks.

6.3.3 What are the open challenges in ecosystem analytics, for which future research is required?

Reference	Open Challenges Found
[3]	Examine relationship between team experience and project maturity and usage of trivial packages
[3]	Compare use of code snippets on Q&A sites and trivial packages
[3]	How to manage and help developers choose the best packages

Reference	Open Challenges Found
[44]	Findings for one ecosystem cannot necessarily be generalized to another
[44]	Transitive dependencies are very frequent, meaning that package failrues can affect a large number of other packages in the ecosystem
[67]	Determing the health of a system from an ecosystem perspective instead of project level is needed to determine which systems to use. This paper provides an initial approach but a lot more research could and should be done to determine system health.

Chapter 7

Release Engineering Analytics

7.1 Motivation

Release engineering is a discipline involved with making software available for end users. Efforts spent within the development environment of a software system should eventually be integrated and deployed such that end users may benefit from them. In recent years, release engineers have developed and adopted techniques to build infrastructures and pipelines which automate the process of releasing software to an increasingly large degree. These modern approaches have resulted in various practices such as releasing new versions of a software system in significantly shorter cycles.

Due to these developments being industry-driven, release engineering forms a largely uncharted territory for software engineering research. It requires the attention from researchers both because these new practices have an often unanticipated impact on software studies and because they require empirical validation [4].

Therefore, this systematic literature review aims to provide an overview of the software analytics research that has been conducted so far on modern release engineering. Its main purpose is to identify the apparent gap between research and practice, in order to guide further research efforts.

7.1.1 Research Questions

Contrary to what is regularly the case, advances in release engineering practices are driven by industry, instead of scientific research. Building on this idea, our questions are constructed to identify in which ways existing modern release engineering practices should still be studied in software analytics research. Our review thus aims to answer the following questions.

- **RQ 1:** *How is modern release engineering done in practice?* This question aims to identify the so-called "state of the practice" in release engineering. We will summarize practices that have been adopted to drive release engineering forward. In addition we will identify the tools utilized to bring this about. Case studies will also be analyzed to this end.
- **RQ 2:** *What aspects of modern release engineering have been studied in software analytics research so far?* In order to answer this question we investigate the practices that previous empirical studies have focused on. In doing so, we identify the associated costs and benefits that have been found, and the analysis methods used.
- **RQ 3:** *What aspects of modern release engineering make for relevant study objects in future software analytics research?* In answering this question we aim to identify the gap between practice and research in release engineering. This way, our intent is not only to guide but also to motivate future research.

7.2 Research Protocol

In this section, we will describe…

7.2.1 Search Strategy

Since release engineering is a relatively new research topic, we took an exploratory approach in collecting any literature revolving around the topic of release engineering from the perspective of software analytics. This aided us in determining a more narrow scope for our survey, subsequently allowing us to find additional literature fitting this scope.

At the start of this project, we were provided with an initial seed of five papers as a starting point for our literature survey. These initial papers were [4], [40], [39], [72], and [71].

We collected publications using two search engines: Scopus and Google Scholar. Each of the two search engines comprises several databases such as ACM Digital Library, Springer, IEEE Xplore and ScienceDirect. The main query that we constructed is displayed in Figure 1. The publications found using this query were:

- [69]
- [70]
- [30]
- [68]
- [36]
- [54]
- [111]
- [79]

```
TITLE-ABS-KEY(
  (
    "continuous release" OR "rapid release" OR "frequent release"
    OR "quick release" OR "speedy release" OR "accelerated release"
    OR "agile release" OR "short release" OR "shorter release"
    OR "lightning release" OR "brisk release" OR "hasty release"
    OR "compressed release" OR "release length" OR "release size"
    OR "release cadence" OR "release frequency"
    OR "continuous delivery" OR "rapid delivery" OR "frequent delivery"
    OR "fast delivery" OR "quick delivery" OR "speedy delivery"
    OR "accelerated delivery" OR "agile delivery" OR "short delivery"
    OR "lightning delivery" OR "brisk delivery" OR "hasty delivery"
    OR "compressed delivery" OR "delivery length" OR "delivery size"
    OR "delivery cadence" OR "continuous deployment" OR "rapid deployment"
    OR "frequent deployment" OR "fast deployment" OR "quick deployment"
    OR "speedy deployment" OR "accelerated deployment" OR "agile deployment"
    OR "short deployment" OR "lightning deployment" OR "brisk deployment"
    OR "hasty deployment" OR "compressed deployment" OR "deployment length"
    OR "deployment size" OR "deployment cadence"
  ) AND (
    "release schedule" OR "release management" OR "release engineering"
    OR "release cycle" OR "release pipeline" OR "release process"
    OR "release model" OR "release strategy" OR "release strategies"
    OR "release infrastructure"
  )
  AND software
) AND (
  LIMIT-TO(SUBJAREA, "COMP") OR LIMIT-TO(SUBJAREA, "ENGI")
```

```
)
AND PUBYEAR AFT 2014
```

Figure 1. Query used for retrieving release engineering publications via Scopus.

In addition to querying search engines as described above, references related to retrieved papers were analyzed. These reference lists were obtained from Google Scholar and from the *References* section in the papers themselves. We selected all papers on release engineering that are citing or being cited by the initial set of papers. Using this approach, we have found six additional papers. The results of the reference analysis are listed in Table 1.

Table 1. Papers found indirectly by investigating citations of/by other papers.

Starting point	Type	Result
[111]	has cited	[99] [87]
[71]	is cited by	[100] [114]
[87]	is cited by	[106] [34]

All the papers that were found, were stored in a custom built web-based tool for conducting literature reviews. The source code of this tool is published in a GitHub repository. The tool was hosted on a virtual private server, such that all retrieved publications were stored centrally, accessible to all reviewers.

7.2.2 Study Selection

We selected the studies that we wanted to include in the survey with aid of the aforementioned tool for storing the papers. In this tool, it is possible to label papers with tags and leave comments and ratings. Every paper is reviewed based on the selection criteria. Based on this, the tool allowed to filter out all papers that appeared not to be relevant for this literature survey.

The selection criteria are as follows:

1. The study must show (at least) one release engineering technique.
2. The study must not just show a release engineering technique, but analyze its performance compared to other techniques.

Based on these selection criteria, the following papers appeared to be irrelevant for the scope of this survey:

- [link to paper] - Excluded based on rule 2.

7.2.3 Study Quality Assessment

Based on [76], the quality of a paper will be assessed by the evidence it provides, based on the following scale. All levels of quality in this scale will be accepted, except for level 5 (evidence obtained from expert opinion).

1. Evidence obtained from at least one properly-designed randomised controlled trial.
2. Evidence obtained from well-designed pseudo-randomised controlled trials (i.e. non-random allocation to treatment).
3. Comparative studies in a real-world setting:
    1. Evidence obtained from comparative studies with concurrent controls and allocation not randomised, cohort studies, case-control studies or interrupted time series with a control group.
    2. Evidence obtained from comparative studies with historical control, two or more single arm studies, or interrupted time series without a parallel control group.
4. Experiments in artificial settings:
    1. Evidence obtained from a randomised experiment performed in an artificial setting.

2. Evidence obtained from case series, either post-test or pre-test/post-test.
3. Evidence obtained from a quasi-random experiment performed in an artificial setting.

5. Evidence obtained from expert opinion based on theory or consensus.

Also, the studies will be examined to see if they contain any type of bias. For this, the same types of biases will be used as described by [76]:

- Selection/Allocation bias: Systematic difference between comparison groups with respect to treatment.
- Performance bias: Systematic difference is the conduct of comparison groups apart from the treatment being evaluated.
- Measurement/Detection bias: Systematic difference between the groups in how outcomes are ascertained.
- Attrition/Exclusion bias: Systematic differences between comparison groups in terms of withdrawals or exclusions of participants from the study sample.

The studies will be labeled by their quality level and possible biases. This information can be used during the Data Synthesis phase to weigh the importance of individual studies [76].

7.2.4 Data Extraction

To accurately capture the information contributed by each publication in our survey, we will use a systematic approach to extracting data. To guide this process, we will be using a data extraction form which describes what aspects of a publication are crucial to record. Besides general publication information (title, author etc.), the form contains questions that are based on our defined research questions. Furthermore, the form contains a section for quantitative research, where aspects such as population and evaluation will be documented. The form that is used for this is shown below:

```
General information:

- Name of person extracting data:
- Date form completed (dd/mm/yyyy):
- Publication title:
- Author information:
- Publication type:
- Conference/Journal:
- Type of study:

What practices in release engineering does this publication mention?

Are these practices to be classified under dated, state of the art or state of
the practice? Why?

What open challenges in release engineering does this publication mention?

What research gaps does this publication contain?

Are these research gaps filled by any other publications in this survey?

Quantitative research publications:

- Study start date:
- Study end date or duration:
- Population description:
- Method(s) of recruitment of participants:
- Sample size:
```

```
- Evaluation/measurement description:
- Outcomes:
- Limitations:
- Future research:

Notes:
```

7.2.5 Data Synthesis

To summarize the contributions and limitations of each of the included publications, we will apply a descriptive synthesis approach. In this part of our survey, we will compare the data that was extracted of the included publications. Publications with similar findings will be grouped and evaluated, and differences between groups of publications will be structured and elaborated on. In this we will compare them using specifics such as their study types, time of publication and study quality.

If the extracted data allows for a structured tabular visualization of similarities and differences between publications this we serve as an additional form of synthesis. However, this depends on the final included publications of this survey.

7.2.6 Included and Excluded Studies

Included:

- [4]
- **???**
- [34]
- [36]
- [39]
- [40]
- [48]
- **???**
- [68]
- [70]
- [71]
- [79]
- [87]
- [99]
- [100]
- [106]
- [111]
- [114]

Excluded:

- [72] has been excluded, because it presents the same results as [71], while the latter is more extensive because it is a journal article instead of a conference article.

7.2.7 Project timetable

The literature review was conducted over the course of four weeks. We worked iteratively and planned for four weekly milestones.

Milestone	Deadline	Goals
Milestone 1	16/9/18	- Develop the search strategy - Collect initial publications
Milestone 2	23/9/18	Write full research protocol
Milestone 3	30/9/18	- Collect additional literature according to the protocol - Perform data extraction
Milestone 4	7/10/18	- Perform data synthesis - Write final version of the chapter

7.3 Answers

7.3.1 RQ1: ...

7.3.2 RQ2: ...

7.3.3 RQ3: ...

7.4 Discussion

7.5 Conclusion

7.6 Raw extracted data

This section is not part of the main content of our chapter. It can be viewed as an Appendix.

7.6.1 Understanding the impact of rapid releases on software quality – The Case of Firefox

Reference: [71]

General information:

- Name of person extracting data: Maarten Sijm
- Date form completed: 27-09-2018
- Author information: Foutse Khomh, Bram Adams, Tejinder Dhaliwal, Ying Zou
- Publication type: Paper in Conference Proceedings
- Conference: Mining Software Repositories (MSR)
- Type of study: Quantitative, empirical case study

What practices in release engineering does this publication mention?

- Changing from traditional to rapid release cycles in Mozilla Firefox

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the practice, because they study Firefox and Firefox is still using rapid release cycles. However, it is dated because the data is six years old.

What open challenges in release engineering does this publication mention?

- More case studies are needed

What research gaps does this publication contain?

- More case studies are needed

Are these research gaps filled by any other publications in this survey?

- Not yet known **TODO**

Quantitative research publications:

- Study start date: 01-01-2010 (Firefox 3.6)
- Study end date or duration: 20-12-2011 (Firefox 9.0)
- Population description: Mozilla Wiki, VCS, Crash Repository, Bug Repository
- Method(s) of recruitment of participants: N/A (case study)
- Sample size: 25 alpha versions, 25 beta versions, 29 minor versions and 7 major versions. Amount of bugs/commits/etc. is not specified.
- Evaluation/measurement description: Wilcoxon rank sum test
- Outcomes:
  - With shorter release cycles, users do not experience significantly more post-release bugs
  - Bugs are fixed faster
  - Users experience these bugs earlier during software execution (the program crashes earlier)
- Limitations: Results are specific to Firefox
- Future research: More case studies are needed

7.6.2 On the influence of release engineering on software reputation

Reference: [99]

General information:

- Name of person extracting data: Maarten Sijm
- Date form completed: 27-09-2018
- Author information: Christian Plewnia, Andrej Dyck, Horst Lichter
- Publication type: Paper in Conference Proceedings
- Conference: 2nd International Workshop on Release Engineering
- Type of study: Quantitative, empirical case study on multiple software

What practices in release engineering does this publication mention?

