The final publication can be accessed under the DOI: https://doi.org/10.1145/3340495.3342752

Integrating Runtime Data with Development Data to Monitor External Quality: Challenges from Practice

Aytaj Aghabayli
aytaj.aghabayli@ut.ee
University of Tartu
Tartu, Estonia

Dietmar Pfahl
dietmar.pfahl@ut.ee
University of Tartu
Tartu, Estonia

Silverio Martínez-Fernández
silverio.martinez@iese.fraunhofer.de
Fraunhofer IESE
Kaiserslautern, Germany

Adam Trendowicz
adam.trendowicz@iese.fraunhofer
Fraunhofer IESE
Kaiserslautern, Germany

ABSTRACT

The use of software analytics in software development companies has grown in the last years. Still, there is little support for such companies to obtain integrated insightful and actionable information at the right time. This research aims at exploring the integration of runtime and development data to analyze to what extent external quality is related to internal quality based on real project data. Over the course of more than three months, we collected and analyzed data of a software product following the CRISP-DM process. We studied the integration possibilities between runtime and development data, and implemented two integrations. The number of bugs found in code has a weak positive correlation with code quality measures and a moderate negative correlation with the number of rule violations found. Other types of correlations require more data cleaning and higher quality data for their exploration. During our study, several challenges to exploit data gathered both at runtime and during development were encountered. Lessons learned from integrating external and internal data in software projects may be useful for practitioners and researchers alike.

CCS CONCEPTS

• **Information systems** → *Data analytics*; *Data mining*.

KEYWORDS

Software quality, software runtime data, external quality, software analytics, CRISP-DM

ACM Reference Format:
Aytaj Aghabayli, Dietmar Pfahl, Silverio Martínez-Fernández, and Adam Trendowicz. 2019. Integrating Runtime Data with Development Data to Monitor External Quality: Challenges from Practice. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on Software Qualities and Their Dependencies (SQUADE '19), August 26, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3340495.3342752

https://doi.org/10.1145/3340495.3342752

1 INTRODUCTION

There has been rapid growth in the use of data analytics to support evidence-based software engineering [14, 20]. Modern software development relies on short feedback cycles as a way to provide flexibility and rapid adaptation to market fluctuations. Practitioners need to investigate software runtime quality problems during the early software development stages, for instance, for guiding refactoring activities [17]. In this context, decisions are also made more frequently and the use of software analytics is particularly attractive [13].

Several studies have reported that the use of data analytics is particularly attractive for software companies [6, 13]. Indeed, companies like Microsoft are hiring data scientists for their software teams [8].

Still, in a recent survey, Svensson et al. showed that future research should develop new automated data collection, analysis, and visualization techniques and methodologies that augment existing decision-making processes by linking relevant data to specific decision contexts [4]. In the context of software quality, relevant data is heterogeneous and can be classified according to several linked quality factors [16]. These quality factors can be external or internal by nature. The integration of the heterogeneous data sources has been identified as a research gap by numerous researchers [2, 12, 19]. For example, it is not obvious how data representing the internal quality of software (e.g., source code metrics, test coverage) is related to data representing the (external) quality in use (e.g., runtime exceptions, most used functionalities).

To address this gap identified in the research literature, the main goal of our study is to explore the integration between runtime errors and crashes during the usage of software systems with development data from software repositories. The related research question (RQ) is as follows: "How can we integrate runtime data with development/repositories data (internal software quality) to improve software quality?"

To answer this RQ, we investigated the possibilities of integrating runtime data with development data in a specific case context. We wanted to find out how useful it would be in the given project context to integrate software runtime data with development data in order to understand and predict external quality. In other words, we tried to understand whether problems occurring during the use of the software (external quality) are caused by problems during the development of the software (internal quality). As conceptualized

in the ISO 25010 standard [7], we expected a connection among internal, external and quality in use. Such connection could be useful for three purposes.

First, to improve quality factors (e.g., stability, functional suitability, usability) based on correlation patterns of runtime and development data. Second, to assess and predict external quality from known internal quality based on the aforementioned patterns. For instance, practitioners can benefit from preprocessed, integrated and analyzed raw data (e.g., ability to identify/predict the source of runtime exception before fatal errors occur in production). Third, to derive data-driven software quality models, since experts-based software quality models are usually costly [11].

