A Literature Survey of Software Analytics

IN4334 2018 TU Delft

2018-10-02

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 faultsPinto et al. (2012). However, in practice it seems that not all developers test actively. In this paper 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. The above example could have been prevented by making tests but is not guaranteed to do so. 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 paper 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 paper, 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 answser 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 paper 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, Google Scholar and Springer.

The keywords used 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.

Category	Reference	Query
Test evolution	Mirzaaghaei et al. (2012)	Google Scholar query: test-suite evolution
Test evolution	Pinto et al. (2013)	Referenced by: Understanding myths and realities of test-suite evolution
Test evolution	Bevan et al. (2005)	Referenced by: Understanding myths and realities of test-suite evolution
Test evolution	Pinto et al. (2012)	Initial Paper Seed
Co-evolution	Marsavina et al. (2014)	Google Scholar keywords: Maintain developer tests, in 'cited by' of "Aiding Software Developers to Maintain Developer Tests" on IEEE
Co-evolution	Zaidman et al. (2011)	Initial Paper Seed
Co-evolution	Greiler et al. (2013)	In 'cited by' of "Understanding myths and realities of test-suite evolution" on Scopus
Co-evolution	Hurdugaci and Zaidman (2012)	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
Production evolution	Eick et al. (2001)	Referenced by: Testing analytics on software variability
Production evolution	@leung2015testing	Initial Paper Seed
Test generation	Robinson et al. (2011)	Referenced in Supporting Test Suite Evolution through Test Case Adaptation

Category	Reference	Query
Test generation	Bowring and Hegler (2014)	Springer: Reverse search on "Automatically generating maintainable regression unit tests for programs"
Test generation	Shamshiri et al. (2018)	Google Scholar query: Automatically generating unit tests
Test generation	@dulz2013model	Scopus query: "software development" AND Computer Science AND Software Engineering
Testing practices	Garousi and Zhi (2013)	Google Scholar query: Survey software testing
Testing practices	Beller et al. (2017a)	Initial Paper Seed
Testing practices	Beller et al. (2015)	In 'cited by' of "Understanding myths and realities of test-suite evolution".
Testing practices	Moiz (2017)	Springer query: software testing
Risk-driven testing	Hemmati and Sharifi (2018)	In 'cited by' of "Test case analytics: Mining test case traces to improve risk-driven testing"
Risk-driven testing	Schneidewind (2007)	Scopus query: risk-driven testing
Risk-driven testing	Vernotte et al. (2015)	Scopus query: "risk-driven" AND testing
Risk-driven testing	Atifi et al. (2017)	In 'cited by' of "Risk-driven software testing and reliability"
Risk-driven testing	Noor and Hemmati (2015)	Initial Paper Seed

3.2.1 Papers per research question

In this section, each of the papers is categorized with a corresponding research question. In the table above, the categories per paper were added based on their general topic. These broad topics will be assigned to a corresponding research question. All papers per research question are ordered on their relevance, which in most cases means that a newer paper is considered as more relevant than an older paper. A lower ranking may also be caused by a lower quality of writing (e.g. Greiler et al. (2013) in RQ2). The categorizations are based on the bullet points extracted from each paper. These bullet points can be found below in section '*Extracted paper information*' below.

- **RQ1** (*How do developers currently test?*):
  - Beller et al. (2017a)
  - Beller et al. (2015)
  - Marsavina et al. (2014)
  - Pinto et al. (2013)
  - Garousi and Zhi (2013)
  - Pinto et al. (2012)
  - Zaidman et al. (2011)
- **RQ2** (*What state of the art technologies are being used?*):
  - Mirzaaghaei et al. (2012)
  - Vernotte et al. (2015)

  - Bowring and Hegler (2014)
  - Hurdugaci and Zaidman (2012)
  - Robinson et al. (2011)
  - Greiler et al. (2013)
  - Dulz (2013)
  - Atifi et al. (2017)
  - Noor and Hemmati (2015)
- **RQ3** (*What future developments can be expect?*):
  - Hemmati and Sharifi (2018)
  - Shamshiri et al. (2018)
  - Vernotte et al. (2015)
  - Noor and Hemmati (2015)
  - Mirzaaghaei et al. (2012)
  - Bowring and Hegler (2014)
  - Leung and Lui (2015)
  - Greiler et al. (2013)
  - Atifi et al. (2017)

3.3 Extracted paper information

The papers retrieved using the research protocol are reviewed for their quality and useful information is extracted to be able to answer the research questions. This information can be found in this section as a list of bullet-points. If a paper is perceived as 'bad' or irrelevant for answering the research questions, this is elaborated.

3.3.1 Test evolution

Supporting Test Suite Evolution through Test Case Adaptation (Mirzaaghaei et al. (2012))

- Test case evolution.
- Automatic test repairing using information available in existing test cases.
- Identifies a set of common actions for adapting test cases by developers.
- Properly repairs 90% of the compilation errors addressed and covers the same amount of instructions.
- Not all prototypes were tested.
- Claims that many test cases designed for the early versions of the system become obsolete during the software lifecycle.
- An approach is proposed for automatically repairing and generating test cases during software evolution.
- 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.
- The results show that the approach can effectively maintain evolving test suites, and perform well compared to competing approaches.
- Frequent actions for adapting test cases that developers commonly adopt to repair and generate test cases are identified.
- In general: automated test case evolution seems fairly possible. (in 2012)

TestEvol: A tool for analyzing test-suite evolution (Pinto et al. (2013))

- Test case evolution.
- Tool for systematic investigating the evolution of the test-suite.
- Motivation: understand test maintenance in general.
- Only for Java and JUnit.

Facilitating software evolution research with kenyon (Bevan et al. (2005)) This paper is too old based on our exclusion criteria.

Understanding myths and realities of test-suite evolution (Pinto et al. (2012))

- Systematic measurement of how test-suites evolve
- Test repairs occur in practice. avg 16 repairs per version –> often enough to warrant the development of automated techniques.
- Test repairs are not the primary reason for test modification. Non-test repair related modifications occur about 4 times as frequently.
- Only 10% of the tests consider fixed assert tests (oracle tests)
- Test repairs frequently consider repairs to method call chains.
- Test deletions and additions are often due to refactoring
- A considerable portion of the additions is due to augmenting tests to make it more *adequate.
- General: automated techniques may be useful.