- Rapid releases

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- Dated practice, data is from before 2014

What open challenges in release engineering does this publication mention?

- Identifying software reputation can better be done using a qualitative study.

What research gaps does this publication contain?

- Identifying software reputation can better be done using a qualitative study.

Are these research gaps filled by any other publications in this survey?

- Not yet known **TODO**

Quantitative research publications:

- Study start date: Q3 2008
- Study end date or duration: Q4 2013
- Population description: Chrome, Firefox, Internet Explorer
- Method(s) of recruitment of participants: N/A (case study)

- Sample size: 3 browsers
- Evaluation/measurement description: No statistical analysis, just presenting market share results
- Outcomes:
  - Chrome's market share increased after adopting rapid releases
  - Firefox's market share decreased after adopting rapid releases
  - IE's market share decreased
- Limitations:
  - Identifying software reputation can better be done using a qualitative study.
- Future research:
  - Identifying software reputation can better be done using a qualitative study.

7.6.3 On rapid releases and software testing: a case study and a semi-systematic literature review

Reference: [87]

General information:

- Name of person extracting data: Maarten Sijm
- Date form completed: 28-09-2018
- Author information: Mäntylä, Mika V. and Adams, Bram and Khomh, Foutse and Engström, Emelie and Petersen, Kai
- Publication type: Journal/Magazine Article
- Journal: Empirical Software Engineering
- Type of study: Empirical case study and semi-systematic literature review

What practices in release engineering does this publication mention?

- Impact of rapid releases on testing effort

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the practice for the case study
- State of the art for the literature review

What open challenges in release engineering does this publication mention?

- Future work should focus on empirical studies of these factors that complement the existing qualitative observations and perceptions of rapid releases.

What research gaps does this publication contain?

- See open challenges

Are these research gaps filled by any other publications in this survey?

- Not yet known **TODO**

Quantitative research publications:

- Study start date: June 2006 (Firefox 2.0)
- Study end date or duration: June 2012 (Firefox 13.0)
- Population description: System-level test execution data
- Method(s) of recruitment of participants: N/A (case study)
- Sample size: 1,547 unique test cases, 312,502 executions, performed by 6,058 individuals on 2,009 software builds, 22 OS versions and 78 locales.
- Evaluation/measurement description: Wilcoxon rank-sum test, Cliff's delta, Cohen's Kappa for Firefox Research Question (FF-RQ) 5.
- Outcomes (FF-RQs; RR = rapid release; TR = traditional release):

    1. RRs perform more test executions per day, but these tests focus on a smaller subset of the test case corpus.
    2. RRs have less testers, but they have a higher workload.
    3. RRs test fewer, but larger builds.
    4. RRs test fewer platforms in total, but test each supported platform more thoroughly.
    5. RRs have higher similarity of test suites and testers within a release series than TRs had.
    6. RR testing happens closer to the release date and is more continuous, yet these findings were not confirmed by the QA engineer.
- Limitations:
  - Study measures correlation, not causation
  - Not generalizable, as it is a case study on FF
- Future research: More empirical studies

Semi-systematic literature survey:

- Study date: Unknown (before 2015)
- Population description: Papers with main focus on:
  - Rapid Releases (RRs)
  - Aspect of software engineering largely impacted by RRs
  - An agile, lean or open source process having results of RRs
  - Excluding: opinion papers without empirical data on RRs
- Method(s) of recruitment of participants: Scopus queries
- Sample size: 24 papers
- Outcomes:
  - Evidence is scarce. Often RRs are implemented as part of agile adoption. This makes it difficult to separate the impact of RRs from other process changes.
  - Originates from several software development paradigms: Agile, FOSS, Lean, internet-speed software development
  - Prevalence
    * Practiced in many software engineering domains, not just web applications
    * Between 23% and 83% of practitioners do RRs
  - (Perceived) Problems:
    * Increased technical debt
    * RRs are in conflict with high reliability and high test coverage
    * Customers might be dipleased with RRs (many updates)
    * Time-pressure / Deadline oriented work
  - (Perceived) Benefits:
    * Rapid feedback leading to increased quality focus of the devs and testers
    * Easier monitoring of progress and quality
    * Customer satisfaction
    * Shorter time-to-market
    * Continuous work / testing
  - Enablers:
    * Sequential development where multiple releases are under work simultaneously
    * Tools for automated testing and efficient deployment
    * Involvement of product management and productive customers
- Limitations:
  - Not all papers that present results about RRs, have “rapid release” mentioned in the abstract.
- Future research:
  - Systematically search for agile and lean adoption papers

Notes:

- Basically contains all the answers we need

7.6.4 Release management in free and open source software ecosystems

Reference: [100]

General information:

- Name of person extracting data: Maarten Sijm
- Date form completed: 28-09-2018
- Author information: Germán Poo-Caamaño
- Publication type: PhD Thesis
- Type of study: Empirical case study on two large-scale FOSSs: GNOME and OpenStack

What practices in release engineering does this publication mention?

- Communication in release engineering

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the practice, because case study

What open challenges in release engineering does this publication mention?

- Is the ecosystem [around the studied software] shrinking or expanding?
- How have communications in the ecosystem changed over time?

What research gaps does this publication contain?

- More case studies are needed

Are these research gaps filled by any other publications in this survey?

- Not yet known **TODO**

Quantitative research publications (GNOME):

- Study start date: January 2009 (GNOME 2.x)
- Study end date or duration: August 2011 (GNOME 3.x)
- Population description: Mailing lists
- Method(s) of recruitment of participants: GNOME's website recommends this channel of communication. IRC is also recommended, but its history is not stored.
- Sample size: 285 mailing lists, 6947 messages, grouped into 945 discussions.
- Evaluation/measurement description: Counting
- Outcomes:
  - Developers also communicate via blogs, bug trackers, conferences, and hackfests.
  - The Release Team has direct contact with almost all participants in the mailing list
  - The tasks of the Release Team:
    - defining requirements of GNOME releases
    - coordinating and communicating with projects and teams
    - shipping a release within defined quality and time specifications
  - Major challenges of the Release Team:
    - coordinate projects and teams of volunteers without direct power over them
    - keep the build process manageable
    - monitor for unplanned changes
    - monitor for changes during the stabilization phase
    - test the GNOME release
- Limitations:
  - Only mailing list was investigated, other channels were not
  - Possible subjective bias in manually categorizing email subjects
  - Not very generalizable, as it's just one case study
- Future research:
  - Fix the limitations

Quantitative research publications (OpenStack):

- Study start date: May 2012
- Study end date or duration: July 2014
- Population description: Mailing lists
- Method(s) of recruitment of participants: Found on OpenStack's website
- Sample size: 47 mailing lists, 24,643 messages, grouped into 7,650 discussions. Filtered data: 14,486 messages grouped into 2,682 discussions.
- Evaluation/measurement description: Counting
- Outcomes:
  - Developers communicate via email, blogs, launchpad, wiki, gerrit, face-to-face, IRC, video-conferences, and etherpad.
  - Project Team Leaders and the Release Team members are the key players in the communication and coordination across projects in the context of release management
  - The tasks for the Release Team and Project Team Leaders:
    - defining the requirements of an OpenStack release
    - coordinating and communicating with projects and teams to reach the objectives of each milestone
    - coordinating feature freeze exceptions at the end of a release
    - shipping a release within defined quality and time specifications
  - Major challenges of these teams:
    - coordinate projects and teams without direct power over them
    - keep everyone informed and engaged
    - decide what becomes part of of the integrated release
    - monitor changes
    - set priorities in cross-project coordination
    - overcome limitations of the communication infrastructure
- Limitations:
  - Only studies mailing list, to compare with GNOME case study
  - Possible subjective bias in manually categorizing email subjects
  - Not very generalizable, as it's just one case study
- Future research:
  - Fix the limitations

Notes:

- Since there are two case studies, the results become a bit more generalizable
- The author set up a theory that encapsulates the communication and coordination regarding release management in FOSS ecosystems, and can be summarized as:
  1. The size and complexity of the integrated product is constrained by the release managers capacity
  2. The release management should reach the whole ecosystem to increase awareness and participation
  3. The release managers need social and technical skills

7.6.5 Release Early, Release Often and Release on Time. An Empirical Case Study of Release Management

Reference: [114]

General information:

- Name of person extracting data: Maarten Sijm
- Date form completed: 28-09-2018
- Author information: Jose Teixeira
- Publication type: Paper in Conference Proceedings
- Conference: Open Source Systems: Towards Robust Practices

- Type of study: Empirical case study

What practices in release engineering does this publication mention?

- Shifting towards rapid releases in OpenStack

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the practice, because it is a recent case study on OpenStack

What open challenges in release engineering does this publication mention?

- More case studies are needed.

What research gaps does this publication contain?

- More case studies are needed.

Are these research gaps filled by any other publications in this survey?

- Not yet known **TODO**

Quantitative research publications:

- Study start date: Not specified
- Study end date or duration: Not specified
- Population description: Websites and blogs
- Method(s) of recruitment of participants: Random clicking through OpenStack websites
- Sample size: Not specified
- Evaluation/measurement description: Not specified
- Outcomes:
  - OpenStack releases in a cycle of six months
  - The release management process is a hybrid of feature-based and time-based
  - Having a time-based release strategy is a challenging coopearative task involving multiple people and technology
- Limitations:
  - Study is not completed yet, these are preliminary results
- Future research:
  - Not indicated

7.6.6 Kanbanize the release engineering process

Reference: [70]

General information:

- Name of person extracting data: Jesse Tilro
- Date form completed: 29-09-2018
- Author information: Kerzazi, N. and Robillard, P.N.
- Publication type: Paper in Conference Proceedings
- Journal: 2013 1st International Workshop on Release Engineering, RELENG 2013 - Proceedings
- Type of study: Action research

What practices in release engineering does this publication mention?

- Following principles of the Kanban agile software development life-cycle model that implicitly describe the release process
- (Switching to) more frequent (daily) release cycles
- (Transitioning to) a structured release process

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- Either dated or state of the practice, not sure. Would have to do some additional research on the adoption of Kanban

What open challenges in release engineering does this publication mention?

- Release effectiveness: minimize system failure and customer impact
- Problems with releasing encountered in practice
  - **TODO** list problems if of interest

What research gaps does this publication contain?

-

Are these research gaps filled by any other publications in this survey?

-

Quantitative research publications:

- Study start date:
- Study end date or duration:
- Population description:
- Method(s) of recruitment of participants:
- Sample size:
- Evaluation/measurement description:
- Outcomes:
  1.
-

7.7 Limitations:

- Future research:

Notes:

-

7.7.1 Is it safe to uplift this patch? An empirical study on mozilla firefox

Reference: [30]

General information:

- Name of person extracting data: Jesse Tilro
- Date form completed: 29-09-2018
- Author information: Castelluccio, M. and An, L. and Khomh, F.
- Publication type: Paper in Conference Proceedings
- Journal: Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017
- Type of study: Case study, both quantitative (data analysis) and qualitative (interviews)

What practices in release engineering does this publication mention?

- Patch uplift (meaning the promotion of patches from development directly to a stabilization channel, potentially skipping several channels)

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the practice: case study of what is being done in the field, quite recently (2017).

What open challenges in release engineering does this publication mention?

- Exploring possibilities to leverage this research by building classifiers capable of automatically assessing the risk associated with patch uplift candidates and recommend patches that can be uplifted safely.
- Validate and extend results of this study for generalizability.

What research gaps does this publication contain?

- Study aimed to fill two identified gaps identified in literature:
  - How do urgent patches in rapid release models affect software quality (in terms of fault proneness)?
  - How can the reliability of the integration of urgent patches be improved?

Are these research gaps filled by any other publications in this survey?