Due to the difficulty of finding commercial software data sets[1] for doing research in software analytics, most research studies are based on open source software data. In contrast to this, we conducted a case study and chose as the unit of analysis a Fraunhofer IESE internal project, hereafter referred to as 'ACME', a platform providing several digital services. Our goal is to explore the integration of runtime and development data. We used the CRISP-DM process method [18], as the cross-industry process for data mining. We decided to use the CRISP methodology because it is a de-facto standard widely used in data analysis problems. It provides a well-defined structure for planning data-driven projects.

The process consisted of following three steps:

(1) Identification of data sources (data understanding).
(2) Identification of key features to integrate the data (data understanding).
(3) Exploration of interrelationships among data (data preparation and modeling).

This paper is structured as follows. Section 2 describes related work. Section 3 describes the solution framework. Section 4 describes in detail the data analysis done for the integration of runtime data with development data to monitor external quality in a case. Section 5 presents challenges and lessons learned. Finally, Section 6 concludes the paper and present future work.

2 RELATED WORK

There exist few empirical studies investigating into the integration of software runtime and system and process data. So far, not a single set of correlated metrics across different projects has been found [5].

Nagappian et al. reported the prediction of component failure based on mining metrics [15]. Five different software systems of Microsoft were analyzed. Spearman correlation method was applied to identify a set of complexity metrics, which are correlated with post-release defects.

Lautenschlager et al. studied cloud software systems. They investigated the root cause identification of runtime data collected from different parts of the software and argue that it is important to integrate and centralize all data into one tool [9]. Analysis tools such as Zipkin, Prometheus, Grafana, fluentid, Elasticsearch[2], and Kibana[3] (exploring log files) were used. They indicated that it was not easy to integrate runtime software data collected from different sources, as there is a need to have a common component (e.g. naming, log structure or sharing timeline). As a result, they created a chatbot, where all runtime data collection, storing and analyzing tools were combined.

Most of the previous studies have focused on either the analysis of the relation between software quality and runtime data or the analysis of the relation between software quality and data collected during development time. Mostly, researchers conducted analyses on completed projects, which makes the data preparation process easier. To actually help developers in on-going projects and give them real-time tool support, there is a need for further research on the integration of real project runtime and development data and its analysis.

3 SOLUTION FRAMEWORK

Our goal is to collect runtime data and analyze its relationship with software development data. For the purpose of data collection, we extended the existing Q-Rapids solution [10] with two connectors to gather real-time data from crashes in HockeyApp[4] and logs in Amazon CloudWatch[5]. We used existing connectors to gather data from Jira[6], Git[7] and SonarQube[8] [1]. The solution framework is depicted in Figure 1. The data flow is as follows. First, data is automatically generated during software development and software usage. Second, the connectors gather data in real-time. Third, data is stored in elastic. Fourth, real-time dashboards offer information to stakeholders. An example of a dashboard to supporting decision-making in the software development process, is shown in Figure 2. This particular dashboard shows in real-time the most occurred crash reasons associated to the software components. Therefore, it aims at helping practitioners better understand the progress in crash solving and prioritize tasks during sprint planning. Fifth, the data is integrated and analyzed.

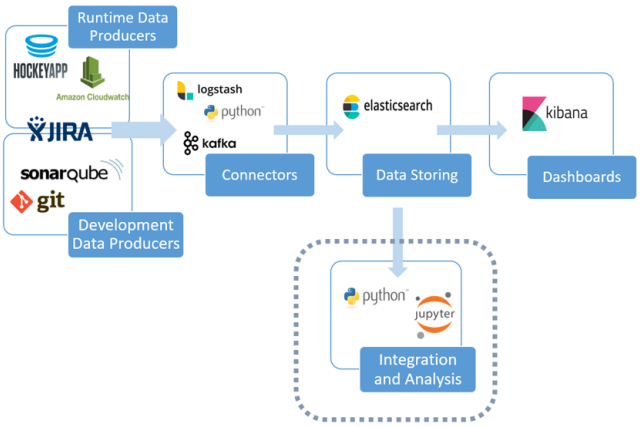

Figure 1: Solution framework (adapted from Q-Rapids).