3.3.2 Co-Evolution

Studying Fine-Grained Co-evolution Patterns of Production and Test Code (Marsavina et al. (2014))

- Co-evolution of production and test code.
- Generally co-evolving test and production code is a difficult tass.
- Mines fine-grained changes from the evolution of 5 open-source systems.
- Also uses an association rule mining algorithm to generate co-evolution patterns.
- The patterns are interpreted by performing a qualitative analysis.
- Meant to gain a deeper understanding of the way in which tests evolve as a result of changes in the production classes and identify possible gaps to signal developers for missed production code parts that have not been addressed adequately by tests.
- Some patterns that were found:
- Tests are mostly removed when production classes they cover are deleted. Programmers are careful not to leave non-compiling tests.
- Only limited effort is done on updating test cases after production classes are modified; tests are rarely changed when attributes or methods are changed in the production classes.
- A pattern indicates that mostly when numerous condition related changes are made in the production methods, test cases are created/deleted in order to address the branches that were removed/added.
- Test cases are rarely updated when changes related to attributes or methods are made in the production code.
- Test methods are in several cases created/deleted when conditional statements are altered in the production code.
- Future work should include the co-evolution patterns of different coding methodologies, for example, Test-Driven Development and their possible respective differences.
- Future work should include intent-preserving techniques, which could help ensure test repairs address the same production code functionalities as before the tests were broken.

Studying the co-evolution of production and test code (Zaidman et al. (2011))

- Testing is phased and co-evolution is synchronous
- No increase in testing activity before major releases. Intense phases were detected.
- Evidence for TDD discovered in 2/6 test cases.
- The fraction of test code (wrt prod code) increases as coverage increases

Strategies for avoiding text fixture smells during software evolution (Greiler et al. (2013))

- Knowledge about how and when smells in test fixtures are produced.
- Test fixture smells do not continuously develop over time.
- A correlation between the number of tests and smells.
- Few test cases contribute to the majority of the smells.

- Not the highest quality paper, the title even contains a typo, where 'text' should be 'test'.

Aiding Software Developers to Maintain Developer Tests (Hurdugaci and Zaidman (2012))

- Support for co-evolution of testing code with production code.
- Introduces TestNForce (Visual Studio only), a tool to help developers to identify unit tests that need to be altered and executed after code change.
- Gives a broad explanation for the need for the co-evolution of test code.
- Three scenarios: show covering tests, enforcing self-contained commits and what tests need to run?
- Used an experimental setup with only eight participants from the Delft University of Technology of which two participants did not use Unit testing. Hard to generalize.
- On average, the participants considered 80 code coverage as "good".

3.3.3 Production evolution

Does code decay? Assessing the evidence from change management data (Eick et al. (2001)) This paper is too old based on our exclusion criteria.

Testing analytics on software variability (Leung and Lui (2015))

- Variability-aware testing.
- System integration testing has to be manually executed to evaluate the system's compliance with its specified requirement and performance.
- Aids testers and developers to reduce their product time-to-market by utilizing historical testing results and similarity among systems.

3.3.4 Test generation

Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs (Robinson et al. (2011))

- A system that has good coverage and mutation kill score, made readable code and required few edits as the system under test evolved. (stable)
- Statement: The costs of unit tests are not perceived to outweigh the benefits.
- Previous techniques: hard to understand / maintain / brittle and only tested on libraries → not real software development code.
- They claim they made a pretty well working test-generation tool.

Obsidian: Pattern-Based Unit Test Implementations (Bowring and Hegler (2014))

- A tool that generates the templates for tests: guarantee compilation, support exception handling, find suitable location… etc.
- Developers still need to fix the oracle tests, but the implementation/template is there.
- Looks at the context in order to decide what template to use.
- Distinguishes implementations from test cases

How Do Automatically Generated Unit Tests Influence Software Maintenance? (Shamshiri et al. (2018))

- Automatically generated tests are usually not based on realistic scenarios, and are therefore generally considered to be less readable.
- Every time a test fails, a developer has to decide whether this failure has revealed a regression fault in the program under test, or whether the test itself needs to be updated.
- Whilst maintenance activities take longer when working with automatically generated tests, they found developers to be equally effective with manually written and automatically generated tests.
- There is a need for research into the generation of more realistic tests.

Model-Based Strategies for Reducing the Complexity of Statistically Generated Test Suites (Dulz (2013))

- By directed 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, like 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.

3.3.5 Testing Practices

A survey of software testing practices in Canada (Garousi and Zhi (2013))

- The importance of testing-related training is increasing
- Functional and unit-testing receive the most effort and attention
- The mutation testing approach is getting attention amongst Canadian firms
- Test last approach is still dominant, few companies try TDD
- In terms of popularity: NUnit and Web application testing overtook JUnit and IBM Rational tools
- Coverage metrics, to most commonly used: branch and conditional coverage
- Number of passing test / defects per day is used as the most popular metric in order to determine a release
- Ratio of testers : developers is somewhere around 1:2 and 1:5. The total effort is estimated to be less than 40%
- More than 70% of the respondents participated in a forum related to testing on a regular basis
- In general: more attention to testing (in 2012)

Developer testing in the IDE: Patterns, beliefs and behavior (Beller et al. (2017a))

- Java C# developer testing behavior
- Little support for TDD
- Developers execute tests phased
- Only half of the developers practice testing actively
- Testing time is overestimated twofold.
- 12% of the test cases show flaky behavior
- Correlation between test flakiness and CI error-proneness?
- Few (25%) tests detect 75% of the execution failures.
- Tests and production code do not co-evolve gracefully.

When, how, and why developers (do not) test in their IDEs (Beller et al. (2015))

- Developers largely do not run tests in the IDE. However, when they do, they do it heftily.
- Tests run in the IDE take a short amount of time
- Developers run selective tests (often 1)
- Most test executions fail
- A typical reaction is to dive into offending code
- TDD is not widely practiced, even by those who say they do (strict definition)
- The way people test is different from how they believe they test.