- The paper itself

Quantitative research publications:

- Study start date:
- Study end date or duration:
- Population description:
- Method(s) of recruitment of participants:
- Sample size:
- Evaluation/measurement description:
- Outcomes:
  1.
- Limitations:
- Future research:

Notes:

-

7.7.2 Systematic literature review on the impacts of agile release engineering practices

Reference: [68]

General information:

- Name of person extracting data: Jesse Tilro
- Date form completed: 29-09-2018
- Author information: Karvonen, T. and Behutiye, W. and Oivo, M. and Kuvaja, P.
- Publication type: Journal/Magazine Article
- Journal: Information and Software Technology
- Type of study: Systematic literature review

What practices in release engineering does this publication mention?

- Agile release engineering (ARE) practices
  - Continuous integration (CI)
  - Continuous delivery (CD)
  - Rapid Release (RR)
  - Continuous deployment

  - DevOps (similar to CD, congruent with release engineering practices)

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the art, for it concerns a state of the art report and was published recently (2017).

What open challenges in release engineering does this publication mention?

- Claims that modern release engineering practices allow for software to be delivered faster and cheaper should be further empirically validated.
- This analysis could be extended with industry case studies, to develop a checklist for analyzing company and ecosystem readiness for continuous delivery and continuous deployment.
- The comprehensive reporting of the context and how the practice is implemented instead of merely referring to usage of the practice should be considered by future research.
- Different stakeholders' points of view, such as customer perceptions regarding practices require further research.
- Research on DevOps would be highly relevant for release engineering and the continuous software engineering research domain.
- Future research on the impact of RE practices could benefit from more extensive use of quantitative methodologies from case studies, and the combination of quantitative with qualitative (e.g. interviews) methods.

What research gaps does this publication contain?

- Refer to challenges

Are these research gaps filled by any other publications in this survey?

- **TODO**

Quantitative research publications:

- Study start date: N/A
- Study end date or duration: N/A
- Population description: N/A
- Method(s) of recruitment of participants: N/A
- Sample size: N/A
- Evaluation/measurement description: N/A
- Outcomes: N/A
- Limitations: N/A
- Future research: N/A

Notes:

-

7.7.3 Abnormal Working Hours: Effect of Rapid Releases and Implications to Work Content

Reference: [36]

General information:

- Name of person extracting data: Jesse Tilro
- Date form completed: 29-09-2018
- Author information: Claes, M. and Mantyla, M. and Kuutila, M. and Adams, B.
- Publication type: Paper in Conference Proceedings
- Journal: IEEE International Working Conference on Mining Software Repositories
- Type of study: Quantitative case study

What practices in release engineering does this publication mention?

- Faster release cycles

Are these practices to be classified under dated, state of the art or state of the practice? Why?

-

What open challenges in release engineering does this publication mention?

- Future research might further study the impact of time pressure and work patterns - indirectly release practices - on software developers.

What research gaps does this publication contain?

-

Are these research gaps filled by any other publications in this survey?

-

Quantitative research publications:

- Study start date: first data item 2012-12-21
- Study end date or duration: last data item 2016-01-03
- Population description: N/A
- Method(s) of recruitment of participants: N/A
- Sample size: 145691 bug tracker contributors (1.8% timezone), 11.11 million comments (53% author with timezone)
- Evaluation/measurement description: measure distributions on number of comments per day of the week and time of the day, before and after transition to rapid release cycles. Test distribution difference using Mann-Whitney U test and test effect size using Cohen's d and Cliff's delta. Also evaluate general development of number of comments, working day against weekend and day against night.
- Outcomes:
  1. Switching to rapid releases has reduced the amount of work performed outside of office hours. (Supported by results in psychology.)
  2. Thus, rapid release cycles seem to have a positive effect on occupational health.
  3. Comments posted during the weekend contained more technical terms.
  4. Comments posted during weekdays contained more positive and polite vocabulary.
- Limitations:
- Future research:

Notes:

-

7.7.4 Does the release cycle of a library project influence when it is adopted by a client project?

Reference: [54]

General information:

- Name of person extracting data: Jesse Tilro
- Date form completed: 29-09-2018
- Author information: Fujibayashi, D. and Ihara, A. and Suwa, H. and Kula, R.G. and Matsumoto, K.
- Publication type: Paper in Conference Proceedings
- Journal: SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering
- Type of study: Quantitative study

What practices in release engineering does this publication mention?

- Rapid release cycles

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the art and practice: practitioners currently practice it, researchers currently research it.

What open challenges in release engineering does this publication mention?

- Gaining an understanding of the effect of a library's release cycle on its adoption.

What research gaps does this publication contain?

- First step towards solving the above challenge.

Are these research gaps filled by any other publications in this survey?

- This paper

Quantitative research publications:

- Study start date: 21-07-2016 (data extraction)
- Study end date or duration:
- Population description:
- Method(s) of recruitment of participants:
- Sample size: 23 libraries, 415 client projects
- Evaluation/measurement description:
- Scott-Knott test to group libraries with similar release cycle.
- Outcomes:
  1. There is a relationship between release cycle of a library project and the time for clients to adopt it: quicker release seems to be associated with quicker adoption.
- Limitations:
  - Small sample size
  - Not controlled for many factors
  - No statistical significance tests?
- Future research:

Notes:

- Very short, probably not very strong evidence, refer to limitations
- Nice that the focus is libraries here, very interesting population because most studies focus on end-user targeting software systems

7.7.5 Rapid releases and patch backouts: A software analytics approach

Reference: [111]

General information:

- Name of person extracting data: Jesse Tilro
- Date form completed: 29-09-2018
- Author information: Souza, R. and Chavez, C. and Bittencourt, R.A.
- Publication type: Journal/Magazine Article
- Journal: IEEE Software
- Type of study: Quantitative

What practices in release engineering does this publication mention?

-

Are these practices to be classified under dated, state of the art or state of the practice? Why?

-

What open challenges in release engineering does this publication mention?

- How rapid release cycles affect code integration

What research gaps does this publication contain?

-

Are these research gaps filled by any other publications in this survey?

-

Quantitative research publications:

- Study start date:
- Study end date or duration:
- Population description:
- Method(s) of recruitment of participants:
- Sample size:
- Evaluation/measurement description:
- Outcomes:
  1.
- Limitations:
- Future research:

Notes:

- Also reviews existing literature very well

7.7.6 Comparison of release engineering practices in a large mature company and a startup

Reference: [79]

General information:

- Name of person extracting data: Jesse Tilro
- Date form completed: 29-09-2018
- Author information: Laukkanen, E. and Paasivaara, M. and Itkonen, J. and Lassenius, C.
- Publication type: Journal/Magazine Article
- Journal: Empirical Software Engineering
- Type of study:

What practices in release engineering does this publication mention?

-

Are these practices to be classified under dated, state of the art or state of the practice? Why?

-

What open challenges in release engineering does this publication mention?

-

What research gaps does this publication contain?

-

Are these research gaps filled by any other publications in this survey?

-

Quantitative research publications:

- Study start date:
- Study end date or duration:
- Population description:
- Method(s) of recruitment of participants:
- Sample size:
- Evaluation/measurement description:
- Outcomes:
    1.
-

7.8 Limitations:

- Future research:

Notes:

-

7.8.1 Modern Release Engineering in a Nutshell

Reference: [4]

General information:

- Name of person extracting data: Nels Numan
- Date form completed (dd/mm/yyyy): 28/09/2018
- Publication title: Modern Release Engineering in a Nutshell
- Author information: Bram Adams and Shane McIntosh
- Journal: 23rd International Conference on Software Analysis, Evolution, and Reengineering (2016)
- Publication type: Conference paper
- Type of study: Survey

What practices in release engineering does this publication mention?

- Branching and merging
    - Software teams rely on Version Control Systems
    - Quality assurance activities like code reviews are used before doing a merge or even allowing a code change to be committed into a branch
    - Keep branches short-lived and merge often. If this is impossible, a rebase can be done.
    - "trunk-based development" can be applied to eliminate most branches below the master branch.
    - Feature toggles are used to provide isolation for new features in case of the absence of branches.
- Building and testing
    - To help assess build and test conflicts, many projects also provide "try" servers to development teams, which automatically runs a build and test process referred to as CI.
    - The CI process often does not run full test, but a representative subset.
    - The more intensive tests, such as integration, system or performance typically get run nightly or in weekends.
- Build system:

  - GNU Make is the most popular file-based build system technology. Ant is the prototypical task-based build system technology. Lifecycle-based build technologies like Maven consider the build system of a project to have a sequence of standard build activities that together form a "build lifecycle."
  - "Reproducible builds" involve for a given feature and hardware configuration of the code base, every build invoca- tion should yield bit-to-bit identical build results.
- Infrastructure-as-code
  - Containers or virtual machines are used to deploy new versions of the system for testing or even production.
  - It has been recommended that infrastructure code is to be stored in a separate VCS repository than source code, in order to restrict access to infrastructure code.
- Deployment
  - The term "dark launching" corresponds to deploying new features without releasing them to the public, in which parts of the system automatically make calls to the hidden features in a way invisible to end users.
  - "Blue green deployment" deploys the next software version on a copy of the production environment, and changes this to be the main enviroment on release.
  - In "canary deployment" a prospective release of the software system is loaded onto a subset of the production environments for only a subset of users.
  - "A/B testing" deploys alternative A of a feature to the environment of a subset of the user base, while alternative B is deployed to the environment of another subset.
- Release
  - Once a deployed version of a system is released, the release engineers monitor telemetry data and crash logs to track the performance and quality of releases. Several frameworks and applications have been introduced for this.

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- The majority of these practices are classified by the paper as state of the practice, but state of the art practices are also mentioned.

What open challenges in release engineering does this publication mention?

- Branching and merging
  - No methodology or insight exists on how to empirically validate the best branching structure for a given organization or project, and what results in the smallest amount of merge conflicts.
  - Release engineers need to pay particular attention to conflicts and incompatibilities caused by evolving library and API dependencies.
- Building and testing
  - Speeding up CI might be the major concern of practitioners. This speed up can be achieved through predicting whether a code change will break the build, or by "chunking" code changes into a group and only compile and test each group once.
  - The concept of "green builds" slowly is becoming an issue, in the sense that frequent triggering of the CI server consumes energy.
  - Security of the release engineering pipeline in general, and the CI server in particular, also has become a major concern.
- Release
  - Qualitative studies are not only essential to understand the rationale behind quantitative findings, but also to identify design patterns and best practices for build systems.
    * How can developers make their builds more maintainable and of higher quality?
    * What refactorings should be performed for which build system anti-patterns?
  - Identification and resolution of build bugs, i.e., source code or build specification changes that cause build breakage, possibly on a subset of the supported platforms.
  - Basic tools have a hard time determining what part of the system is necessary to build.
  - Studies on non-GNU Make build systems are missing.
  - Apart from identifying bottlenecks, such approaches should also suggest concrete refactorings of

the build system specifications or source code.
- Infrastructure-as-code
  - Research on differences between infrastructure languages is lacking.
  - Best practices and design patterns for infrastructure-as-code need to be documented.
  - Qualitative analysis of infrastructure code will be necessary to understand how developers address different infrastructure needs.
  - Quantitative analysis of the version control and bug report systems can then help to determine which patterns were beneficial in terms of maintenance effort and/or quality.
- Deployment
  - More emperical studies can be done to answer question like this:
    - Is blue-green deployment the fastest means to deploy a new version of a web app?
    - Are A/B testing and dark launching worth the investment and risk?
    - Should one use containers or virtual machines for a medium-sized web app in order to meet application performance and robustness criteria?
    - If an app is part of a suite of apps built around a common database, should each app be deployed in a different container?
  - Better tools for quality assurance are required, to prevent showstopper bugs from slipping through and requiring re-deployment of a mobile app version (with corresponding vetting), these include:
    - Defect prediction (either file- or commit-based)
    - Smarter/safer update mechanisms
    - Tools for improving code review
    - Generating tests
    - Filtering and interpreting crash reports
    - Prioritization and triaging of defect reports
- Release
  - More research is needed on determining which code change is the perfect one for triggering the release of one of these releases, or whether a canary is good enough to be released to another data centre.
  - Question such as the following should be investigated:
    - Should one release on all platforms at the same time?
    - In the case of defects, which platform should receive priority?
    - Should all platforms use the same version numbering, or should that be feature-dependent?
    - Research on the continuous delivery and rapid releases from other systems should be explored.

What research gaps does this publication contain?

