Machine Learning and SonarQube KPIs to Predict Increasing Bug Resolution Times

Tom Gustafsson[1]

[1] Lappeenranta-Lahti University of Technology LUT, Finland

Abstract. Technical dept metaphor is widely discussed topic in research, but there is no common model on how to manage technical dept [3]. Companies invest a lot of money in maintenance and in commercial software system maintenance it is typical to have some penalties on missing the SLA deadline on bug resolution times. Adding technical dept can be understandable, in order to be quicker in markets with new features, but in order to manage it effectively, it is mandatory to understand the risks and impacts of the interest of it. In this research machine learning technology was used to evaluate whether SonarQube technical dept KPIs can be used to predict bug resolution times. The fact that the data was collected only from open source projects was limitation, but the results were encouraging. Accuracy approximately of 90% was reached. As it was seen that number of lines of code is also a valid indicator of bug resolution times, it was concluded, that it would be best to repeat this study in environment of commercial company which maintains many projects of similar size.

Keywords: Machine Learning, SonarQube, Technical dept, SLA, Software product maintenance.

1 Introduction

Technical dept metaphor is found attractive to practitioners as it communicates to both technical and nontechnical audiences that if quality problems are not addressed, things may get worse. In their research, Ernst et al. concluded that the even though technical dept is widely known and accepted, making it visible and measurable is a big gap in practice. Tools are installed, but the complexity of configuring them or interpreting results meant that they were unused. Only very small minority of business managers were actively managing technical debt. [3]

This paper focuses to investigate whether technical depth KPIs could be used together with machine learning in order to manage technical dept in software maintenance project in controlled manner.

Dataset [7] which was collected from various open source projects was used together with machine learning technologies in order to see, if it is possible to estimate based on technical dept KPIs, whether the bug fixing SLA thresholds are kept or not.

Limitations of this paper include the fact that strict SLA policies are more common in commercial software products under maintenance than in open source projects.

Also, the open source software products which were used in dataset [7] are collected from various stages of the software products lifecycle.

However, results indicated, that machine learning methodologies could estimate quite accurately, when bug fixing times start to have big probabilities of growing too big and outside of SLA limits. When using ROC-AUC as measure of how good the estimation is, excellent correlation was found. When simple number of lines variable was removed from variables, still good and close to excellent AUC numbers were found.

2 Background

Software maintenance has dramatically evolved in the last four decades in order to cope with the continuously changing development models. Maintenance is also an increasingly popular research topic, with an increasing number of new models and approaches being proposed. Interestingly, the number of models proposed is increasing rather than consolidating. The fact highlights that more research effort is needed to identify reusable and tunable models that can be applied in different contexts.[5]

In their work Lenarduzzi et al., presented results of a Systematic Literature Review, highlighting the evolution of the metrics and models adopted in the last forty years. Key findings included, that there is increase in the size of the data analyzed. One reason is due to the availability of an enormous open source code base that can be easily used to build maintenance models. Despite the open source ideology, it is nearly impossible to replicate the studies, because almost all papers are based on private data sets and/or used custom tools that are not available to other researchers to support the replication.[5]

2.1 Technical Debt Dataset

Researchers and industry are adopting various tools for static code analysis to measure technical dept and evaluate the quality of their code [7]. SonarQube is one of the most commonly used tools to support software maintenance [6]. When Lenarduzzi et al. compared it to other commonly used tools, it was found as an exception compared to other tools due to its increasing trend in popularity [6].

Empirical studies on software projects are expensive because it takes a lot of time to analyze the projects. Also, the results are difficult to compare as studies commonly consider different projects. In their work, Lenarduzzi et al [7] proposed the “Technical Debt Dataset”, a set of measurement data from 33 Java projects from the Apache Software Foundation. They analyzed all commits from separately defined time frames with SonarQube to collect Technical Debt information and with Ptidej to detect code smells. The Dataset includes also all available commit information from the git logs and fault information reported in the issue trackers (Jira). That information was used together with SZZ algorithm to identify the fault-inducing and -fixing commits. In the resulting dataset, one can find information about more than 78K commits from the selected 33 projects, approximately 1.9M SonarQube issues, 38K code smells, and

28K faults. The analysis took more than 200 days. In their paper, researchers also describe the data retrieval pipeline together with the tools used for the analysis. The dataset is available in CSV format as well as in SQLite database format to facilitate queries on the data. Aim of The Technical Debt Dataset is to open diverse opportunities for Technical Debt research, enabling researchers to compare results on common projects. [7]