Uncertainty in Software Testing (Moiz (2017))

- Mechanisms are needed to address uncertainty in each of the deliverables produced during software development process. The uncertainty metrics can help in assessing the degree of uncertainty.

3.3.6 Risk-driven testing

Investigating NLP-Based Approaches for Predicting Manual Test Case Failure (Hemmati and Sharifi (2018))

- System-level manual acceptance testing is one of the most expensive testing activities.

- A new test case failure prediction approach is proposed, which does not rely on source code or specification of the software under test.
- The approach uses basic Information Retrieval (IR) methods on the test case descriptions, written in natural language, based on the frequency of terms in the manual test scripts.
- The test fail prediction is accurate and the NLP-based feature can improve the prediction models.
- "To the best of our knowledge, this work is the first use of NLP on manual test case scripts for test failure prediction and has shown promising results, which we are planning to replicate on different systems and expand on different NLP-based features to more accurately extract features keywords from test cases."

Risk-driven software testing and reliability (Schneidewind (2007))

This paper is discarded from the survey, because it uses weak models to validate the claims made and is too old based on our exclusion criteria.

Risk-driven vulnerability testing: Results from eHealth experiments using patterns and model-based approach (Vernotte et al. (2015))

- This paper introduces and reports on an original tooled risk-driven security testing process called Pattern-driven and Model-based Vulnerability Testing. This fully automated testing process, drawing 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, conducted on a complex and freely-accessible eHealth system developed by Info World, 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 comparative study of software testing techniques (Atifi et al. (2017))

- They highlight two software testing techniques considered among the most used techniques to perform software tests, and then perform a comparative study of these techniques, the approaches that support studied techniques, and the tools used for each technique.
- The first technique is Model-based-testing, the second Risk-based testing.

Test case analytics: Mining test case traces to improve risk-driven testing (Noor and Hemmati (2015))

- In risk-driven testing, test cases are generated and/or prioritized based on different risk measures. *The most basic risk measure would analyze the history of the software and assigns higher risk to the test cases that used to detect bugs in the past.
- A new risk measure is defined 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.
- "Though our initial and simple implementation in this paper was very promising, we are planning to investigate other similarity functions, specifically those that account for the method orders in the trace. In addition, this project is a sub-project of a bigger research on risk-driven model-based testing, where we are planning to extract specification models of the system and augment them with the similarity-based risk measures. Those models can later be used in both risk-driven test generation and prioritization."

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.

Build analytics covers research done on data extracted from a build process inside a project. This contains amongst others, build logs from continuous integration such as Travis CI[1], Circle CI[2], Jenkins[3], AppVeyor[4] and TeamCity[5] or surveys among developers about their usage of Continuous Integration or build systems. This information is often paired with data from Version Control Systems (VCS) such as git.

In order to get a complete view of the current state of the build analytics field, we ask the following research questions.

RQ1: What is the current state of the art in the field of build analytics?

In order to answer to the first reseacrh question, we need to present the current topics that are being explored in the build analytics domain alogside with the research methods, tools and datasets acquired for the problems in hand and aggregate and reflect about the main research findings that the state-of-the-art papers display.

RQ2: What is the current state of practice in the field of build analytics?

This section examines scientific papers to analyse the current trend of build analytics in the software development industry. We looks at the popularity of CI in the industry as Hilton et al. (2016) describes. In addition, it also explores the increase in the use of Continuous Integration (CI) by discussing its ample benefits as Fowler and Foemmel (2006) present. Furthermore, it will discuss the practises used by engineers in the industry to ensure that their code is improving and not decaying.

RQ3: What future research can we expect in the field of build analytics?

In this section we will explore where new challenges lie in the field of build analytics. We will also show what open research items are described in the papers. This section ends with research questions based on the open research and challenges in current research.

[1]See https://travis-ci.org/
[2]See https://circleci.com/
[3]See https://jenkins.io/
[4]See https://www.appveyor.com/
[5]See https://www.jetbrains.com/teamcity/

4.2 Research protocol

Using the initial seed consisting of Bird and Zimmermann (2017), Beller et al. (2017b), Rausch et al. (2017), Beller et al. (2017c), Pinto and Rebouças (2018), Zhao et al. (2017), Widder et al. (2018) and Hilton et al. (2016) 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.

4.3 Answers

RQ1: 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.

The topics that are being explored are: * the importance of the build process in a VCS project Hassan and Wang (2018) * the impact factors of user satisfaction from using a CI tools Widder et al. (2018) * methods from helping the developer to fix bugs Hassan and Wang (2018), Vassallo et al. (2018) * predicting build time Bisong et al. (2017) * predicting build failures Santolucito et al. (2018), Ni and Li (2018)

The tools that are being proposed are: * BART to help developers fix build errors by generating a summary of the failures with useful information, thus eliminating the need to browse error logs Vassallo et al. (2018) * HireBuild to automatically fix build failures based on previous changes Hassan and Wang (2018) * 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 Santolucito et al. (2018) * ACONA capable of predicting build failure prediction in CI environment for newer projects with less data available Ni and Li (2018)

The build process is important part of a project that uses VCS in the way that based on the findings from Hassan and Wang (2018) 22% of code commits include changes in build script files for either build working or build fixing purposes. Moreover, recent studies has focused on how satisfied are the users of CI tools, one particular paper Widder et al. (2018) analyzing what factors have an impact on abandonment of Travis CI. The paper finds that increased build complexity reduces the chance of abandonment, but larger projects abandon at a higher rate and that a 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.

The topics that are being explored are: * the importance of the build process in a VCS project Hassan and Wang (2018) * the impact factors of user satisfaction from using a CI tools Widder et al. (2018) * methods from helping the developer to fix bugs Hassan and Wang (2018), Vassallo et al. (2018) * predicting build time Bisong et al. (2017) * predicting build failures Santolucito et al. (2018), Ni and Li (2018)

RQ2:

Continuous Integration is a development practise that requires developers to integrate code into a share repository several times a day. Each check-in is then verified by an automated build which allows engineers to detect any bugs early.