- As is common with surveys, it does not contain the state of the field today. More quantitive and qualitive research has been done, which can not possibly be included.

Are these research gaps filled by any other publications in this survey?

- An example of further research that expand on this study is [40]

7.8.2 The Impact of Switching to a Rapid Release Cycle on the Integration Delay of Addressed Issues

Reference: [40]

General information:

- Name of person extracting data: Nels Numan
- Date form completed (dd/mm/yyyy): 28/09/2018
- Publication title: The Impact of Switching to a Rapid Release Cycle on the Integration Delay of Addressed Issues
- Author information: Daniel Alencar da Costa, Shane McIntosh, Uira Kulesza, Ahmed E. Hassan
- Journal: 13th Working Conference on Mining Software Repositories (2016)

- Publication type: Conference paper
- Type of study: Emperical study

What practices in release engineering does this publication mention?

- To give a context to the study, the paper describes the concept of traditional releases, rapid releases, their differences, and how issue reports are structured.

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the practice. The paper describes common practices that were in use at the time of the publication.

What open challenges in release engineering does this publication mention?

- The study mentions that comparing systems with different release structures is difficult since one has to distinguish to what extent the results are due to the release strategy and which are due to intricacies of the systems or organization itself.

What research gaps does this publication contain?

- The main gap in this study is the specificity of the data. Only Mozilla has been considered, and external factors such as other organizational challenges which could have an effect on release time could not be included. More research that looks further into comparing this case to that of other organizations is needed.

Are these research gaps filled by any other publications in this survey?

-

Quantitative research publications:

- Study start date: Used data starts from 1999
- Study end date or duration: Used data ends in 2010
- Population description: The paper describes multiple steps to describe their data collection approach. The paper collected the date and version number of each Firefox release. Tags within the VCS were used to link issue IDs to releases. The paper discards issues that are potential false positives: IDs that have less five digits, issues that refer to tests instead of bugfixes, any potential ID that is the name of a file. Since the commit logs are linked to the VCS tags, the paper is able to link the issue IDs found within these commit logs to the releases that correspond to those tags.
- Method(s) of recruitment of participants: Firefox release history wiki and VCS logs
- Sample size: 72114 issue reports from the Firefox system (34673 for traditional releases and 37441 for rapid releases)
- Evaluation/measurement description: The paper aims to answer three research questions:
  - Are addressed issues integrated more quickly in rapid releases?
    - Approach: Through beanplots to compare the distributions, the paper first observes the lifetime of the issues of traditional and rapid releases. Next, it looks at the time span of the triaging, fixing, and integration phases within the lifetime of an issue.
  - Why can traditional releases integrate addressed issues more quickly?
    - Approach: the paper groups traditional and rapid releases into major and minor releases and study their integration delay through beanplots, Mann-Whiteney-Wilcoxon tests, Cliff's delta, and MAD.
  - Did the change in the release strategy have an impact on the characteristics of delayed issues?
    - Approach: the paper builds linear regression models for both release approaches. The paper firstly estimates the degrees of freedom that can be spent on the models. Secondly, they check for metrics that are highly correlated using Spearman rank correlation tests and perform a redundancy check to remove redundant metrics. The paper then assesses the fit of our models using the ROC area and the Brier score. The ROC area is used to evaluate the degree of discrimination achieved by the model. The Brier score is used to evaluate the accuracy of probabilistic predictions. The used metrics include reporter experience, resolver experience,

issue severity, issue priority, project queue rank, number of impacted files and fix time. A full list of metrics can be found in Table 2 of the paper.

- Outcomes:
  - Are addressed issues integrated more quickly in rapid releases?
    - Results: There is no significant difference between traditional and rapid releases regarding issue lifetime. Results:
  - Why can traditional releases integrate addressed issues more quickly?
    - Results: Minor-traditional releases tend to have less integration delay than major/minor-rapid releases.
  - Did the change in the release strategy have an impact on the characteristics of delayed issues?
    - Results: The models achieve a Brier score of 0.05- 0.16 and ROC areas of 0.81-0.83. Traditional releases prioritize the integration of backlog issues, while rapid releases prioritize the inte- gration of issues of the current release cycle.
- Limitations: Defects in the tools that were developed to perform the data collection and evaluation could have an effect on the outcomes. Furthermore, the way that issue IDs are linked to releases may not represent the total addressed issues per release. The results cannot be generalized as the evaluation was solely done on the Firefox system.
- Future research: Further research can look into applying the same evaluation strategy to other organizations that switched from traditional to rapid release.

Notes:

7.8.3 An Empirical Study of Delays in the Integration of Addressed Issues

Reference: [39]

General information:

- Name of person extracting data: Nels Numan
- Date form completed (dd/mm/yyyy): 29/09/18
- Publication title: An Empirical Study of Delays in the Integration of Addressed Issues
- Author information: Daniel Alencar da Costa, Surafel Lemma Abebe, Shane McIntosh, Uira Kulesza, Ahmed E. Hassan
- Journal: 2014 IEEE International Conference on Software Maintenance and Evolution
- Publication type: Conference paper
- Type of study: Emperical study

What practices in release engineering does this publication mention?

- This publication discusses the usage of issue tracking systems, and what the term issue means to form a context around the study.

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- State of the practice.

What open challenges in release engineering does this publication mention?

- The results based on the investigated open source projects may not be generalizable and replication of the study is required on a larger set of projects to form a more general conclusion. Another challenge is finding metrics that are truly correlated with the integration delay of issues.

What research gaps does this publication contain?

- Please see last question.

Are these research gaps filled by any other publications in this survey?

- [40]

Quantitative research publications:

- Study start date:
- Used data start dates:
  - ArgoUML: 18/08/2003
  - Eclipse: 03/11/2003
  - Firefox: 05/06/2012
- Used data end dates:
  - ArgoUML: 15/12/2011
  - Eclipse: 12/02/2007
  - Firefox: 04/02/2014
- Population description:
- Method(s) of recruitment of participants: The data was collected from both ITSs and VCSs of the studied systems.
- Sample size: 20,995 issues from ArgoUML, Eclipse and Firefox projects
- Evaluation/measurement description:
  - How long are addressed issues typically delayed by the integration process?
    - Approach: models are created using metrics from four dimensions: reporter, issue, project, and history. Please refer to Table 2 in the paper for all of the metrics considered. The models are trained using the random forest technique. Precision, recall, F-measure, and ROC area are used to evaluate the models.
- Outcomes:
  - How long are addressed issues typically delayed by the integration process?
    - Addressed issues are usually delayed in a rapid release cycle. Many delayed issues were addressed well before releases from which they were omitted. Many delayed issues were addressed well before releases from which they were omitted.
  - Can we accurately predict when an addressed issue will be integrated?
    - The prediction models achieve a weighted average precision between 0.59 to 0.88 and a recall between 0.62 to 0.88, with ROC areas of above 0.74. The models achieve better F-measure values than Zero-R.
  - What are the most influential attributes for estimating integration delay?
    - The integrator workload has a bigger influence on integrator delay than the other attributes. Severity and priority have little influence on issue in- tegration delay.
- Limitations: See open challenges.
- Future research: See open challenges.

Notes:

7.8.4 Towards Definitions for Release Engineering and DevOps

Reference: [48]

General information:

- Name of person extracting data: Nels Numan
- Date form completed (dd/mm/yyyy): 30/09/2018
- Publication title: Towards Definitions for Release Engineering and DevOps
- Author information: Andrej Dyck, Ralf Penners, Horst Lichter
- Journal:
- Publication type:
- Type of study: Survey

What practices in release engineering does this publication mention?

- This paper talks about approaches to improve the collaboration between development and IT operations teams, in order to streamline software engineering processes. The paper defines for release engineering

and devops.

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- Not applicable.

What open challenges in release engineering does this publication mention?

- The paper mentions that creating a definition which is uniform and valid for many situations is difficult to find and that further research is needed.

What research gaps does this publication contain?

- This paper aims to form a uniform definition for release engeneering and devops, in collaboration with experts. It is unclear how many experts were consulted for this definition, and more consultations and research could be done to further improve the definition.

Are these research gaps filled by any other publications in this survey?

-

Quantitative research publications:

- Study start date:
- Study end date or duration:
- Population description:
- Method(s) of recruitment of participants:
- Sample size:
- Evaluation/measurement description:
- Outcomes:
- Limitations:
- Future research:

Notes:

7.8.5 Continuous deployment of software intensive products and services: A systematic mapping study

Reference: [106]

General information:

- Name of person extracting data: Nels Numan
- Date form completed (dd/mm/yyyy): 30/09/18
- Publication title: Continuous deployment of software intensive products and services: A systematic mapping study
- Author information: Pilar Rodrígueza, Alireza Haghighatkhaha, Lucy Ellen Lwakatarea, Susanna Teppolab, Tanja Suomalainenb, Juho Eskelib, Teemu Karvonena, Pasi Kuvajaa, June M. Vernerc, Markku Oivoa
- Journal:
- Publication type:
- Type of study: Semantic study

What practices in release engineering does this publication mention?

- This paper discussed the developments of continuous development over the years until June 2014. This paper has performed a semantic study to identify, classify and analyze primary studies related to continuous development. The paper has found the following major points:
  - Almost all primary studies make reference in one way or another to accelerate the releae cycle by shortening the release cadence and turning it into a continuous flow.

  - Some reviewed publications claim that accelerating the release cycle can make it harder to perform re-engineering activities.
  - CD challenges and changes traditional planning towards continuous planning in order to achieve fast and frequent releases.
  - Tighter integration between planning and execution is required in order to achieve a more holisitic view on planning in CD.
  - It is important for the engineering and QA teams to ensure backward compatibility of enhancements, so that users perceive only improvements rather than experience any loss of functionality.
  - Code change activities tend to focus more on bug fixing and maintenance than functional- ity expansion
  - The architecture must be robust enough to allow the organization to invest its resources in offensive initiatives such as new functionalitity, product enhancements and innovation rather than defensive efforts such as bugfixes.
  - A major challenge in CD is to retain the balance between speed and quality. Some approaches reviewed by this study propose a focus on measuring and monitoring source code and architectural quality.
  - To avoid issues such as duplicated testing efforts and slow feedback loops it is important to make all testing activities transparent to individual developers.

What open challenges in release engineering does this publication mention?

- Continuous and rapid experimentation is an emerging research topic with many possibilities for future work. This is why it's important to keep up with the newly contributed studies and add them to future reviews to compare their findings.

What research gaps does this publication contain?

-

Notes:

7.8.6 Frequent Releases in Open Source Software: A Systematic Review

Reference: [34]

General information:

- Name of person extracting data: Nels Numan
- Date form completed (dd/mm/yyyy): 30/09/18
- Publication title: Frequent Releases in Open Source Software: A Systematic Review
- Author information: Antonio Cesar Brandão Gomes da Silva, Glauco de Figueiredo Carneiro, Fernando Brito e Abreu and Miguel Pessoa Monteiro
- Journal: Information
- Publication type: Journal
- Type of study: Survey

What practices in release engineering does this publication mention?

- This paper discussed the developments of continuous development over the years. This paper has performed a semantic study to identify, classify and analyze primary studies related to continuous development. The paper finds:
  - Two main motivations for the implementation of frequent software releases in the context of OSS projects, which are the project attractiveness/increase of participants and maintenance and increase of market share
  - Four main strategies are adopted by practitioners to implement frequent software releases in the context of OSS projects: time-based release, automated release, test-driven development and continuous delivery/deployment.

  - The main positive points associated to rapid releases are: quick return on customer needs, rapid delivery of new features, quick bug fixes, immediate release security patches, increased efficiency, entry of new collaborators, and greater focus on quality on the part of developers and testers.
  - The main negative points assocaited to rapid releases are reliability of new versions, increase in the "technical debt", pressure felt by employees and community dependence.

Are these practices to be classified under dated, state of the art or state of the practice? Why?

- The practices discussed are a combination of state of the art and state of the practice approaches.

What open challenges in release engineering does this publication mention?

- A meta-model for the mining of open source bases in view of gathering data that leads to assessment of the quality of projects adoping the frequent release approach.

What research gaps does this publication contain?

-

Are these research gaps filled by any other publications in this survey?

-

Chapter 8

Code Review

8.1 Review protocol

This section describes the review protocol used for the systematic review presented in this section. The protcol has been set up using Kitchenham's method as described by Kitchenham et al. [74].