[1] https://index.co/program/data-market-services
[2] Elasticsearch: https://www.elastic.co/
[3] Kibana: https://www.elastic.co/products/kibana/
[4] HockeyApp: https://hockeyapp.net/
[5] CloudWatch: https://aws.amazon.com/cloudwatch/
[6] Jira: https://www.atlassian.com/software/jira
[7] Git: https://git-scm.com/
[8] Sonarqube: https://www.sonarqube.org/

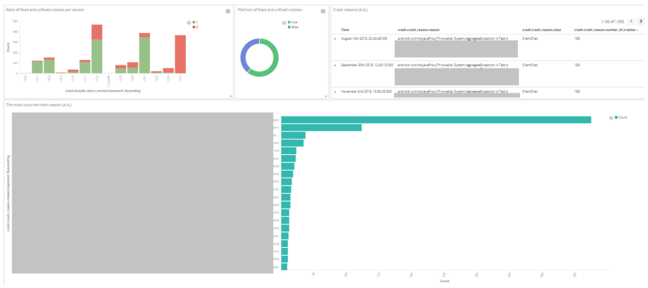

Figure 2: Dashboard example.

Figure 3: CRISP-DM.

In this paper, we focus on the presentation of the integration and analysis step in Figure 1. For this, we used the Cross-Industry Standard Process for Data Mining (CRISP-DM) [18] as a reference process (see Figure 3) and provided the necessary Python scripts, which we made publicly available [3].

4 APPLICATION OF THE SOLUTION FRAMEWORK

The application case starts with the *Business Understanding* phase in which the research goals and business expectations of the Q-Rapids project were translated into the specific objectives of data analysis. One of the basic objectives was to learn quality dependencies (i.e., a quality model) from the software development and runtime data. Particularity interesting were dependencies between characteristics of the software as well as the software development process and its behavior in runtime. In the following sections, we focus on the subsequent CRIPS-DM phases: *Data Understanding, Data Preparation, and Modeling.*

4.1 Data Understanding

Objectives of *Data Understanding* were to gain an overview of the available software quality data in the context of the ACME project, to understand the data, and to identify possibilities to integrate it for the purpose of analysis. In the first step, we identified potentially relevant data sets and features within each set. Next, we assessed the quality of the relevant data and identified quality issues we needed to address prior to the analysis.

Relevant Data Sets. The relevant, in the scope of the research, data sets collected from the ACME project were grouped into external and internal quality aspects as follows:

- **External quality data sets**, collected from the runtime of the software:
  - **Access logs**: all HTTP server access requests of the project. The data were collected from the Amazon CloudWatch platform.
  - **Error logs**: all HTTP server errors of the project. The data were collected from the Amazon CloudWatch platform.
  - **Crashes**: android application crashes occurred at runtime. The data were collected from the HockeyApp platform.
  - **Sprint issues**: all reported bugs, tasks and change requests. The data were collected from the tool Jira.
- **Internal quality data sets**, collected from the development process of the software:
  - **Code quality measures**: metrics evaluations for either each file, directory, or module. The data were collected from the SonarQube tool.
  - **Quality rule violations**: rule violations for either each file, directory, or module. The data were collected from the SonarQube tool.
  - **Commits**: source code changes in the files. The data were collected from the version control system (Git).

Relevant Features. In this step, we described relevant features from runtime (see Table 1) and development data sets (see Table 2) to explore their integrations. We identified potentially relevant features for quality modeling: (1) identifiers required for data integration; and, (2) features capturing potentially relevant quality aspects.

First, as a key artifact for the runtime data, we extracted the 'timestamp' field from each data set. Then, we investigated specific factors for individual data sets. The relevant specific features and their descriptions are the following:

- From access logs, we extracted 'request' (HTTP request) and 'response code'. As stakeholders from ACME needed to know how the software was accessed. This can be distinguished by request name and response code. The 'request' is a key factor in possible integrations. The initial idea was to integrate access logs with development data sets by 'request'.
- From the data set error logs, we selected 'error type'. This field is the only granularity factor in the data set and has a description of the occurred errors.
- From data set crashes 'crash reason', 'status' and 'class' were considered as relevant features. The 'crash reason' filed shows the full description of the classes and reason for the failed crashes. We consider this feature important to be able

Table 1: Runtime data sets.