A survey conduced in open-source projects by Hilton et al. (2016) indicated that 40% of all projects used CI. It observed that a median project introduces CI a year into development. Furthermore, the paper claims that CI is widely used in practise nowadays. The paper by Rausch et al. (2017) explores how CI is widely available for projects of every size with the introduction of Version Control Systems (VCS) such as GitHub, and hosted build automation platforms such as Travis. In this way, CI adoption rates will increase further in the future.

This is an overview of CI and how it works in daily life. Maintaining this system requires engineers to follow fundamental practises presented by Fowler and Foemmel (2006). The practises presented by Fowler and Foemmel (2006) are still commonly used today, particularly in the agile software industry as suggested by Stolberg (2009).

- Maintain a Single Source Repository
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits To the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Fix Broken Builds Immediately
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it Easy for Anyone to Get the Latest Executable
- Everyone can see what's happening
- Automate Deployment

It is important to note that CI does not get rid of bugs, but it does make them dramatically easier to find and remove. The above practises are important for the smooth functioning of CI framework.

RQ3:

Currently research on build analytics is limited by some challenges, some are specific to build analytics and some are applicable to the entire field of software engineering.

In Bisong et al. (2017) the main limitation was the performance of the machine learning algorithm used. In the implementation R was used and it proved not capable of processing the amounts of data needed. This shows that it is important to choose the right tool when analyzing data.

Future research in build analytics branches in a couple of different topics. Pinto and Rebouças (2018) proposes to focus on getting a better understanding of the users and why they might choose to abandon an automatic build platform.

According to Baltes et al. (2018) 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.

- How do teams change their pull request review practices in response to the introduction of continuous integration? (Zhao et al. (2017))
- How can we detect if fixing a build configuration requires changes in the remote environment? (Vassallo et al. (2018))
- Does breaking the build often translate to worse project quality and decreased productivity? (Beller et al. (2017b))

From the synthesis of the works discussed in this section the following research questions emerged:

- What is the impact of the choice of Continuos Integration platform? Most of the research is done on users using Travis CI, there are many other platforms out there. Every platform has their own characteristics.

4.4 Summary of papers

Through this we found the following papers

4.4.1 Bird and Zimmermann (2017)

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.

4.4.2 Beller et al. (2017b)

Initial Seed

This paper explores data from Travis CI[6] 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.

4.4.3 Rausch et al. (2017)

Initial Seed

A stuy on the build results of 14 open source software Java projects. It is similar to Beller et al. (2017b), albeit on a smaller scale. It does go more in depth on the result and changes over time.

4.4.4 Beller et al. (2017c)

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.

4.4.5 Pinto and Rebouças (2018)

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.

4.4.6 Zhao et al. (2017)

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.

4.4.7 Widder et al. (2018)

Initial Seed

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

[6] 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.

4.4.8 Hilton et al. (2016)

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.

4.4.9 Vassallo et al. (2017)

References Beller et al. (2017b)

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.

4.4.10 Hassan and Wang (2018)

References Beller et al. (2017c)

This paper uses TravisTorrent (Beller et al. (2017c)) 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.

4.4.11 Vassallo et al. (2018)

References Beller et al. (2017b), Rausch et al. (2017)

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.

4.4.12 Zampetti et al. (2017)

Referenced by Vassallo et al. (2018)

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.

4.4.13 Baltes et al. (2018)

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.

4.5 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:

4.5.1 Bisong et al. (2017)

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.

4.5.2 Santolucito et al. (2018)

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.

4.5.3 Ni and Li (2018)

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.

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

IBM's Grady Booch first proposed the methodology of Continuous Integration in the 1990s. It became widely used when adopted by Extreme Programming in the 1996. In 2000, Fowler and Foemmel (2006) wrote the cornerstone description of integration practises. The following section provides critical analysis of the papers presented and explains the current practises of build analytics in our industry.

4.6.1 Fowler and Foemmel (2006)

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.

4.6.2 Hilton et al. (2016)

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.

4.6.3 Rausch et al. (2017)

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 to keep the build system healthy.

4.6.4 Stolberg (2009)

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.

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 D'Ambros et al. (2012), 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:

- Gyimothy et al. (2005)
- Catal and Diri (2009a)
- Arisholm et al. (2010)
- D'Ambros et al. (2010)
- Hall et al. (2012)
- Lewis et al. (2013)

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
D'Ambros et al. (2010)	is author of	D'Ambros et al. (2012)
Catal and Diri (2009a)	is author of	Catal (2011) Catal and Diri (2009b)

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 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.

Together with this comes the growth in open source projects. On platforms such as GitHub[3], it is easy and quick to create a new software product, which can be developed, reviewed and used by the whole community.

This in turn leads to intertwined landscapes of software products, which are deemed a *software ecosystems*. As stated by Messerschmitt and Szyperski (2003), a *software ecosystem* is "a collection of software products that have some given degree of symbiotic relationships." Another, similar definition is given by Lungu (2009): "A software ecosystem is a collection of software projects which are developed and co-evolve in the same environment." Mens et al. (2013) extends this definition, "by explicitly considering the communities involved (e.g. user and developer communities) as being part of the software ecosystem."

By performing analysis on these software ecosystems, people aim to generate meaningful insights, which can then be used to improve the efficiency and effectivity of the software development process throughout the lifecycle of the developed software.

In order to perform a survey on the current progress in the field of software ecosystem analytics, we have formulated three research questions:

- **RQ1**: What is the current state of the art in software analytics for ecosystem analytics?
- **RQ2**: What is the current state of practice in software analytics for ecosystem analytics?
- **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.

6.2 Research Protocol

In order to select literature to answer the research questions given in the previous section, the survey method suggested by Kitchenham (2004a) is used. This method creates a systematic way to select a set of papers, which is relevant to the research question(s).