8.1.1 Research questions

The goal of the review is to summarize the state of the art and identify future challenges in the code review area. The research questions are as follows:

- **RQ1**: *What is the state of the art in the research area of code review?* This question focusses on topics that are researched often, the results of that research, and research methods, tools and datasets that are used.
- **RQ2**: *What is the current state of practice in the area of code review?* This concerns tools and techniques that are developed and used in practice, by open source projects but also by commercial companies.
- **RQ3**: *What are future challenges in the area of code review?* This concerns both research challenges and challenges for use in practice.

8.1.2 Search process

The search process consists of the following:

- A Google Scholar search using the search query *"modern code review" OR "modern code reviews"*. The results list will be sorted by decreasing relevance by Google Scholar and will be considered by us in order.
- A general Google search for non-scientific reports (e.g., blog posts) and implemented code review tools. For this search queries *code review* and *code review tools* are used, respectively. The result list will be considered in order.
- All papers in the initial seed provided by the course instructor will be considered.
- All papers referenced by already collected papers will be considered.

From now on, all four categories listed above in general will be called *resource*.

8.1.3 Inclusion criteria

From the scientific literature, the following types of papers will be considered:

Papers researching recent code review

- concepts,
- methodologies,
- tools and platforms,
- and experiments concerning the preceding.

From non-scientific resources, all resources discussing recent tools and techniques used in practice will be considered.

8.1.4 Exclusion criteria

Resources published before 2008 will be excluded from the study.

8.1.5 Primary study selection process

We will select a number of candidate resources based on the criteria stated above. For each resource, each person participating in the review can select it as a candidate.

From all candidates, resource will be selected that will actually be reviewed. This can also be done by each person participating in the review. All resources that are candidates but are not selected for actual review must be explicitly rejected, with accompanying reasoning, by at least two persons participating in the review.

8.1.6 Data collection

The following data will be collected from each considered resource:

- Source (for example, the blog website or specific journal)
- Year published
- Type of resource
- Author(s) and organization(s)
- Summary of the resource of a maximum of 100 words
- Data for answering **RQ1**:
  - Sub-topic of research
  - Research method
  - Used tools
  - Used datasets
  - Research questions and their answers
- Data for answering **RQ2**:
  - Tools used
  - Company/organization using the tool
  - Evaluation of the tool
- Data for answering **RQ3**:
  - Future research challenges posed

All data will be collected by one person participating in the review and checked by another.

8.2 Candidate resources

In this section, all candidates that are collected using the described search process are presented. The in survey column in the tables below indicates whether the paper has been included in the survey in the end or if it has been excluded for some reason. If it has been excluded, the reason is stated along with the paper summary.

8.2.1 Initial seed

These following table lists all initial seed papers provided by the course intructor. They are listed in alphabetical order of the first author's name, and then by publish year.

First author	Year	Reference	In survey? (Y/N)
Bacchelli, A.	2013	[7]	
Beller, M.	2014	[15]	
Bird, C.	2015	[23]	
Fagan, M.	2002	[52]	
Gousios, G.	2014	[56]	
McIntosh, S.	2014	[89]	

8.2.2 Google Scholar

The following table lists all candidates that have been collected through the Google Scholar search described in the search process. They are listed in alphabetical order of the first author's name, and then by publish year. Note that as described in the search process section, papers in the search are considered in order.

First author	Year	Reference	In survey? (Y/N)
Baysal, O.	2016	[12]	
Thongtanunam, P.	2015	[117]	
Thongtanunam, P.	2016	[116]	
Xia, X.	2015	[123]	
Zanjani, M. B.	2016	[126]	

8.2.3 By reference

The following table lists all candidates that have been found by being referenced by another paper we found. They are listed in alphabetical order of the first author's name, and then by publish year.

First author	Year	Reference	Referenced by	In survey? (Y/N)
Baum	2016	[10]		
Baum	2017	[9]		
Baysal	2013	[13]		
Bosu	2013	[27]		
Ciolkowski	2003	[35]		
Czerwonka	2015	[42]		

8.3 Paper summaries

This section contains summaries of all papers included in the survey. They are listed in alphabetical order of first author name, and then by year published.

8.3.1 Expectations, outcomes, and challenges of modern code review

Reference: [7]

This paper describes research about the goals and actual effects of code reviews. Interviews and experiments have been done with people in the programming field.

One of the main conclusions is that the main effect of doing code reviews is that everyone involved understands the code better. This is opposed to what the goal of code reviews is generally: discovering errors.

8.3.2 A Faceted Classification Scheme for Change-Based Industrial Code Review Processes

Reference: [10]

The broad research questions treated in this article are: How is code review performed in industry today? Which commonalities and variations exist between code review processes of different teams and companies? The article describes a classification scheme for change-based code review processes in industry. This scheme is based on descriptions of the code review processes of eleven companies, obtained from interviews with software engineering professionals that were performed during a Grounded Theory study.

8.3.3 The Choice of Code Review Process: A Survey on the State of the Practice

Reference: [9]

This paper, published in 2017, is trying to answer 3 RQs. Firstly, how prevalent is change-based review in the industry? Secondly, does the chance that code review remains in use increase if code review is embedded into the process (and its supporting tools) so that it does not require a conscious decision to do a review? Thirdly, are the intended and acceptable levels of review effects a mediator in determining the code review process?

8.3.4 The influence of non-technical factors on code review

Reference: [13]

8.3.5 Investigating technical and non-technical factors influencing modern code review

Reference: [12]

8.3.6 Modern code reviews in open-source projects: Which problems do they fix?

Reference: [15]

It has been researched what kinds of problems are solved by doing code reviews. The conclusion is that 75% are improvements in evolvability of the code, and 25% in functional aspects.

It has also been researched which part of the review comments is actually followed up by an action, and which part of the edits after a review are actually caused by review comments.

8.3.7 Lessons learned from building and deploying a code review analytics platform

Reference: [23]

A code review data analyzation platform developed and used by Microsoft is discussed. It is mainly presented what users of the system think of it and how its use influences development teams. One of the conclusions is that in general, the platform has a positive influence on development teams and their products.

8.3.8 Impact of peer code review on peer impression formation: A survey

Reference: [27]

8.3.9 Software Reviews: The State of the Practice

Reference: [35]

To investigate how industry carries out software reviews and in what forms, this paper con- ducted a two-part survey in 2002, the first part based on a national initiative in Germany and the second involving companies world- wide. Additionally, this paper also include some fundenmental concepts of code review, such as functionalities of code review.

8.3.10 Code reviews do not find bugs: how the current code review best practice slows us down

Reference: [42]

As code review has many uses and benefits, the authors hope to find out whether the current code review methods are sufficiently efficient. They also research whether other methods may be more efficient. With experience gained at Microsoft and with support of data, the authors posit (1) that code reviews often do not find functionality issues that should block a code submission; (2) that effective code reviews should be performed by people with a specific set of skills; and (3) that the social aspect of code reviews cannot be ignored.

8.3.11 Design and code inspections to reduce errors in program development

Reference: [52]

This paper describes a method to thoroughly check code quality after each step of the development process, in a heavyweight manner. It does not really concern agile development.

The authors state that these methods do not affect the developing process negatively, and that they work well for improving software quality.

8.3.12 An exploratory study of the pull-based software development model

Reference: [56]

This article focusses on how much pull requests are being used and how they are used, focussing on GitHub. For example, it is concluded that pull-request are not being used that much, that pull-requests are being merged fast after they have been submitted, and that a pull request not being merged is most of the time not caused by technical errors in the pull-request.

8.3.13 The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects

Reference: [89]

This paper focusses on the influence of doing light-weight code reviews on software quality. In particular, the effect of review coverage (the part of the code that has been reviewed) and review participation (a measure for how much reviewers are involved in the review process) are being assessed.

It turns out that both aspects improve software quality when they are higher. Review participation is the most influential. According to the authors there are other aspects, which they have not looked into, that are of significant importance for the review process.

8.3.14 Who should review my code? A file location-based code-reviewer recommendation approach for modern code review

Reference: [117]

8.3.15 Revisiting code ownership and its relationship with software quality in the scope of modern code review

Reference: [116]

8.3.16 Who should review this change?: Putting text and file location analyses together for more accurate recommendations

Reference: [123]

8.3.17 Automatically recommending peer reviewers in modern code review

Reference: [126]

Chapter 9

Runtime and Performance Analytics

In this chapter, we discuss the field of performance and runtime analytics. This chapter does not cover the entire field because it is too broad. Using Kitchenham's method [76], we have narrowed down the scope of this survey.

For inspiration, we started reading five recent papers on runtime and performance analytics published at top conferences. These five were selected because the papers handle the software side of performance and runtime analytics which is more in line with the other chapters of this book. However, focussing on only software, the field is still very broad. Currently, we are leaning towards a focus on performance and runtime analytics literature regarding the Android platform. As we still are at the start of this research, we might deviate from this initial focus.

We have gathered a few other papers (excluding the five initial papers) to find out if this field is suited for this survey. These papers can be found in Figure INSERT FIGURE NUMBER HERE. To get relevant papers, we used the following keywords: Android, performance, runtime, reliability, synchronization, security, monitoring. Furthermore, we only retrieved papers published at top venues, which we list here:

- ACM Transactions on Software Engineering Methodology (TOSEM),
- Empirical Software Engineering (EMSE),
- IEEE Transactions on Software Engineering (TSE),
- Information and Software Technology (IST),
- Journal of Systems and Software (JSS),
- ACM Computing Surveys (CSUR),
- Foundations of Software Engineering (SIGSOFT FSE),
- International Conference on Automated Software Engineering (ASE),
- Working Conference on Mining Software Repositories (MSR)
- Symposium on Operating Systems Design and Implementation (OSDI)

9.1 Week 1

Because we consider the five starting papers to be our inspiration, we have chosen to briefly describe these papers by giving some basic metrics about them (citations), summarizing them and by adding a few notes about them. This is our initial work that we would like to expand on in the coming weeks.

9.1.1 Charting the API minefield using software telemetry data

In this paper, researchers used software telemetry data from mobile application crashes. With heuristics, they separated the API calls from application calls so they can analyze what the most common causes for

crashes are. Top crash causes are: memory exhaustion, race conditions or deadlocks, and missing resources. A significant percentage was not suitable for analysis as these crashes were associated with generic exceptions (10%). They performed a literature search to find solutions to the problems that cause the crashes. For each crash cause category, an implementation recommendation is made. More specific exceptions, non-blocking algorithms, and default resources can eliminate the most frequent crashes. They also suggest that development tools like memory analyzers, thread debuggers, and static analyzers can prevent many application failures. They also propose features of execution platforms and frameworks related to process and memory management that could reduce application crashes.

Remarks

- Among the papers that refer to this paper or are referenced by this paper there are four papers that share the topic of crash data on mobile platforms that have been publiced to top software engineering venues [].
- The paper seems to be quite discerning as they evaluate their methods and reason about the threats to validity.

9.1.2 Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring

The mobile applications market continues to grow and many applications are available. It is important for developers that their application keeps working and that crashes are fixed as fast as possible to keep up with competitors. However, the mobile market is complex as for end users there are endless configurations of application versions, mobile hardware and user input sequences. Therefore, it is difficult to reproduce software crashes under the same context and conditions that triggered the observed crash. This is why the researchers developed MoTiF which uses machine learning to reproduce the steps the end users take before the app crashes on the end user's phone and generates a test suite. MoTiF also uses the crowd to validate whether the generated test suite truly reproduces the observed crash.

Remarks

- The datasets used for the research are a bit questionable. One is based on simply performing a large amount of random event on the app, the other dataset is created by letting a group of 10 student try to crash the app in one hour.
- Only 5 different apps have been tested.
- Contains reference to "Charting the API minefield using software telemetry data".

9.1.3 An exploratory study on faults in web api integration in a large-scale payment company

This research explores what the implications of web API faults are, what the most common web API faults are and best practices for API design. The faults in API integration can be grouped in 11 causes: invalid user input, missing user input, expired request data, invalid request data, missing request data, insufficient permissions, double processing, configuration, missing server data, internal and third party. Most faults can be attributed to the invalid or missing request data, and most API consumers seem to be impacted by faults caused by invalid request data and third party integration. Furthermore, API consumers most often use official API documentation to implement an API correctly, followed by code examples. The challenges of preventing runtime problems are the lack of implementation details, insufficient guidance on certain aspects of the integration, insufficient understanding of the impact of problems, and missing guidance on how to recover from errors.