Data Set	Relevant Futures	Origin
Access logs	timestamp, request, response code	HTTP access logs (CloudWatch)
Error logs	timestamp, error type	HTTP error logs (CloudWatch)
Crashes	timestamp, crash reason, status, class (if exists)	Application crashes (HockeyApp)
Sprint issues	timestamp, issueid, issuetype	Issue tracking system (JIRA)

Table 2: Development data sets.

Data Set	Relevant Futures	Origin
Code quality measures	timestamp, path, metric, value	Static code analysis tool (SonarQube)
Quality rule violations	timestamp, path, rule	Static code analysis tool (SonarQube)
Commits	timestamp, path, issues	Version control system (Git)

to connect the data set with development data. The 'status' indicates the crash was solved or not. For some samples, the feature 'class' existed, which is the direct indicator of the crashed code class name.
- Relevant features of data set sprint issues are 'issueid' and 'issuetype'. The field 'issueid' was extracted as a key factor in the integration with commits. We needed the feature 'issuetype' to be able to distinguish bugs from stories, tasks, and change requests.

First, from each development data set we selected factors representing the granularity of the data, i.e., 'timestamp' and 'path' (path to the file or class). Then, fields representing quality aspects were extracted:

- In data set 'Code quality measures', the relevant features were 'metric' and 'value'. 'Metric' indicates the quality measure type and 'value' indicates the quantitative measure of the metric.
- Extracted feature from the 'Quality rule violations' data set was 'rule'. This field gives information about rule violations detected by the static code analysis tool.
- From the 'Commits' data set, we extracted the field 'issues', which is the list of the solved issues for each code change. This factor is needed to be able to integrate the data set with sprint issues.

Quality of Data. In the collected data, we had the following issues, which limit to some degree the integration of data sets:

- **Incomplete information** - missing values in the data sets. For example, crashes had a field 'class name' which was missing in most of the samples.
- **Data inconsistency** - data sets have differently structured 'path' fields.
- **Inaccurate data** - not all data sets correspond to the correct software version and code segments.

Figure 4 shows the data sets which were not possible to integrate owing to the indicated quality problems. The green circles show the relevant development data sets, the red - relevant runtime data sets. The integration of logs and crashes with development data were inapplicable due to insufficient consistency and accuracy of data:

- The available commits data set did not contain changes to the application source code repositories.
- Access and error log data sets do not have a direct link with the development of data granularity factors.

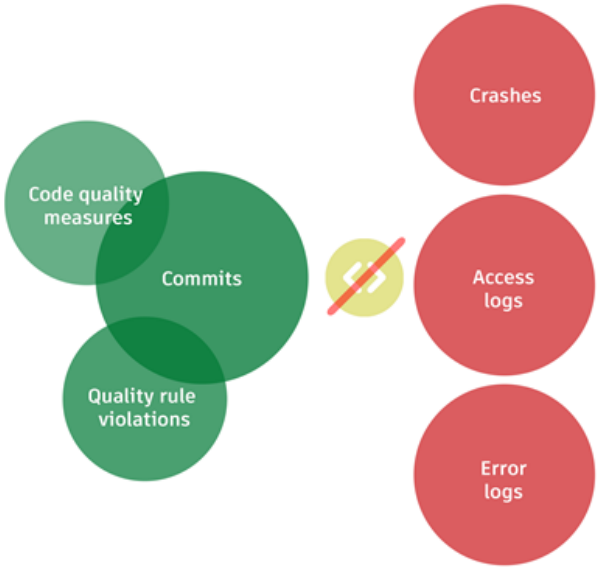

Figure 4: Inapplicable integrations.

4.2 Data Preparation

The objective of the data preparation phase was to prepare the identified data for analysis. This included resolving data quality deficits (data cleaning), deriving additional features based on the features available in the data (feature engineering), and combining data available in distinct sources (data integration).

Data Cleaning. Cleaning data started with filtering data sets from non-used features and rows. The extracted rows are the following:

- From 'Sprint issues' data set: reported bugs
- From 'Commits' data set: commits solving an issue

Then, we cleaned data sets from duplications and missing values, to resolve the incomplete information issue. Finally, to solve the data inconsistency issue, we constructed - identical across all data sets - 'path' and 'timestamp' structures. After data cleaning, we conducted the feature engineering process described in the following subsection.