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

The search strategy, as described by Kitchenham (2004a), 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

6.2.1 Initial seed

The first type is quite straightforward. To give this survey a head start, 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
-	Abate et al. (2009)	Strong dependencies between software components	2009	
-	Abate and Cosmo (2011)	Predicting upgrade failures using dependency analysis	2011	
+	Abdalkareem et al. (2017)	Why do developers use trivial packages? An empirical case study on NPM	2017	JavaScript; Node.js; Code Reuse; Empirical Studies
+	Bogart et al. (2016)	How to break an api: Cost negotiation and community values in three software ecosystem	2016	Software ecosystems; Dependency management; semantic versioning; Collaboration; Qualitative research
+	Claes et al. (2015)	A historical analysis of Debian package incompatibilities	2015	debian, conflict, empirical, analysis, software, evolution, distribution, package, dependency, maintenance
+	Constantinou and Mens (2017)	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
+	Hejderup et al. (2018)	Software Ecosystem Call Graph for Dependency Management	2018	
+	Kikas et al. (2017)	Structure and evolution of package dependency networks	2017	
+	Kula et al. (2017b)	Do developers update their library dependencies?	2017	Software reuse, Software maintenance, Security vulnerabilities
-	Mens et al. (2013)	Studying Evolving Software Ecosystems based on Ecological Models	2013	Coral Reef, Natural Ecosystem, Open Source Software, Ecological Model, Software Project

[4] https://github.com:

Selected	Author(s)	Title	Year	Keywords
+	Raemaekers et al. (2017)	Semantic versioning and impact of breaking changes in the Maven repository	2017	Semantic versioning, Breaking changes, Software libraries
+	Robbes et al. (2012)	How do developers react to API deprecation? The case of a smalltalk ecosystem	2012	Ecosystems, Mining Software Repositories, Empirical Studies
+	Trockman (2018)	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
Abate et al. (2009)	This paper seems to delve more into the software itself whereas we are more interested in the surrounding ecosystems

Paper Reference	Reason not selected
Abate and Cosmo (2011)	Kind of the same reason as Abate et al. (2009), again we are more interested in the surrounding ecosystems

Paper Reference	Reason not selected
Mens et al. (2013)	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[5] is used. 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)*

[5]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.

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
Decan et al. (2018)	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"
Dittrich (2014)	Software engineering beyond the project – Sustaining software ecosystems	2014		engineering software ecosystems
Hora et al. (2016)	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"
Izquierdo et al. (2018)	Software Development Analytics for Xen: Why and How	2018	Companies, Ecosystems, Software, Measurement, Object recognition, Monitoring, Virtualization	software ecosystem analytics
Jansen (2014)	Measuring the Health of Open Source Software Ecosystems: Beyond the Scope of Project Health	2014		"open source software ecosystems"
Kula et al. (2017a)	An exploratory study on library aging by monitoring client usage in a software ecosystem	2017		"software ecosystem" AND "analysis"
Malloy and Power (2018)	An empirical analysis of the transition from Python 2 to Python 3	2018	Python programming, Programming language evolution, Compliance	"software ecosystem" AND "empirical"
Manikas (2016)	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"
Rajlich (2014)	Software evolution and maintenance	2014		Software Evolution and Maintenance

First Author	Title	Year	Keywords	Query Used
Teixeira et al. (2015)	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
Bavota et al. (2014)	How the Apache community upgrades dependencies: an evolutionary study	2014	Software Ecosystems · Project dependency upgrades · Mining software repositories	Kula et al. (2017b)
Blincoe et al. (2015)	Ecosystems in GitHub and a method for ecosystem identification using reference coupling.	2015		Constantinou and Mens (2017)
Cox et al. (2015)	Measuring Dependency Freshness in Software Systems	2015		Kikas et al. (2017)
Decan et al. (2017)	An empirical comparison of dependency issues in OSS packaging ecosystems	2017		Abdalkareem et al. (2017), Constantinou and Mens (2017), Decan et al. (2018)
Dietrich et al. (2014)	Broken Promises - An Empirical Study into Evolution Problems in Java Programs Caused by Library Upgrades	2014		Raemaekers et al. (2017)
Malloy and Power (2017)	Quantifying the transition from Python 2 to 3: an empirical study of Python applications.	2017		Malloy and Power (2018)
McDonnell et al. (2013)	An empirical study of api stability and adoption in the android ecosystem	2013		Manikas (2016)

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

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 (Adams and McIntosh, 2016).

Therefore, this systematic literature review aims to provide an overview of the software analytics research that has been conducted so far on 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, with release engineering, practice seems to be ahead of 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 Adams and McIntosh (2016), da Costa et al. (2016), d. Costa et al. (2014), Khomh et al. (2012), and Khomh et al. (2015).

We collected publications using two search engines: Scopus and Google Scholar. These each encompass various 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:

- Kaur and Vig (2019)
- Kerzazi and Robillard (2013)
- Castelluccio et al. (2017)
- Karvonen et al. (2017)
- Claes et al. (2017)
- Fujibayashi et al. (2017)
- Souza et al. (2015)
- Laukkanen et al. (2018)

```
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. For each paper, the review concerned the publications cited by the paper, as well as those citing the paper. These reference lists were obtained from Google Scholar and from the reference lists in the papers themselves. 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
Souza et al. (2015)	has cited	Plewnia et al. (2014) Mäntylä et al. (2015)
Khomh et al. (2015)	is cited by	Poo-Caamaño (2016) Teixeira (2017)
Mäntylä et al. (2015)	is cited by	Rodríguez et al. (2017) Cesar Brandão Gomes da Silva et al. (2017)

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

In the utilized tool for conducting the survey, 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 Kitchenham (2004b), the quality of a paper will be assessed by the evidence it provides, based on the following scale. All levels of quality will be accepted, except for level 5 (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 Kitchenham (2004b):

- 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 (Kitchenham, 2004b).

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:
    Journal:
    Publication type:
    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

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

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. (Kitchenham, 2007).

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	Bacchelli and Bird (2013)	
Beller, M.	2014	Beller et al. (2014)	
Bird, C.	2015	Bird et al. (2015)	
Fagan, M.	2002	Fagan (2002)	
Gousios, G.	2014	Gousios et al. (2014)	
McIntosh, S.	2014	McIntosh et al. (2014)	

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	Baysal et al. (2016)	
Thongtanunam, P.	2015	Thongtanunam et al. (2015)	
Thongtanunam, P.	2016	Thongtanunam et al. (2016)	
Xia, X.	2015	Xia et al. (2015)	
Zanjani, M. B.	2016	Zanjani et al. (2016)	

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	Baum et al. (2016)		
Baum	2017	Baum et al. (2017)		
Baysal	2013	Baysal et al. (2013)		
Bosu	2013	Bosu and Carver (2013)		
Ciolkowski	2003	Ciolkowski et al. (2003)		
Czerwonka	2015	Czerwonka et al. (2015)		

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: Bacchelli and Bird (2013)

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: Baum et al. (2016)

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: Baum et al. (2017)

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: Baysal et al. (2013)

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

Reference: Baysal et al. (2016)

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

Reference: Beller et al. (2014)

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: Bird et al. (2015)

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: Bosu and Carver (2013)

8.3.9 Software Reviews: The State of the Practice

Reference: Ciolkowski et al. (2003)

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: Czerwonka et al. (2015)

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: Fagan (2002)

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: Gousios et al. (2014)

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: McIntosh et al. (2014)

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: Thongtanunam et al. (2015)

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

Reference: Thongtanunam et al. (2016)

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

Reference: Xia et al. (2015)

8.3.17 Automatically recommending peer reviewers in modern code review

Reference: Zanjani et al. (2016)

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 (Kitchenham, 2004b), 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.

Bibliography

Abate, P. and Cosmo, R. D. (2011). Predicting upgrade failures using dependency analysis. In *2011 IEEE 27th International Conference on Data Engineering Workshops*. IEEE.

Abate, P., Cosmo, R. D., Boender, J., and Zacchiroli, S. (2009). Strong dependencies between software components. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE.

Abdalkareem, R., Nourry, O., Wehaibi, S., Mujahid, S., and Shihab, E. (2017). Why do developers use trivial packages? an empirical case study on npm. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*. ACM Press.

Adams, B. and McIntosh, S. (2016). Modern release engineering in a nutshell–why researchers should care. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 5, pages 78–90. IEEE.

Arisholm, E., Briand, L. C., and Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17.

Atifi, M., Mamouni, A., and Marzak, A. (2017). *A comparative study of software testing techniques*, volume 10299 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

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

Baltes, S., Knack, J., Anastasiou, D., Tymann, R., and Diehl, S. (2018). (no) influence of continuous integration on the commit activity in github projects. *arXiv preprint arXiv:1802.08441*.

Baum, T., Leßmann, H., and Schneider, K. (2017). The choice of code review process: A survey on the state of the practice. In *International Conference on Product-Focused Software Process Improvement*, pages 111–127. Springer.

Baum, T., Liskin, O., Niklas, K., and Schneider, K. (2016). A faceted classification scheme for change-based industrial code review processes. In *Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on*, pages 74–85. IEEE.

Bavota, G., Canfora, G., Penta, M. D., Oliveto, R., and Panichella, S. (2014). How the apache community upgrades dependencies: an evolutionary study. *Empirical Software Engineering*, 20(5):1275–1317.

Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. W. (2013). The influence of non-technical factors on code review. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 122–131. IEEE.

Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. W. (2016). Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering*, 21(3):932–959.

Beller, M., Bacchelli, A., Zaidman, A., and Juergens, E. (2014). Modern code reviews in open-source projects: Which problems do they fix? In *Proceedings of the 11th working conference on mining software repositories*, pages 202–211. ACM.

Beller, M., Georgios, G., Panichella, A., Proksch, S., Amann, S., and Zaidman, A. (2017a). Developer testing in the ide: Patterns, beliefs, and behavior. *IEEE Transactions on Software Engineering*, (1):1–1.

Beller, M., Gousios, G., Panichella, A., and Zaidman, A. (2015). When, how, and why developers (do not) test in their ides. In *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*, pages 179–190. Cited By :39.

Beller, M., Gousios, G., and Zaidman, A. (2017b). Oops, my tests broke the build: An explorative analysis of travis ci with github. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*, pages 356–367. IEEE.

Beller, M., Gousios, G., and Zaidman, A. (2017c). Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 447–450. IEEE press.

Bevan, J., Whitehead Jr., E. J., Kim, S., and Godfrey, M. (2005). Facilitating software evolution research with kenyon. In *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)*, pages 177–186. Cited By :86.

Bird, C., Carnahan, T., and Greiler, M. (2015). Lessons learned from building and deploying a code review analytics platform. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 191–201. IEEE Press.

Bird, C. and Zimmermann, T. (2017). Predicting software build errors. US Patent 9,542,176.

Bisong, E., Tran, E., and Baysal, O. (2017). Built to last or built too fast?: evaluating prediction models for build times. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 487–490. IEEE Press.

Blincoe, K., Harrison, F., and Damian, D. (2015). Ecosystems in GitHub and a method for ecosystem identification using reference coupling. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE.