Remarks

- Easy to read
- Paper only considers a single API

- Survey only has 40 responses

9.1.4 Search-based test data generation for SQL queries

SQL queries should be tested as thoroughly as program code. However, it is hard to generate test data for testing. Other researchers proposed viewing this problem as a constraint solving problem, so test data could be generated with a SAT-solver. However, strings are not supported by current SAT-solver tools and it is a complex task to translate a query to a satisfiability problem. In this research, the test generation problem is treated as a search-based problem. They use random search, biased random search and genetic algorithms (GA) to generate the data. The methods are combined in a tool called EvSQL and the tool is tested on more than 2000 queries. The GA method is the best and is able to cover a little over 98% of the queries.

Remarks

- Easy to read
- Utilizes queries of 4 different systems
- Generation of test data for SQL questies implies easier generation of unit- regression- and integration tests for SQL queries.

9.1.5 Anomaly detection using program control flow graph mining from execution logs

The paper attempts to diagnose distributed applications. For this purpose they mine templates and their sequences from exedcution logs, from this information they create a control flow graph. The main cause of failures identified: making an API request to another application. This results in many new calls to other services or even other applications. This flow gets interrupted at some point. So when the top level API is not working, they want to show where it goes wrong. In earlier work, primarily metrics and logs were used to find the cause. However these approaches struggled with many benign warnings or errors in healthy state or faults do not manifest as errors. Manually checking a transaction flow is also very hard. Instead, templates are used as print statements from the source code. These represent the nodes, the edges are the flows. This approach imposes two major challenges. One, mining print statements is hard because parameters are different in every log. Two, flows can happen at the simultaneously. The paper tries to solve these challenges by applying a join on two print statements if the statements are preceded and followed by approximately the same steps.

Remarks

- Has a presentation on YouTube
- Difficult to read

9.2 Week 2

Because we are still working on the exact scope of the survey as well as the lay-out of the chapter, we have chosen to temporarily divide the papers by week. This will be changed later on. A more suitable focus for this survey would be the Energy vs performance sub-domain of runtime and performance analytics. To explore this domain we have summarized some initial papers.

The survey on performance vs energy efficiency focuses on the following research questions: **RQ1** What is the current state of art? **RQ2** What is the current state of practice? **RQ3** What are the challenges of the future work?

To answer these questions three papers are selected to form the basis of this literature survey:

- Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2014. Characterizing and detecting performance bugs for smartphone applications. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). ACM, New York, NY, USA, 1013-1024. DOI: https://doi.org/10.1145/2568225.2568229
- Rui Pereira, Pedro Simão, Jácome Cunha, and João Saraiva. 2018. jStanley: placing a green thumb on Java collections. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018). ACM, New York, NY, USA, 856-859. DOI: https://doi.org/10.1145/3238147.3240473
- Stefanos Georgiou, Maria Kechagia, Panos Louridas, and Diomidis Spinellis. 2018. What are your programming language's energy-delay implications?. In Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18). ACM, New York, NY, USA, 303-313. DOI: https://doi.org/10.1145/3196398.3196414

By looking into the state of programming languages in regards to energy performance, the state of the art will be determined, thus answering RQ1. For the current state of practice (RQ2) literature on the topic of energy efficiency in Android applications will be used. From both topics the challenges and related work will be used for answering RQ3.

Running list of domain keywords: Programming Languages, Energy-Delay-Product, Energy-Efficiency, Empirical study, performance bug, testing, static analysis, Green Software, Energy-aware Software, JCF

9.2.1 What Are Your Programming Language's Energy-Delay Implications?

Motivated by the lack of studies that investigate the energy consumption of software applications compared to the number of studies in the energy efficiency of hardware, the researchers set out to investigate the run-time performance of commonly used programming tasks in different languages on different platforms. The paper contributes by giving a customized and extended data set that can be used as a benchmark for similar studies, a set of publicly available tools for measuring the Energy Delay Product (EDP) of various programming tasks implemented in different programming languages, an empirical study on programming language EDP implications, by using different types of programming tasks and software platforms, and a programming language-based ranking catalogue, in the form of heat maps, where developers can find which programming language to pick for particular tasks and platforms; when energy or run-time performance are important. The research questions which are answered are as follows: Which programming languages are the most EDP efficient and inefficient for particular tasks? Which types of programming languages are, on average, more EDP efficient and inefficient for each of the selected platforms (i.e. server, laptop and embedded system)? How much does the EDP of each programming language differ among the selected platforms? To answer these questions the Rosetta Code Repository, a publicly available repository for programming tasks, is used. It offers 868 tasks, 204 draft tasks and has implementations in 675 programming languages. The results of the paper are that for most tasks the compiled programming languages outperform the interpreted ones.

Keywords: Programming Languages; Energy-Delay-Product; Energy-Efficiency

9.2.2 Characterizing and Detecting Performance Bugs for Smartphone Applications

Bugs can cause significant performance degradation, which in turn may lead to losing the competitive edge for the application. The paper is motivated by people having little understanding for performance bugs and the lack of effective techniques to fight these bugs. In the paper the questions are researched what the common types of performance bugs are in Android applications, and what impact they have on the user experience (RQ1), how the performance bugs manifest themselves and if their manifestation needs special input (RQ2), if performance bugs are more difficult to debug and fix compared to non-performance bugs and what information or tools can help with that (RQ3) and if there are common causes of performance

bugs, and if patterns can be distilled to facilitate performance analysis and bug detection (RQ4). Answering these questions leads to the paper making two major contributions: The first empirical study of real-world performance bugs in smartphone applications. The findings can help understand characteristics of performance bugs in smartphone applications, and provide guidance to related research. The implementation of a static code analyzer, PerfChecker, which successfully identified performance optimization opportunities in 18 popular Android applications. The selected Android applications needed to have more than 10.000 downloads and own a public bug tracking system. Furthermore there should be at least hundreds of code revisions. These criteria provide an indicator of the popularity and maturity of the selected applications. At first 29 Android applications were selected, with PerfChecker successfully detecting 126 matching instances of the bug patterns in 18 of these applications. Of these detected 126 matching instances of performance bug patterns, 68 were quickly confirmed by developers as previously unknown issues that affect application performance.

Keywords: Empirical study, performance bug, testing, static analysis.

9.2.3 jStanley: Placing a Green Thumb on Java Collections

In this short paper the tool jStanley is presented. With the help of this tool developers can obtain information and suggestions on the energy efficiency of their Java code. jStanley is available as Eclipse plugin. In a preliminary evaluation jStanley shows energy gains between 2% and 17%, and a reduction in execution time between 2% and 13%.

Keywords: Green Software, Energy-aware Software, JCF, Eclipse Plugin

9.2.4 A Study on the Energy Consumption of Android App Development Approaches

In this study, an analysis is given of the energy consumption of Android app according to which development method was used to create them. They look mainly at the difference between programming languages and their respective frameworks. They measured across multiple devices, which presented little difference between them. They also rewrote some app to use a hybrid framework in the hopes of improving the performance vs Energy consumption balance and they report a non-negligible improvement.

Keywords: Android, runtime, performance (search keywords, the paper itself did not contain keywords)

Chapter 10

App Store Analytics

10.1 Motivation

In the year 2008, the first app stores became available. These stores have grown rapidly in size since then, with over 3 million apps in the Google Play store alone at the time of writing [REFERENCE]. These app stores together with the large user bases associated with them provide software developers and researchers with valuable data. The process of exploiting this data from app stores to gain valuable insights is what we would call "App Store Analytics". Because apps have not been around for a long time the research field of App Store Analytics is still very young. However, because apps are used so much nowadays it plays an important role in the field of Software Engineering. Therefore, to get an overview of the current state of this young research field this chapter(?) is devoted as a survey on the field of App Store Analytics. We present three research questions to structure this survey:

- **RQ1** Current state of the art in software analytics for App Store Analytics:
  - Topics that are being explored.
  - Research methods, tools, and datasets being used.
  - Main research findings, aggregated.
- **RQ2** Current state of practice in software analytics for App Store Analytics:
  - Tools and companies creating/employing them.
  - Case studies and their findings.
- **RQ3** Open challenges and future research required.

10.2 Research protocol

TODO: here are just ideas of what I'm doing but they should be properly written

The research protocol is divided into two important parts: the articles search process and the article selection process. In the following paragraphs, both processes will be explained. [Refer to Kitchenham?]

10.2.1 Search queries (Article search process)

Our initial seed of the papers came from the survey of the field of App Store Analytics by Martin et al. [TODO: REFER to survey] and after that we used the keywords **apps**, **app store**, **app store analytics** and **app store mining** to search for other relevant papers on Google Scholar, ACM, IEEE Xplore and pages of individual journals (CSUR, TSE, EMSE, JSS, TOSEM, IST) and conferences (ICSE, FSE, ASE, MSR, OSDI). From the results only articles with relevant titles were selected and added to the list for consideration.

TODO: Include a table with journals/conferences including their full names

10.2.2 Article selection

In order to retain only the most relevant papers to answer the research questions, we devised a composed metric that takes into account the number of citations and the year the paper was published. Taking these elements the scoring scheme is the following: Citations (C): Year of publication (Y): The metric is computed as follows: Inclusion_metric = C (0.5) * Y (0.5)

For each paper the previously mentioned metric was calculated and the top 30 were selected, discarding the rest.

10.2.2.1 Inclusion criteria

- The paper was published in well established journal or conference.
- Title or abstract of the paper mentions app stores, mining from app stores or app store analytics.
- Tha paper was published in 2010 or later.

10.2.2.2 Exclution criteria

- The paper has at least 10 citations on Google Scholar.
- The paper focuses on mobile app development or is an analysis of arbitrary selection of apps and does not extend to the app stores as a whole.

10.2.3 Fact extraction

Taking into consideration the example presented by Kitchenham et al in [reference], the following data were extracted from each of the papers: - Source (journal or conference) - Complete reference - Main topic area - Authors information (full names, institution, and country) - Summary (research questions and answers) - Research question / issue - MORE?

Each one of the team members was in charge of reviewing and extracting the data of a set of papers . Then, the extracted data was checked by another member. The allocation of team members to the papers was random, equally splitting the workloads.

10.3 Answers

- **RQ1** Current state of the art in software analytics for App Store Analytics
- **RQ2** Current state of practice in software analytics for App Store Analytics
- **RQ3** Open challenges and future research required

10.4 Paper extracted data

10.4.1 API change and fault proneness: A threat to the success of Android apps

Source: Conference ESEC/FSE'17 Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering

Main topic area: using user feedback/reviews, API changes

Authors information (full names, institution, and country): - Mario Linares-Vásquez - Universidad de los Andes, Colombia - Gabriele Bavota - University of Sannio, Italy - Carlos Bernal-Cárdenas - Universidad Nacional de Colombia, Colombia - Massimiliano Di Penta - University of Sannio, Italy - Rocco Oliveto - University of Molise, Italy - Denys Poshyvanyk - College of William and Mary, USA

The paper presents an empirical study that aims to corroborate the relationship between the fault and change-proneness of APIs and the degree of success of Android apps measured by their user ratings. For this, the authors selected a sample of 7,097 free Android apps from the Google Play Market and gathered information of the changes and faults that the APIs used by them presented. Using this data and statistical tools such as box-plots and the Mann-Whitney test, two main hypotheses were analyzed. The first hypothesis tested the relationship between fault-proneness (number of bugs fixed in the API) and the success of an app. The second tested the relationship between change-proneness (overall method changes, changes in method signatures and changes to the set of exceptions thrown by methods) and the success of an app. Finally, although no causal relationships between the variables can be assumed, the paper found significant differences of the level of success of the apps taking into consideration the change and fault-proneness of the APIs they use.

Research question/issue: relationship between fault- and change-proneness of APIs and the degree of success in Android apps.

