A Literature Survey of Software Analytics

IN4334 2018 TU Delft

2018-09-26

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.

4.2 Research Questions

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

4.3 Research protocol

Using the initial seed consisting of 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.

4.4 Answers

Through this we found the following papers

4.5 Summary of papers

4.5.1 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.

4.5.2 Beller et al. (2017b)

Initial Seed

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

4.5.3 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.

4.5.4 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.

4.5.5 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.

4.5.6 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.

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

4.5.7 Widder et al. (2018)

Initial Seed

This paper analyzes what factors have impact on abandonment of Travis. They find that increased build complexity reduces the chance of abandonment, but larger projects abandon at a higher rate and that a project's language has significant but varying effect. A surprising result is that metrics of configuration attempts and knowledge dispersion in the project do not affect the rate of abandonment.

4.5.8 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.

4.5.9 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.

4.5.10 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.

4.5.11 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.

4.5.12 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.

Chapter 5

Sample Sub-Topic

This is an example for the deliverable every group works on. Every group works on one independent chapter (starting as one Rmd file).

5.1 Motivation

A short introduction about why the topic you are working on is interesting.

The RQs that everyone should be aiming at are:

- **RQ1** Current state of the art in software analytics for *your topic* :
  - 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 *your topic* :
  - Tools and companies creating / employing them
  - Case studies and their findings
- **RQ3** Open challenges and future research required

5.2 Research protocol

Here, you describe the details of applying Kitchenham's survey method for your topic, including search queries, fact extraction, coding process and an intial groupping of the papers that you will be analyzing.

5.3 Answers

Aggregated answers to the RQs, per RQ. You need:

- For **RQ1**
  - Topics that are being explored
  - Research methods, tools and datasets being used
  - Main research findings, aggregated
- For **RQ2** :
  - Tools and companies creating / employing them
  - Case studies and their findings

- For **RQ3**:
  - List of challenges
  - An aggregated set of open research items, as described in the papers
  - Research questions that emerge from the synthesis of the presented works

Chapter 6

Sample Sub-Topic

This is an example for the deliverable every group works on. Every group works on one independent chapter (starting as one Rmd file).

6.1 Motivation

A short introduction about why the topic you are working on is interesting.

The RQs that everyone should be aiming at are:

- **RQ1** Current state of the art in software analytics for *your topic* :
  - 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 *your topic* :
  - Tools and companies creating / employing them
  - Case studies and their findings
- **RQ3** Open challenges and future research required

6.2 Research protocol

Here, you describe the details of applying Kitchenham's survey method for your topic, including search queries, fact extraction, coding process and an intial groupping of the papers that you will be analyzing.

6.3 Answers

Aggregated answers to the RQs, per RQ. You need:

- For **RQ1**
  - Topics that are being explored
  - Research methods, tools and datasets being used
  - Main research findings, aggregated
- For **RQ2** :
  - Tools and companies creating / employing them
  - Case studies and their findings

- For **RQ3**:
  - List of challenges
  - An aggregated set of open research items, as described in the papers
  - Research questions that emerge from the synthesis of the presented works

Chapter 7

Release Engineering Analytics

7.1 Search Strategy

Release engineering is a relatively new research topic, given that modern processes for releasing software (e.g. continuous delivery) are industry-driven. Therefore, we took an exploratory approach in collecting any literature revolving around the topic of release engineering from the perspective of software analytics. This will aid 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, five papers were given to us as a starting point for the 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 queries we entered are summarized 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 the search strategy that is based on a combination of keywords and subject headings as described above, a review of the list of publications that retrieved papers cite and are cited by is done. These lists are provided by Google Scholar, as well as the reference lists of the papers themselves. Results of which 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)

In order to aggregate our collective efforts, selected sources 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. By commenting on and tagging our findings we were able to export a filtered list of publications, the relevance of which was agreed upon.

Chapter 8

Sample Sub-Topic

This is an example for the deliverable every group works on. Every group works on one independent chapter (starting as one Rmd file).

8.1 Motivation

A short introduction about why the topic you are working on is interesting.

The RQs that everyone should be aiming at are:

- **RQ1** Current state of the art in software analytics for *your topic* :
  - 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 *your topic* :
  - Tools and companies creating / employing them
  - Case studies and their findings
- **RQ3** Open challenges and future research required

8.2 Research protocol

Here, you describe the details of applying Kitchenham's survey method for your topic, including search queries, fact extraction, coding process and an intial groupping of the papers that you will be analyzing.

8.3 Answers

Aggregated answers to the RQs, per RQ. You need:

- For **RQ1**
  - Topics that are being explored
  - Research methods, tools and datasets being used
  - Main research findings, aggregated
- For **RQ2** :
  - Tools and companies creating / employing them
  - Case studies and their findings

- For **RQ3**:
  - List of challenges
  - An aggregated set of open research items, as described in the papers
  - Research questions that emerge from the synthesis of the presented works

Chapter 9

Sample Sub-Topic

This is an example for the deliverable every group works on. Every group works on one independent chapter (starting as one Rmd file).

9.1 Motivation

A short introduction about why the topic you are working on is interesting.

The RQs that everyone should be aiming at are:

- **RQ1** Current state of the art in software analytics for *your topic* :
  - 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 *your topic* :
  - Tools and companies creating / employing them
  - Case studies and their findings
- **RQ3** Open challenges and future research required

9.2 Research protocol

Here, you describe the details of applying Kitchenham's survey method for your topic, including search queries, fact extraction, coding process and an intial groupping of the papers that you will be analyzing.

9.3 Answers

Aggregated answers to the RQs, per RQ. You need:

- For **RQ1**
  - Topics that are being explored
  - Research methods, tools and datasets being used
  - Main research findings, aggregated
- For **RQ2** :
  - Tools and companies creating / employing them
  - Case studies and their findings

- For **RQ3**:
  - List of challenges
  - An aggregated set of open research items, as described in the papers
  - Research questions that emerge from the synthesis of the presented works

Chapter 10

App Store analytics

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

M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, in Proceedings of the 2013 9th joint meeting on foundations of software engineering, 2013, pp. 477–487.

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.

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

G. Bavota, M. Linares-Vásquez, C. E. Bernal-Cárdenas, M. D. Penta, R. Oliveto and D. Poshyvanyk, in IEEE Transactions on Software Engineering, vol. 41, no. 4, pp. 384-407, 1 April 2015. doi: 10.1109/TSE.2014.2367027

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.

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

S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, in 2015 ieee international conference on software maintenance and evolution (icsme), 2015, pp. 281–290.

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

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.

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)*.

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.