Bogart, C., Kästner, C., Herbsleb, J., and Thung, F. (2016). How to break an API: cost negotiation and community values in three software ecosystems. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*. ACM Press.

Bosu, A. and Carver, J. C. (2013). Impact of peer code review on peer impression formation: A survey. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pages 133–142. IEEE.

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

Castelluccio, M., An, L., and Khomh, F. (2017). Is it safe to uplift this patch? an empirical study on mozilla firefox. pages 411–421. cited By 0.

Catal, C. (2011). Software fault prediction: A literature review and current trends. *Expert Systems with Applications*, 38(4):4626–4636. Cited By :138.

Catal, C. and Diri, B. (2009a). A systematic review of software fault prediction studies.

Catal, C. and Diri, B. (2009b). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, 179(8):1040–1058. Cited By :109.

Cesar Brandão Gomes da Silva, A., de Figueiredo Carneiro, G., Brito e Abreu, F., and Pessoa Monteiro, M. (2017). Frequent releases in open source software: A systematic review. *Information*, 8(3):109.

Ciolkowski, M., Laitenberger, O., and Biffl, S. (2003). Software reviews: The state of the practice. *IEEE software*, (6):46–51.

Claes, M., Mantyla, M., Kuutila, M., and Adams, B. (2017). Abnormal working hours: Effect of rapid releases and implications to work content. pages 243–247. cited By 3.

Claes, M., Mens, T., Cosmo, R. D., and Vouillon, J. (2015). A historical analysis of debian package incompatibilities. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE.

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):101–115.

Cox, J., Bouwers, E., van Eekelen, M., and Visser, J. (2015). Measuring dependency freshness in software systems. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE.

Czerwonka, J., Greiler, M., and Tilford, J. (2015). Code reviews do not find bugs: how the current code review best practice slows us down. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 27–28. IEEE Press.

d. Costa, D. A., Abebe, S. L., Mcintosh, S., Kulesza, U., and Hassan, A. E. (2014). An empirical study of delays in the integration of addressed issues. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 281–290.

da Costa, D. A., McIntosh, S., Kulesza, U., and Hassan, A. E. (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. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 374–385.

D'Ambros, M., Lanza, M., and Robbes, R. (2010). An extensive comparison of bug prediction approaches. *Proceedings - International Conference on Software Engineering*, pages 31–41.

D'Ambros, M., Lanza, M., and Robbes, R. (2012). Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577. Cited By :167.

Decan, A., Mens, T., and Claes, M. (2017). An empirical comparison of dependency issues in OSS packaging ecosystems. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE.

Decan, A., Mens, T., and Grosjean, P. (2018). An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*.

Dietrich, J., Jezek, K., and Brada, P. (2014). Broken promises: An empirical study into evolution problems in java programs caused by library upgrades. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE.

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

Dulz, W. (2013). Model-based strategies for reducing the complexity of statistically generated test suites. In *International Conference on Software Quality*, pages 89–103. Springer.

Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., and Mockus, A. (2001). Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12.

Fagan, M. (2002). Design and code inspections to reduce errors in program development. In *Software pioneers*, pages 575–607. Springer.

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

Fujibayashi, D., Ihara, A., Suwa, H., Kula, R., and Matsumoto, K. (2017). Does the release cycle of a library project influence when it is adopted by a client project? pages 569–570. cited By 0.

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

Gousios, G., Pinzger, M., and Deursen, A. v. (2014). An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. ACM.

Greiler, M., Zaidman, A., Van Deursen, A., and Storey, M. . (2013). Strategies for avoiding text fixture smells during software evolution. In *IEEE International Working Conference on Mining Software Repositories*, pages 387–396. Cited By :12.

Gyimothy, T., Ferenc, R., and Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910.

Hall, T., Beecham, S., Bowes, D., Gray, D., and Counsell, S. (2012). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304.

Hassan, F. and Wang, X. (2018). Hirebuild: an automatic approach to history-driven repair of build scripts. In *Proceedings of the 40th International Conference on Software Engineering*, pages 1078–1089. ACM.

Hejderup, J., van Deursen, A., and Gousios, G. (2018). Software ecosystem call graph for dependency management. In *Proceedings of the 40th International Conference on Software Engineering New Ideas and Emerging Results - ICSE-NIER '18*. ACM Press.

Hemmati, H. and Sharifi, F. (2018). Investigating nlp-based approaches for predicting manual test case failure. In *Proceedings - 2018 IEEE 11th International Conference on Software Testing, Verification and Validation, ICST 2018*, pages 309–319.

Hilton, M., Tunnell, T., Huang, K., Marinov, D., and Dig, D. (2016). Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 426–437. ACM.

Hora, A., Robbes, R., Valente, M. T., Anquetil, N., Etien, A., and Ducasse, S. (2016). How do developers react to API evolution? a large-scale empirical study. *Software Quality Journal*, 26(1):161–191.

Hurdugaci, V. and Zaidman, A. (2012). Aiding software developers to maintain developer tests. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 11–20.

Izquierdo, D., Gonzalez-Barahona, J., Kurth, L., and Robles, G. (2018). Software development analytics for xen: Why and how. *IEEE Software*, pages 1–1.

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

Karvonen, T., Behutiye, W., Oivo, M., and Kuvaja, P. (2017). Systematic literature review on the impacts of agile release engineering practices. *Information and Software Technology*, 86:87–100. cited By 5.

Kaur, A. and Vig, V. (2019). On understanding the release patterns of open source java projects. *Advances in Intelligent Systems and Computing*, 711:9–18. cited By 0.

Kerzazi, N. and Robillard, P. (2013). Kanbanize the release engineering process. pages 9–12. cited By 3.

Khomh, F., Adams, B., Dhaliwal, T., and Zou, Y. (2015). Understanding the impact of rapid releases on software quality. *Empirical Software Engineering*, 20(2):336–373.

Khomh, F., Dhaliwal, T., Zou, Y., and Adams, B. (2012). Do faster releases improve software quality?: An empirical case study of mozilla firefox. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 179–188, Piscataway, NJ, USA. IEEE Press.

Kikas, R., Gousios, G., Dumas, M., and Pfahl, D. (2017). Structure and evolution of package dependency networks. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE.

Kitchenham (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University; University of Durham.

Kitchenham, B. (2004a). Procedures for performing systematic reviews. *Keele*, 33(1):1–26.

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

Kula, R. G., German, D. M., Ishio, T., Ouni, A., and Inoue, K. (2017a). An exploratory study on library aging by monitoring client usage in a software ecosystem. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE.

Kula, R. G., German, D. M., Ouni, A., Ishio, T., and Inoue, K. (2017b). Do developers update their library dependencies? *Empirical Software Engineering*, 23(1):384–417.

Laukkanen, E., Paasivaara, M., Itkonen, J., and Lassenius, C. (2018). Comparison of release engineering practices in a large mature company and a startup. *Empirical Software Engineering*, pages 1–43. cited By 0; Article in Press.

