A Literature Survey of Software Analytics

Moritz Beller, IN4334 2018 TU Delft

2018-09-24

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.

Moritz Beller

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

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).

3.1 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

3.2 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.

3.3 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 4

Final Words

We have finished a nice book on Software Analytics.

Chapter 5

App Store analytics

5.1 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.

5.2 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.

5.3 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 6

Build analytics

6.1 Background

When building a project from source code to executables everything should go smoothly. This is not always the case, a build can break for several reasons. This chapter will give an overview of research done on build scripts and continuous integration.

6.2 Research Questions

- What are causes of a broken build?
- With which goals is continuous integration applied?
- What can be used to effectively fix a broken build?

6.3 Search Strategy

Using the initial seed consisting of Bird and Zimmermann (2017), Beller et al. (2017a), Rausch et al. (2017), Beller et al. (2017b), 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.

6.4 Study Selection

Through this we found the following papers

6.5 Summary of papers

6.5.1 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.

6.5.2 Beller et al. (2017a)

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.

6.5.3 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. (2017a), albeit on a smaller scale. It does go more in depth on the result and changes over time.

6.5.4 Beller et al. (2017b)

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.

6.5.5 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.

6.5.6 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.

6.5.7 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.

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

6.5.8 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.

6.5.9 Vassallo et al. (2017)

References Beller et al. (2017a)

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.

6.5.10 Hassan and Wang (2018)

References Beller et al. (2017b)

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

6.5.11 Vassallo et al. (2018)

References Beller et al. (2017a), 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.

6.5.12 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 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.

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.

Beller, M., Gousios, G., and Zaidman, A. (2017a). 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. (2017b). 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.

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

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.

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.

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.

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.

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.

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.

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

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.

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.

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.

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

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.