Feature Engineering. The primary objective of feature engineering was to adjust existing or derive new data features to boost data analysis. This was helpful, because, on the one hand, derived features contributed useful information for the analysis and, on the other hand, helped resolve minor data quality issues. To support analysis, we generalized code measurement data in by aggregating it per specific time interval. For each metric, we calculated the average of the metric values per each file during a specific period. Furthermore, we grouped quality rule violations by source code files to derive the total 'violations count' per unique file.

In the 'Commits' data set, we reconstructed the field 'issues' and put each issue in a separate row.

In the 'Sprint issues' data set, we conducted feature engineering after integration with the 'Commits' data set. We created a new field 'number of bugs' and per each file, we calculated an overall number of occurred bugs. Integrated files which had no bugs were considered as having zero bugs.

Integration. Figure 5 shows the integrated data sets and integration approach. The green circles indicate the relevant development data sets, the red - the relevant runtime data sets. We integrated commits with bugs by 'issue_id', as each commit contains the 'id' of the solved issues from the bug reporting system. Due to the absence of file (class) path in data set 'Sprint issues', direct integration of bugs with code quality measures and quality rule violations was impossible. We used an indirect approach to integrate the number of bugs with development data. After the integration of commits, we integrated code quality measures and quality rule violations by a file (class) path with the commits. In that way, we indirectly integrated bugs with static code quality measures and rule violations.

As a conclusion, we integrated bugs directly with commits and indirectly with static code quality measures and rule violations.

4.3 Modeling

The objective of the modeling phase was to analyze data and accomplish objectives defined in the business understanding phase. In particular, we aimed at gaining knowledge about quality dependencies of software during development (internal quality) and runtime (external / in-use quality).

After the integration of the bugs with code quality measures, quality rule violations and commits, we performed analysis on the integrated data. The purpose was to analyze to what extent we can use development data in the identification of software runtime quality problems.

The hypotheses were the following:

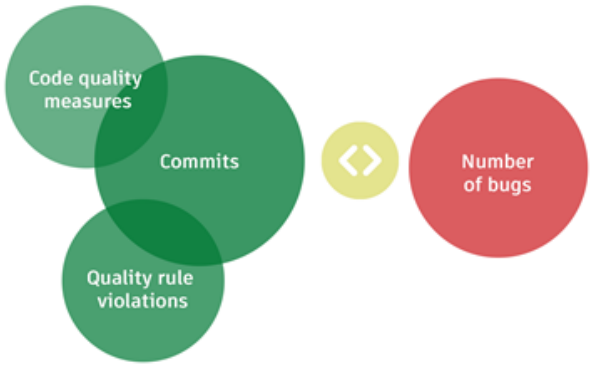

Figure 5: Integration approach.

(1) The number of bugs occurred in the file is positively correlated with the code quality measures of the file.
(2) The number of bugs occurred in the file is positively correlated with the number of quality rule violations founded in this file.

Considering data quality aspects to identify dependencies between data sets, we decided to apply correlation analysis. Thus, we calculated Spearman correlation coefficients (between '-1' - negatively correlated and '1' - positively correlated) based on the integrated data sets. Regarding our first hypothesis, Figure 6 presents the correlation coefficients between the number of occurred bugs and code quality measures of the files for each code quality metric. The results indicate only weak correlations. The number of bugs has a weak positive correlation with the number of functions, lines, and net lines of code (ncloc). There are very weak correlations with other metrics.

Figure 6: Spearman correlation coefficients between the number of bugs and code quality measures..

Regarding our second hypothesis, the correlation between the number of occurred bugs and the number of rule violations was -0.41. Surprisingly, the result indicates a moderate negative correlation between the data sets, which is contradictory to our hypothesis. Further research is needed to investigate the reason.

A detailed report of our analyses as well as the Python scripts used have been made publicly available [3].

5 CHALLENGES AND LESSONS LEARNED FROM PRACTICE

In our study we faced a number of challenges related to data quality. Besides a few evident flows in the data collection, which for instance, resulted in incorrect data, the major source of data quality related problems was associated with lacking fitness-for-use. This observation was consistent with experience we previously had gained in other contexts where we applied machine learning methods to data from software projects with the aim to gain actionable knowledge about software quality. In all cases, we had to invest a significant amount of effort to prepare data before it could be analyzed at all, and still, the quality of the resulting data was relatively poor. Most of those challenges were caused by the fact that the data was not generated with the purpose of data-driven quality of mind.