Leung, H. K. and Lui, K. M. (2015). Testing analytics on software variability. In *Software Analytics (SWAN), 2015 IEEE 1st International Workshop on*, pages 17–20. IEEE.

Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R., and Whitehead, E. J. (2013). Does bug prediction support human developers? Findings from a Google case study. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 372–381. IEEE.

Lungu, M. (2009). *Reverse Engineering Software Ecosystems*. PhD thesis, University of Lugano.

Malloy, B. A. and Power, J. F. (2017). Quantifying the transition from python 2 to 3: An empirical study of python applications. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE.

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

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

Mäntylä, M. V., Adams, B., Khomh, F., Engström, E., and Petersen, K. (2015). On rapid releases and software testing: a case study and a semi-systematic literature review. *Empirical Software Engineering*, 20(5):1384–1425. Had found "https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7006390", this paper is second reference on page 2.

Marsavina, C., Romano, D., and Zaidman, A. (2014). Studying fine-grained co-evolution patterns of production and test code. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, pages 195–204.

McDonnell, T., Ray, B., and Kim, M. (2013). An empirical study of API stability and adoption in the android ecosystem. In *2013 IEEE International Conference on Software Maintenance*. IEEE.

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201. ACM.

Mens, T., Claes, M., Grosjean, P., and Serebrenik, A. (2013). Studying evolving software ecosystems based on ecological models. In *Evolving Software Systems*, pages 297–326. Springer Berlin Heidelberg.

Messerschmitt, D. G. and Szyperski, C. (2003). *Software Ecosystem: Understanding an Indispensable Technology and Industry (MIT Press)*. The MIT Press.

Mirzaaghaei, M., Pastore, F., and Pezze, M. (2012). Supporting test suite evolution through test case adaptation. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 231–240.

Moiz, S. A. (2017). Uncertainty in software testing. In *Trends in Software Testing*, pages 67–87. Springer.

Ni, A. and Li, M. (2018). Acona: active online model adaptation for predicting continuous integration build failures. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 366–367. ACM.

Noor, T. B. and Hemmati, H. (2015). Test case analytics: Mining test case traces to improve risk-driven testing. In *Software Analytics (SWAN), 2015 IEEE 1st International Workshop on*, pages 13–16. IEEE.

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

Pinto, L. S., Sinha, S., and Orso, A. (2012). Understanding myths and realities of test-suite evolution. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 33. ACM.

Pinto, L. S., Sinha, S., and Orso, A. (2013). Testevol: A tool for analyzing test-suite evolution. In *Proceedings - International Conference on Software Engineering*, pages 1303–1306. Cited By :1.

Plewnia, C., Dyck, A., and Lichter, H. (2014). On the influence of release engineering on software reputation. In *Mountain View, CA, USA: In 2nd International Workshop on Release Engineering*. Had found "https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7006390", this paper is first reference on page 2.

Poo-Caamaño, G. (2016). *Release management in free and open source software ecosystems*. PhD thesis.

Raemaekers, S., van Deursen, A., and Visser, J. (2017). Semantic versioning and impact of breaking changes in the maven repository. *Journal of Systems and Software*, 129:140–158.

Rajlich, V. (2014). Software evolution and maintenance. In *Proceedings of the on Future of Software Engineering - FOSE 2014*. ACM Press.

Rausch, T., Hummer, W., Leitner, P., and Schulte, S. (2017). An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 345–355. IEEE Press.

Robbes, R., Lungu, M., and Röthlisberger, D. (2012). How do developers react to API deprecation? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*. ACM Press.

Robinson, B., Ernst, M. D., Perkins, J. H., Augustine, V., and Li, N. (2011). Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 23–32.

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J. M., and Oivo, M. (2017). Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, 123:263–291.

Santolucito, M., Zhang, J., Zhai, E., and Piskac, R. (2018). Statically verifying continuous integration configurations. *arXiv preprint arXiv:1805.04473*.

Schneidewind, N. F. (2007). Risk-driven software testing and reliability. *International Journal of Reliability, Quality and Safety Engineering*, 14(2):99–132. Cited By :10.

Shamshiri, S., Rojas, J. M., Galeotti, J. P., Walkinshaw, N., and Fraser, G. (2018). How do automatically generated unit tests influence software maintenance? In *Software Testing, Verification and Validation (ICST), 2018 IEEE 11th International Conference on*, pages 250–261. IEEE.

Souza, R., Chavez, C., and Bittencourt, R. (2015). Rapid releases and patch backouts: A software analytics approach. *IEEE Software*, 32(2):89–96. cited By 9.

Stolberg, S. (2009). Enabling agile testing through continuous integration. In *Agile Conference, 2009. AGILE'09.*, pages 369–374. IEEE.

Teixeira, J. (2017). Release early, release often and release on time. an empirical case study of release management. In *Open Source Systems: Towards Robust Practices*, pages 167–181, Cham. Springer International Publishing. Paper extracted manually from book at URL.

Teixeira, J., Robles, G., and González-Barahona, J. M. (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).

Thongtanunam, P., McIntosh, S., Hassan, A. E., and Iida, H. (2016). Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Proceedings of the 38th international conference on software engineering*, pages 1039–1050. ACM.

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., and Matsumoto, K.-i. (2015). Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 141–150. IEEE.

Trockman, A. (2018). Adding sparkle to social coding. In *Proceedings of the 40th International Conference on Software Engineering Companion Proceeedings - ICSE '18*. ACM Press.

Vassallo, C., Proksch, S., Zemp, T., and Gall, H. C. (2018). Un-break my build: Assisting developers with build repair hints.

Vassallo, C., Schermann, G., Zampetti, F., Romano, D., Leitner, P., Zaidman, A., Di Penta, M., and Panichella, S. (2017). A tale of ci build failures: An open source and a financial organization perspective. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, pages 183–193. IEEE.

Vernotte, A., Botea, C., Legeard, B., Molnar, A., and Peureux, F. (2015). *Risk-driven vulnerability testing: Results from eHealth experiments using patterns and model-based approach*, volume 9488 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Widder, D. G., Hilton, M., Kästner, C., and Vasilescu, B. (2018). I'im leaving you, travis: A continuous integration breakup story.

Xia, X., Lo, D., Wang, X., and Yang, X. (2015). Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 261–270. IEEE.

Zaidman, A., Van Rompaey, B., van Deursen, A., and Demeyer, S. (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):325–364.

Zampetti, F., Scalabrino, S., Oliveto, R., Canfora, G., and Di Penta, M. (2017). How open source projects use static code analysis tools in continuous integration pipelines. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*, pages 334–344. IEEE.

Zanjani, M. B., Kagdi, H., and Bird, C. (2016). Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering*, 42(6):530–543.

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., and Vasilescu, B. (2017). The impact of continuous integration on other software development practices: a large-scale empirical study. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 60–71. IEEE Press.