10.4.2 What would users change in my app? summarizing app reviews for recommending software changes

Source: Proceeding FSE 2016 Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering

Main topic area: using user feedback/reviews

Authors information (full names, institution, and country): - Andrea Di Sorbo - University of Sannio, Italy - Sebastiano Panichella - University of Zurich, Switzerland - Carol V. Alexandru - University of Zurich, Switzerland - Junji Shimagaki - Sony Mobile Communications, Japan - Corrado A. Visaggio - University of Sannio, Italy - Gerardo Canfora - University of Sannio, Italy - Harald C. Gall - University of Zurich, Switzerland

Summary (research questions and answers): The paper proposes a new approach for analyzing App Store user reviews, deriving insights from them. The presented solution has two components. First, the User Reviews Model (URM) that enable the classification of users intentions (e.g., UI improvements, bug fixes, etc.). Second, the Summarizer of User Review Feedback (SURF). A tool that, by leveraging the URM, is capable of generating summaries of users feedback. After evaluating the proposed approach, TODO

Research question/issue: there is no approach that is able to do, at the same time, the following: (i) determine for a large number of reviews the specific topic discussed in the review (e.g., UI improvements, security/licensing issues, etc.), (ii) identify the maintenance task to perform for addressing the request stated in the review (e.g., bug fixing, feature enhancement, etc.), and (iii) present such information in the form of a condensed, interactive and structured agenda of recommended software changes, which is actionable for developers. [Reference paper]

10.4.3 App Store, Marketplace, Play! An Analysis of Multi-Homing in Mobile Software Ecosystems

Source: Proceedings of the Fourth International Workshops on Software Ecosystems **Main topic area:** App store ecosystem

Authors information (full names, institution, and country): Sami Hyrynsalmi, University of Turku, Finland

Tuomas Mäkilä, University of Turku, Finland

Antero Järvi, University of Turku, Finland

Arho Suominen, VTT Technical Research Centre of Finland, Finland

Marko Seppänen, Tampere University of Technology, Finland

Timo Knuutila, University of Turku, Finland

Summary (research questions and answers): Multi-homing is not used by many developers, where multi-homing is the strategy of releasing your application to multiple platforms. An analysis of Google Play, App Store and Windows Phone Store shows that not many developers use this strategy. Next to this, the paper found that the type and popularity of apps does not differ from those that use a single-homing strategy.

Research question/issue: Analysis of multi-homing in different app stores. How much is it used by developers and is there a difference in popularity?

10.4.4 A systematic literature review: Opinion mining studies from mobile app store user reviews

*Source:** Journal of Systems and Software

Main topic area: Opinion Mining and Requirement Engineering

Authors information (full names, institution, and country): Necmiye Genc-Nayebi, École de Technologie Supérieure (ETS) - Université du Québec, Canada Dr. Alain Abran, École de Technologie Supérieure (ETS) - Université du Québec, Canada

Summary (research questions and answers): TODO: summary

Research question/issue: What are the proposed solutions for mining online opinions in app store user reviews, challenges and unsolved problems in the domain, new contributions to software requirements evolution and future research direction.

10.4.5 The Impact of API Change and Fault-Proneness on the User Ratings of Android Apps

TODO: template

The paper by Bavota et al. aims to find empirical evidence supporting the success of apps and the relationship with change- and fault-proneness of the underlying APIs, where the success of the app is measured by its user rating. They performed two case studies to find quantitative evidence using 5848 free Android apps as well as an explanation for these results doing a survey with 45 professional Android developers. The quantitative case study was done by comparing the user ratings to the number of bug fixes and changes in the API that an app uses. They found that apps with a high user rating are significantly less change- and fault-prone than APIs used by apps with a low user rating. In the second case study the paper found that most of the 45 developers observed a direct relationship between the user ratings of apps and the APIs those apps use.

10.4.6 How can i improve my app? Classifying user reviews for software maintenance and evolution

TODO: template

The most popular apps in the app stores (such as Google Play or App Store) receive thousands of user reviews per day and therefore it would be very time demanding to go through the reviews manually to obtain relevant information for the future development of the apps. This paper uses a combination of Natural Language Processing Sentiment Analysis and Text Analysis to extract relevant sentences from the reviews and to classify them into the following categories: Information Seeking, Information Giving, Feature Request, Problem Discovery, and Others. The results show 75% precision and 74% recall when classifier (J48 using data from NLP+SA+TA) is trained on 20% of the data (1421 manually labeled sentences from reviews of seven different apps) and the rest is used for testing. The paper also states that the results do not differ in a statistically significant manner when a different classifier is used and shows that precision and recall can be further improved by increasing the size of the data set.

Chapter 11

Final Words

We have finished a nice book on Software Analytics.

[1] Abate, P. and Cosmo, R.D. 2011. Predicting upgrade failures using dependency analysis. *2011 IEEE 27th international conference on data engineering workshops* (Apr. 2011).

[2] Abate, P. et al. 2009. Strong dependencies between software components. *2009 3rd international symposium on empirical software engineering and measurement* (Oct. 2009).

[3] Abdalkareem, R. et al. 2017. Why do developers use trivial packages? An empirical case study on npm. *Proceedings of the 2017 11th joint meeting on foundations of software engineering - ESEC/FSE 2017* (2017).

[4] Adams, B. and McIntosh, S. 2016. Modern release engineering in a nutshell–why researchers should care. *Software analysis, evolution, and reengineering (saner), 2016 ieee 23rd international conference on* (2016),

78–90.

[5] Arisholm, E. et al. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*. 83, 1 (2010), 2–17.

[6] Atifi, M. et al. 2017. *A comparative study of software testing techniques*.

[7] Bacchelli, A. and Bird, C. 2013. Expectations, outcomes, and challenges of modern code review. *Proceedings of the 2013 international conference on software engineering* (2013), 712–721.

[8] Baltes, S. et al. 2018. (No) influence of continuous integration on the commit activity in github projects. *arXiv preprint arXiv:1802.08441*. (2018).

[9] Baum, T. et al. 2017. The choice of code review process: A survey on the state of the practice. *International conference on product-focused software process improvement* (2017), 111–127.

[10] Baum, T. et al. 2016. A faceted classification scheme for change-based industrial code review processes. *Software quality, reliability and security (qrs), 2016 ieee international conference on* (2016), 74–85.

[11] Bavota, G. et al. 2014. How the apache community upgrades dependencies: An evolutionary study. *Empirical Software Engineering*. 20, 5 (Sep. 2014), 1275–1317.

[12] Baysal, O. et al. 2016. Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering*. 21, 3 (2016), 932–959.

[13] Baysal, O. et al. 2013. The influence of non-technical factors on code review. *Reverse engineering (wcre), 2013 20th working conference on* (2013), 122–131.

[14] Beck, K. 2003. *Test-driven development: By example*. Addison-Wesley Professional.

[15] Beller, M. et al. 2014. Modern code reviews in open-source projects: Which problems do they fix? *Proceedings of the 11th working conference on mining software repositories* (2014), 202–211.

[16] Beller, M. et al. 2017. Developer testing in the ide: Patterns, beliefs, and behavior. *IEEE Transactions on Software Engineering*. 1 (2017), 1–1.

[17] Beller, M. et al. 2015. How (much) do developers test? *Proceedings of the 37th international conference on software engineering - volume 2* (Piscataway, NJ, USA, 2015), 559–562.

[18] Beller, M. et al. 2017. Oops, my tests broke the build: An explorative analysis of travis ci with github. *Mining software repositories (msr), 2017 ieee/acm 14th international conference on* (2017), 356–367.

[19] Beller, M. et al. 2017. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. *Proceedings of the 14th international conference on mining software repositories* (2017), 447–450.

[20] Beller, M. et al. 2015. When, how, and why developers (do not) test in their ides. *2015 10th joint meeting of the european software engineering conference and the acm sigsoft symposium on the foundations of software engineering, esec/fse 2015 - proceedings* (2015), 179–190.

[21] Bevan, J. et al. 2005. Facilitating software evolution research with kenyon. *ESEC/fse'05 - proceedings of the joint 10th european software engineering conference (esec) and 13th acm sigsoft symposium on the*

foundations of software engineering (fse-13) (2005), 177–186.

[22] Bird, C. and Zimmermann, T. 2017. Predicting software build errors. Google Patents.

[23] Bird, C. et al. 2015. Lessons learned from building and deploying a code review analytics platform. *Proceedings of the 12th working conference on mining software repositories* (2015), 191–201.

[24] Bisong, E. et al. 2017. Built to last or built too fast?: Evaluating prediction models for build times. *Proceedings of the 14th international conference on mining software repositories* (2017), 487–490.

[25] Blincoe, K. et al. 2015. Ecosystems in GitHub and a method for ecosystem identification using reference coupling. *2015 IEEE/ACM 12th working conference on mining software repositories* (May 2015).

[26] Bogart, C. et al. 2016. How to break an API: Cost negotiation and community values in three software ecosystems. *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering - FSE 2016* (2016).

[27] Bosu, A. and Carver, J.C. 2013. Impact of peer code review on peer impression formation: A survey. *Empirical software engineering and measurement, 2013 acm/ieee international symposium on* (2013), 133–142.

[28] Bouwers, E. et al. 2012. Getting what you measure. *Commun. ACM.* 55, 7 (Jul. 2012), 54–59.

[29] Bowring, J. and Hegler, H. 2014. Obsidian: Pattern-based unit test implementations. *Journal of Software Engineering and Applications.* 7, 02 (2014), 94.

[30] Castelluccio, M. et al. 2017. Is it safe to uplift this patch? An empirical study on mozilla firefox. *Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017* (2017), 411–421.

[31] Catal, C. 2011. Software fault prediction: A literature review and current trends. *Expert Systems with Applications.* 38, 4 (2011), 4626–4636.

[32] Catal, C. and Diri, B. 2009. A systematic review of software fault prediction studies.

[33] Catal, C. and Diri, B. 2009. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences.* 179, 8 (2009), 1040–1058.

[34] Cesar Brandão Gomes da Silva, A. et al. 2017. Frequent releases in open source software: A systematic review. *Information.* 8, 3 (2017), 109.

[35] Ciolkowski, M. et al. 2003. Software reviews: The state of the practice. *IEEE software.* 6 (2003), 46–51.

[36] Claes, M. et al. 2017. Abnormal working hours: Effect of rapid releases and implications to work content. *IEEE International Working Conference on Mining Software Repositories* (2017), 243–247.

[37] Claes, M. et al. 2015. A historical analysis of debian package incompatibilities. *2015 IEEE/ACM 12th working conference on mining software repositories* (May 2015).

[38] Constantinou, E. and Mens, T. 2017. An empirical comparison of developer retention in the RubyGems and npm software ecosystems. *Innovations in Systems and Software Engineering.* 13, 2-3 (Aug. 2017), 101–115.

[39] Costa, D.A. da et al. 2014. An empirical study of delays in the integration of addressed issues. *2014 ieee international conference on software maintenance and evolution* (2014), 281–290.

[40] Costa, D.A. da et al. 2016. The impact of switching to a rapid release cycle on the integration delay of addressed issues - an empirical study of the mozilla firefox project. *2016 ieee/acm 13th working conference*

on mining software repositories (msr) (2016), 374–385.

[41] Cox, J. et al. 2015. Measuring dependency freshness in software systems. *2015 IEEE/ACM 37th IEEE international conference on software engineering* (May 2015).

[42] Czerwonka, J. et al. 2015. Code reviews do not find bugs: How the current code review best practice slows us down. *Proceedings of the 37th international conference on software engineering-volume 2* (2015), 27–28.

[43] Decan, A. et al. 2017. An empirical comparison of dependency issues in OSS packaging ecosystems. *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)* (Feb. 2017).

[44] Decan, A. et al. 2018. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*. (Feb. 2018).

[45] Dietrich, J. et al. 2014. Broken promises: An empirical study into evolution problems in java programs caused by library upgrades. *2014 software evolution week - IEEE conference on software maintenance, reengineering, and reverse engineering (CSMR-WCRE)* (Feb. 2014).

[46] Dittrich, Y. 2014. Software engineering beyond the project sustaining software ecosystems. *Information and Software Technology*. 56, 11 (Nov. 2014), 1436–1456.

[47] Dulz, W. 2013. Model-based strategies for reducing the complexity of statistically generated test suites. *International conference on software quality* (2013), 89–103.

[48] Dyck, A. et al. 2015. Towards definitions for release engineering and devops. *Release engineering (releng), 2015 ieee/acm 3rd international workshop on* (2015), 3–3.