3 Reasoning

At the level of IT management in industry, organizations are interested in the economic consequences of technical debt and risks that they may pose. Technical debt can decrease efficiency of running software systems and create difficulties in extending them. Cost overhead in fixing issues or adding new functionality caused by technical debt is considered as the interest of technical debt. In economic perspective, technical debt is defined as the cost of repairing quality issues in software systems to achieve an ideal quality level. An ideal quality level is the highest achievable level of quality defined in a quality model adopted by an organization. The amount of debt is the gap between the current and the ideal level. Interest is defined as the extra maintenance cost spent for not achieving the ideal quality level. Maintenance activities include adding new functionality and fixing bugs. Maintenance and technical quality repair action is different in that former involves visible changes and their impacts are immediately visible. Extra effort spent on new functionality or fixing bugs are examples of interest on technical debt. Interest of technical debt is not the same as maintenance costs, because systems without technical issues will still spend some effort on maintenance.[7]

In software development and maintenance cost point of view, the earlier bugs are found, the cheaper it is. On the other hand, companies want to be fast in market. But faster time-to-market and quick user feedback also implies less time for testing and bug fixing in early stages [2].

Code with bad quality is more expensive to maintain, but also refactoring is risky. It requires changes to working code that can introduce subtle bugs. Refactoring, if done wrongly, can set you back days or weeks. Refactoring becomes riskier when practiced informally or ad hoc.[1]

If there is a way to apply machine learning techniques to identify problems earlier, there is a good motivation to include it already in CI/CD pipeline, using the SonarQube metrics and technical Dept information to forecast problems. This could help organizations to balance between time to refactor an be fast in time-to-market. Using the open dataset described collected from various open source projects [7], this paper is focusing on SonarQube technical dept metrics and their relationship to actual bug resolution times reported to issue tracking tool JIRA.

If the technical dept KPIs can predict future bugs, those can be used as good indicator for refactoring purposes. **RQ: Can SonarQube technical depth KPIs be used to estimate increasing bug resolution times?**

3.1 Study Design

In this study, the dataset [7] collected from various open source projects was used. Machine learning methods were used to figure out, if the technical dept KPIs can be used to estimate longer bug resolution times. Bug resolution time, in commercial maintenance processes, usually have SLAs, which set limits on how quickly bugs should be fixed. For this reason, in this study bug resolution time was used as independent variable. Several Technical dept KPIs were used as dependent variables and python scripts were run against the data in order to get the results with prediction KPIs to determine whether the prediction was accurate.

3.2 Data collection

Technical depth, by definition, impacts to the time which is used to add features or fix bugs. In the used dataset [7], faults table includes the JIRA issues with "bug" as a type. The timestamps, which are available are the creation time in JIRA as well as the resolution time in JIRA. With those timestamps, elapsed time from the moment bug was reported in JIRA, to the moment when the bug was closed in JIRA was calculated. Each bug was given a resolution time with this method. 30 days resolution time was used to represent SLA violation, too long resolution time. This research only investigates bugs reported in JIRA, as those would best represent the problems in commercial software in sense that the more severe the bug is, the quicker it will get fixed. In their research on technical dept diffuseness, Saarimäki et al [4] interestingly pointed out that in technical dept items, the more severe the finding is, the longer in remained unresolved.

The dataset [7] includes table named "sonar_measures", where one can find the technical depth KPIs. For technical dept KPIs, monthly figures were used, because there is no single number to represent the dept for the period of bug fixing. Monthly figures included: the analysis month, max sqaleindex, max sqaledebtratio, max scalerating, max lines of code, max securityremediationeffort, max reliabilityremediationeffort and the security and reliability remediation effort per 1000 lines of code.

Since number of lines of code is easy KPI to predict longer resolution times, it was decided to execute the python scripts with lines of code included as one KPI and then collecting another set of data with same methodology and taking away the number of lines of code, in order to verify if the results differ when the most evident single KPI which predicts bug resolution times was removed.

4 Limitations

This study is using big dataset [7] as input for collecting variables for machine learning algorithms use. However, the projects used for the dataset are all open source projects with different ways of working different policies on how quick to fix bugs. Also, the selected open source projects are from projects which vary a lot in terms of size of the project and the products phase in its lifecycle.

The research also relies on monthly averages of technical dept measures while it would be best to analyze the dept KPIs from the exact moment when bug is found. As the data comes from various phases of the products lifecycle, one can find very short bug resolution times, which are due to the fact which can be seen on commit dates compared to bug opening and closing times, that some bugs are probably found during development, fixed at the same time and the bug is reported afterwards.