[49] D'Ambros, M. et al. 2010. An extensive comparison of bug prediction approaches. *Proceedings - International Conference on Software Engineering*. (2010), 31–41.

[50] D'Ambros, M. et al. 2012. Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Empirical Software Engineering*. 17, 4-5 (2012), 531–577.

[51] Eick, S.G. et al. 2001. Does code decay? Assessing the evidence from change management data. *IEEE Transactions on Software Engineering*. 27, 1 (Jan. 2001), 1–12.

[52] Fagan, M. 2002. Design and code inspections to reduce errors in program development. *Software pioneers*. Springer. 575–607.

[53] Fowler, M. and Foemmel, M. 2006. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*. 122, (2006), 14.

[54] Fujibayashi, D. et al. 2017. Does the release cycle of a library project influence when it is adopted by a client project? *SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and*

Reengineering (2017), 569–570.

[55] Garousi, V. and Zhi, J. 2013. A survey of software testing practices in canada. *Journal of Systems and Software*. 86, 5 (2013), 1354–1376.

[56] Gousios, G. et al. 2014. An exploratory study of the pull-based software development model. *Proceedings of the 36th international conference on software engineering* (2014), 345–355.

[57] Greiler, M. et al. 2013. Strategies for avoiding text fixture smells during software evolution. *IEEE international working conference on mining software repositories* (2013), 387–396.

[58] Gyimothy, T. et al. 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*. 31, 10 (Oct. 2005), 897–910.

[59] Hall, T. et al. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering*. 38, 6 (Nov. 2012), 1276–1304.

[60] Hassan, F. and Wang, X. 2018. HireBuild: An automatic approach to history-driven repair of build scripts. *Proceedings of the 40th international conference on software engineering* (2018), 1078–1089.

[61] Hejderup, J. et al. 2018. Software ecosystem call graph for dependency management. *Proceedings of the 40th international conference on software engineering new ideas and emerging results - ICSE-NIER 18* (2018).

[62] Hemmati, H. and Sharifi, F. 2018. Investigating nlp-based approaches for predicting manual test case failure. *Proceedings - 2018 ieee 11th international conference on software testing, verification and validation, icst 2018* (2018), 309–319.

[63] Hilton, M. et al. 2016. Usage, costs, and benefits of continuous integration in open-source projects. *Proceedings of the 31st ieee/acm international conference on automated software engineering* (2016), 426–437.

[64] Hora, A. et al. 2016. How do developers react to API evolution? A large-scale empirical study. *Software Quality Journal*. 26, 1 (Oct. 2016), 161–191.

[65] Hurdugaci, V. and Zaidman, A. 2012. Aiding software developers to maintain developer tests. *2012 16th european conference on software maintenance and reengineering* (March 2012), 11–20.

[66] Izquierdo, D. et al. 2018. Software development analytics for xen: Why and how. *IEEE Software*. (2018), 1–1.

[67] Jansen, S. 2014. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*. 56, 11 (Nov. 2014), 1508–1519.

[68] Karvonen, T. et al. 2017. Systematic literature review on the impacts of agile release engineering practices. *Information and Software Technology*. 86, (2017), 87–100.

[69] Kaur, A. and Vig, V. 2019. On understanding the release patterns of open source java projects. *Advances in Intelligent Systems and Computing*. 711, (2019), 9–18.

[70] Kerzazi, N. and Robillard, P. 2013. Kanbanize the release engineering process. *2013 1st International Workshop on Release Engineering, RELENG 2013 - Proceedings* (2013), 9–12.

[71] Khomh, F. et al. 2015. Understanding the impact of rapid releases on software quality. *Empirical Software Engineering*. 20, 2 (2015), 336–373.

[72] Khomh, F. et al. 2012. Do faster releases improve software quality?: An empirical case study of mozilla firefox. *Proceedings of the 9th ieee working conference on mining software repositories* (Piscataway, NJ, USA,

2012), 179–188.

[73] Kikas, R. et al. 2017. Structure and evolution of package dependency networks. *2017 IEEE/ACM 14th international conference on mining software repositories (MSR)* (May 2017).

[74] Kitchenham 2007. *Guidelines for performing systematic literature reviews in software engineering*. Keele University; University of Durham.

[75] Kitchenham, B. 2004. Procedures for performing systematic reviews. *Keele*. 33, 1 (2004), 1–26.

[76] Kitchenham, B. 2004. Procedures for performing systematic reviews. *Keele, UK, Keele University*. 33, 2004 (2004), 1–26.

[77] Kula, R.G. et al. 2017. An exploratory study on library aging by monitoring client usage in a software ecosystem. *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)* (Feb. 2017).

[78] Kula, R.G. et al. 2017. Do developers update their library dependencies? *Empirical Software Engineering*. 23, 1 (May 2017), 384–417.

[79] Laukkanen, E. et al. 2018. Comparison of release engineering practices in a large mature company and a startup. *Empirical Software Engineering*. (2018), 1–43.

[80] Leung, H.K. and Lui, K.M. 2015. Testing analytics on software variability. *Software analytics (swan), 2015 ieee 1st international workshop on* (2015), 17–20.

[81] Lewis, C. et al. 2013. Does bug prediction support human developers? Findings from a Google case study. *2013 35th international conference on software engineering (icse)* (May 2013), 372–381.

[82] Lungu, M. 2009. *Reverse engineering software ecosystems*. University of Lugano.

[83] Malloy, B.A. and Power, J.F. 2018. An empirical analysis of the transition from python 2 to python 3. *Empirical Software Engineering*. (Jul. 2018).

[84] Malloy, B.A. and Power, J.F. 2017. Quantifying the transition from python 2 to 3: An empirical study of python applications. *2017 ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)* (Nov. 2017).

[85] Manikas, K. 2016. Revisiting software ecosystems research: A longitudinal literature study. *Journal of Systems and Software*. 117, (Jul. 2016), 84–103.

[86] Marsavina, C. et al. 2014. Studying fine-grained co-evolution patterns of production and test code. *2014 ieee 14th international working conference on source code analysis and manipulation* (Sept 2014), 195–204.

[87] Mäntylä, M.V. et al. 2015. On rapid releases and software testing: A case study and a semi-systematic literature review. *Empirical Software Engineering*. 20, 5 (2015), 1384–1425.

[88] McDonnell, T. et al. 2013. An empirical study of API stability and adoption in the android ecosystem. *2013 IEEE international conference on software maintenance* (Sep. 2013).

[89] McIntosh, S. et al. 2014. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. *Proceedings of the 11th working conference on mining*

software repositories (2014), 192–201.

[90] Mens, T. et al. 2013. Studying evolving software ecosystems based on ecological models. *Evolving software systems*. Springer Berlin Heidelberg. 297–326.

[91] Messerschmitt, D.G. and Szyperski, C. 2003. *Software ecosystem: Understanding an indispensable technology and industry (mit press)*. The MIT Press.

[92] Mirzaaghaei, M. et al. 2012. Supporting test suite evolution through test case adaptation. *2012 ieee fifth international conference on software testing, verification and validation* (April 2012), 231–240.

[93] Moiz, S.A. 2017. Uncertainty in software testing. *Trends in software testing*. Springer. 67–87.

[94] Ni, A. and Li, M. 2018. ACONA: Active online model adaptation for predicting continuous integration build failures. *Proceedings of the 40th international conference on software engineering: Companion proceeedings* (2018), 366–367.

[95] Noor, T.B. and Hemmati, H. 2015. Test case analytics: Mining test case traces to improve risk-driven testing. *Software analytics (swan), 2015 ieee 1st international workshop on* (2015), 13–16.

[96] Pinto, G. and Rebouças, F.C.R.B.M. 2018. Work practices and challenges in continuous integration: A survey with travis ci users. (2018).

[97] Pinto, L.S. et al. 2013. TestEvol: A tool for analyzing test-suite evolution. *Proceedings - international conference on software engineering* (2013), 1303–1306.

[98] Pinto, L.S. et al. 2012. Understanding myths and realities of test-suite evolution. *Proceedings of the acm sigsoft 20th international symposium on the foundations of software engineering* (2012), 33.

[99] Plewnia, C. et al. 2014. On the influence of release engineering on software reputation. *Mountain view, ca, usa: In 2nd international workshop on release engineering* (2014).

[100] Poo-Caamaño, G. 2016. *Release management in free and open source software ecosystems*.

[101] Raemaekers, S. et al. 2017. Semantic versioning and impact of breaking changes in the maven repository. *Journal of Systems and Software*. 129, (Jul. 2017), 140–158.

[102] Rajlich, V. 2014. Software evolution and maintenance. *Proceedings of the on future of software engineering - FOSE 2014* (2014).

[103] Rausch, T. et al. 2017. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. *Proceedings of the 14th international conference on mining software repositories* (2017), 345–355.

[104] Robbes, R. et al. 2012. How do developers react to API deprecation? *Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering - FSE 12* (2012).

[105] Robinson, B. et al. 2011. Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs. *2011 26th ieee/acm international conference on automated software*

engineering (ase 2011) (Nov. 2011), 23–32.

[106] Rodríguez, P. et al. 2017. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*. 123, (2017), 263–291.

[107] Romano, S. et al. 2017. Findings from a multi-method study on test-driven development. *Information and Software Technology*. 89, (2017), 64–77.

[108] Santolucito, M. et al. 2018. Statically verifying continuous integration configurations. *arXiv preprint arXiv:1805.04473*. (2018).

[109] Schneidewind, N.F. 2007. Risk-driven software testing and reliability. *International Journal of Reliability, Quality and Safety Engineering*. 14, 2 (2007), 99–132.

[110] Shamshiri, S. et al. 2018. How do automatically generated unit tests influence software maintenance? *Software testing, verification and validation (icst), 2018 ieee 11th international conference on* (2018), 250–261.

[111] Souza, R. et al. 2015. Rapid releases and patch backouts: A software analytics approach. *IEEE Software*. 32, 2 (2015), 89–96.

[112] Stallman, R. 2002. *Free software, free society: Selected essays of richard m. stallman*. Lulu. com.

[113] Stolberg, S. 2009. Enabling agile testing through continuous integration. *Agile conference, 2009. agile'09.* (2009), 369–374.

[114] Teixeira, J. 2017. Release early, release often and release on time. an empirical case study of release management. *Open source systems: Towards robust practices* (Cham, 2017), 167–181.

[115] Teixeira, J. et al. 2015. Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem. *Journal of Internet Services and Applications*. 6, 1 (Jul. 2015).

[116] Thongtanunam, P. et al. 2016. Revisiting code ownership and its relationship with software quality in the scope of modern code review. *Proceedings of the 38th international conference on software engineering* (2016), 1039–1050.

[117] Thongtanunam, P. et al. 2015. Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. *Software analysis, evolution and reengineering (saner), 2015 ieee 22nd international conference on* (2015), 141–150.

[118] Trockman, A. 2018. Adding sparkle to social coding. *Proceedings of the 40th international conference on software engineering companion proceeedings - ICSE 18* (2018).

[119] Vassallo, C. et al. 2018. Un-break my build: Assisting developers with build repair hints. (2018).

[120] Vassallo, C. et al. 2017. A tale of ci build failures: An open source and a financial organization perspective. *Software maintenance and evolution (icsme), 2017 ieee international conference on* (2017), 183–193.

[121] Vernotte, A. et al. 2015. *Risk-driven vulnerability testing: Results from eHealth experiments using patterns and model-based approach*.

[122] Widder, D.G. et al. 2018. I'im leaving you, travis: A continuous integration breakup story. (2018).

[123] Xia, X. et al. 2015. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. *Software maintenance and evolution (icsme), 2015 ieee international conference on* (2015), 261–270.

[124] Zaidman, A. et al. 2011. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering*. 16, 3 (2011), 325–364.

[125] Zampetti, F. et al. 2017. How open source projects use static code analysis tools in continuous integration pipelines. *Mining software repositories (msr), 2017 ieee/acm 14th international conference on*

(2017), 334–344.

[126] Zanjani, M.B. et al. 2016. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering*. 42, 6 (2016), 530–543.

[127] Zhao, Y. et al. 2017. The impact of continuous integration on other software development practices: A large-scale empirical study. *Proceedings of the 32nd ieee/acm international conference on automated software engineering* (2017), 60–71.