5 Results

First executions of the python scripts against the data was executed three times. The dataset was balanced so, that there was just as many very long bug resolution times included as there were bugs with resolution times less than 30 days. Random forest classifier algorithm was used. Three executions gave following results:

1st run:

```
TN: 86 - FP: 26 - FN: 2 - TP: 111
Precision: 0.8756; MCC: 0.7686
F1: 0.8756; AUC: 0.8976%
```

2nd run:

```
TN: 82 - FP: 22 - FN: 1 - TP: 120
Precision: 0.8978; MCC: 0.8062
F1: 0.8978; AUC: 0.9197%
```

3rd run:

```
TN: 89 - FP: 27 - FN: 6 - TP: 103
Precision: 0.8533; MCC: 0.7206
F1: 0.8533; AUC: 0.8952%
```

Area under curve (AUC) shows that the random forest classifier gave predictions, which were between 90 and 92% accurate. This gives an idea, that technical dept KPIs do have strong impact to the bug resolution times. However, one of the used KPIs from dataset [7] was the number of lines of code. It can be obvious, that the bigger the project grows, the more it takes to implement changes. Therefore, the second set of executions were using the exact same parameters, except the number of lines of code was removed from the list of dependent variables.

1st run:

```
TN: 76 - FP: 35 - FN: 7 - TP: 107
Precision: 0.8133; MCC: 0.6458
F1: 0.8133; AUC: 0.8836%
```

2nd run:

```
TN: 72 - FP: 38 - FN: 9 - TP: 106
Precision: 0.7911; MCC: 0.6001
F1: 0.7911; AUC: 0.8849%
```

3rd run:

```
TN: 72 - FP: 34 - FN: 12 - TP: 107
Precision: 0.7956; MCC: 0.5969
F1: 0.7956; AUC: 0.8569%
```

When comparing the results against the values with the number of lines of code icluded, the AUC number is still giving quite good accuracy from 86 to 88 percentages. MCC number shows also, that the classifier can be used in estimations. However, lines of code do boost the accuracy of the estimation, which was also logical guess at the beginning of the study.

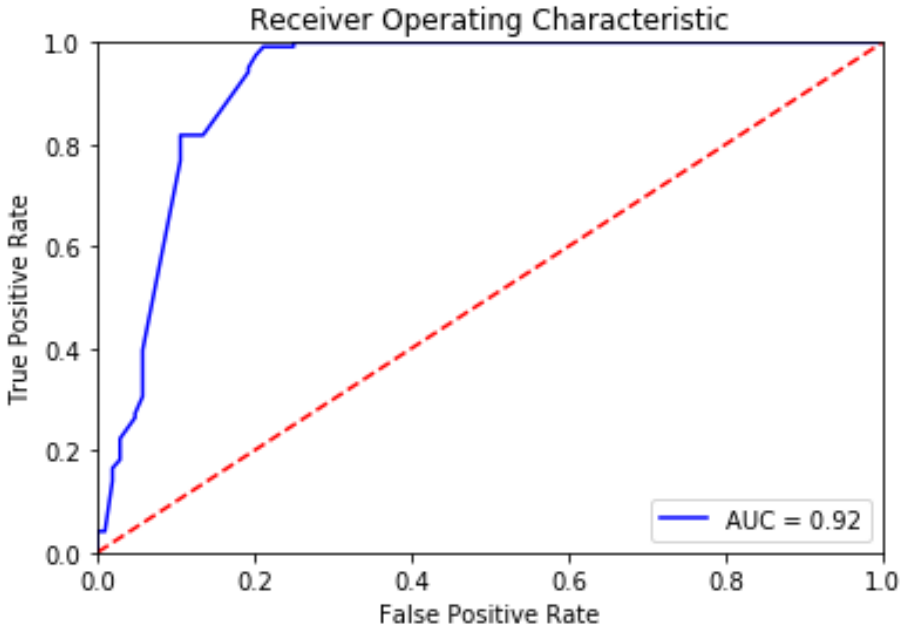

Fig. 1. Best AUC of first tests included AUC = 0.92, while the other two runs had AUC 0.90

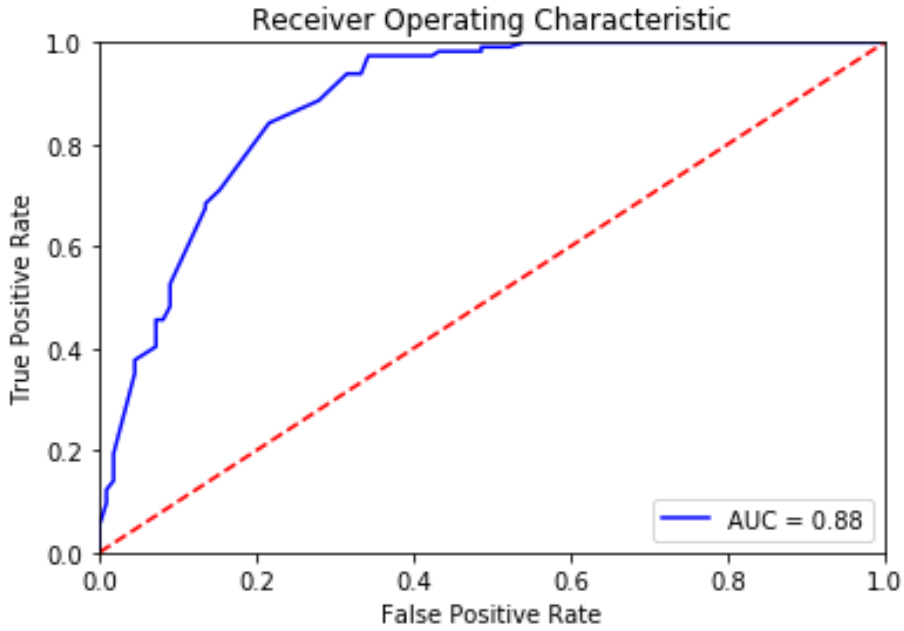

Fig. 2. Best AUC of second execution set was 0.88, while the other runs gave AUC = 0.88 and AUC = 0.86

6 Discussion

One can make conclusion from the results, that with the help of machine learning classifiers one can build a model to predict SLA violations. Research question was: **Can SonarQube technical dept KPIs be used to estimate increasing bug resolution times?** Research shows that technical dept KPIs can be used for this purpose, however, technical dept KPIs by themselves did not work as good as used together with simple number of lines of code KPI. Results are encouraging, taking into concern that all bugs were treated equal and that sizes of the projects did not impact the results too much.

It is obvious that the size of the project impacts on the bug resolution times. Also, it is normal that severity of the bug impacts on SLAs on commercial products. Therefore, in following research the projects themselves should be classified by size and bugs separately based on severity. Then machine learning models could be trained against the set which includes data from similar sized projects. This way the model it could be possible to build logic in CI/CD pipeline to predict SLA violations. Further research is needed to replicate similar study in environment, which includes several similarly sized commercial software products under maintenance. As a result, it could be possible to build model which gives good reasons for refactoring.

7 Conclusion

Technical dept, by definition, impacts to the time which is used to add features or fix bugs [7]. Sometimes it is good to have technical dept, because it is important to focus resources on new features and to be quick in market [2]. The interest of technical dept can be seen in bug resolution times and in commercial software systems, it is typical to have SLAs which needs to be kept in order to avoid penalties. There is no industry standard way of fixing technical dept issues and fixing those ad-hoc is also risky [1]. One way of managing the costs of technical dept could be utilizing machine learning models in CI/CD pipeline and use the predictions which algorithms give as triggers for paying technical dept.

Dataset, which consist of 33 projects, approximately 1.9M SonarQube issues, 38K code smells, and 28K faults [7] was used to form a dataset for random forest classifier, which was used to study, whether it is possible for model to learn predict SLA violations. Various technical dept KPIs were used together with number of lines of code and bug resolution times to teach the model.

Results showed, that the model can predict bug resolution times to pass or meet 30 days threshold quite accurately. With all the selected KPIs included, the model predicted correct results with 90 – 92 percentage accuracy. Lines of codes was obvious easy single KPI to predict the resolution times, since it is assumed that the bigger the project is, the more time it takes to make changes. When the impact of number of lines of codes was taken away, accuracy remained high, 86 – 88 percentages. Research concludes, that machine learning models could be used to predict software products bug resolution times.

Limitations were seen also. Even the dataset [7] is big, it consists only of open source projects and open source projects may not have as strict SLA handling as commercial products may do. Also, the projects may have different ways of working compared to other projects and therefore the best possible environment for this kind of study would be a set of data from one company, consisting from various software products with similar size.

References

1. Fowler, M., Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.
2. M.V. Mäntylä, B. Adams, F. Khomh. et al. Empir Software Eng (2015) 20: 1384.
3. N. Ernst, S Bellomo, I. Ozkaya, R. Nord and I. Gorton. “Measure it? Manage it? Ignore it? software practitioners and technical debt.” In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). ACM, New York, NY, USA, 2015, pp. 50-60.
4. N. Saarimäki, V. Lenarduzzi, and D. Taibi, “On the diffuseness of code technical debt in open source projects of the apache ecosystem,” International Conference on Technical Debt (TechDebt 2019), 2019, pp 98-107
5. V. Lenarduzzi, A. Sillitti and D. Taibi, "Analyzing Forty Years of Software Maintenance Models," 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, 2017, pp. 146-148.
6. V. Lenarduzzi, A. Sillitti, and D. Taibi, "A survey on code analysis tools for software maintenance prediction," in Software Engineering for Defence Applications — SEDA, 2019.
7. V. Lenarduzzi, N. Saarimäki, and D. Taibi. The Technical Debt Dataset. Proceedings of the 15th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE’19), September 18, 2019, Recife, Brazil. https: //doi.org/10.1145/3345629.3345630 (3)