Therefore, the general lesson we learned is the following:

Wherever possible, data should be collected with a specific purpose in mind because post-mortem retrofitting of the collected data to the actual problem to be solved might be very expensive or even impossible.

The rest of this section develops the above general lesson learned and summarizes the specific examples of data quality challenges and the approach we took to handle them.

Data Completeness. One challenge was that large parts of the source data were partially missing. For example, information on issue type stored in the issue tracking system was largely missing. In consequence, for many reported issues it was not clear whether a given issue referred to the development of a new software feature or it was associated to quality problem, e.g., bug. One more example of incomplete runtime data was crash reports, in which only a few crashes contained a code class name field. This particular field would be useful in the integration with code quality aspects.

Further data completeness problems occurred after data integration due to misalignment of the individual data sources. On the type of misalignment was when corresponding data were not consistently available in the integration data sets. For instance, the change management system contained commit information on only already resolved issues. In consequence, integration of the commit data with issue tracking data resulted in the loss of data on non-solved bug issues. In this case, data incompleteness resulted from the nature of the data, rather than being caused by deficient data collection. An example of incompleteness due to flows in data collection would a situation we faced during the integration of application crashes with development data, where available data referred to different parts of the software. The code quality measures were not collected for the parts of the software system to which the runtime crash reports corresponded.

Error logs contained only a general description of the issues. We could not find any direct or indirect connection with development data. Thus, there were no clear integration possibilities.

Data Accessibility. Even though, in several cases, the quality of the data could be improved by retrospective measurement of software artifacts we could not access them. For example, during the research, we did not have access to the ACME project android application source code due to confidentiality reasons. Furthermore, access logs data set has entire request queries, which possibly can be integrated with internal quality aspects, by considering the class handled this request in the source code. The access to particular parts of the source code during the study.

Data Consistency. In different data sets, semantically corresponding features had a different structure and/or formatting. For example, the path to the files in the commits data set and bugs had a different pattern. The code quality measures data set contained additionally the platform and service name contrary for commits. To handle this issue after analyzing the data sets individually, we selected a common pattern and produced feature engineering based on it. Thus, we structured the field 'path' from all data sets to the identical structure. The data preparation process, in particular integration, would be easier if the data sets were constructed consistently.

Data Correctness. We faced the problem of having data sets corresponded to different time periods. For example, code quality measures and commits data sets were collected for the different time periods. Therefore, we spent a lot of time on finding the correct data sets from the static code analysis tool. It is important to collect the data sets in the correct way beforehand.

6 CONCLUSIONS AND FUTURE WORK

Our study has two main contributions. First, we applied the CRISP-DM method to integrate software runtime and development data. The scripts are reusable and available on [3]. We could integrate runtime data and development data in two cases (number of bugs with code quality measures and number of rule violations). More integrations require more data preparation. We can conclude that more efforts are needed to create high-quality data at runtime and during development for its later integration and analysis

Second, we reported challenges and lessons learned from the integration of software runtime and development data can be used for further research. We learned that wherever possible, data should be collected with a specific purpose in mind because post-mortem retrofitting of the collected data to the actual problem to be solved might be very expensive or even impossible.

Future work will contain solving challenges in the integration of application crashes and access logs with development data sets. We will conduct research as well on improving the quality of the data sets in software engineering. Another challenge is to combine data-driven studies with experts' knowledge, which might help to find out some of the correlations that are not clearly visible in the data. However, the poll of methods available for combining subjective expert's knowledge and quantitative data is limited, e.g., Bayesian belief networks.

ACKNOWLEDGMENTS

This research partially was made possible by the Erasmus+ traineeship grant. We also thank the European Union's Horizon 2020 research and innovation program under grant agreement No 732253 (Q-Rapids). This research was partly supported by the institutional research grant IUT20-55 of the Estonian Research Council and the Estonian Center of Excellence in ICT research (EXCITE).

REFERENCES

[1] [n. d.]. Q-Rapids source code. https://github.com/q-rapids/qrapids-connect

[2] Tamer Mohamed Abdellatif, Luiz Fernando Capretz, and Danny Ho. 2015. Software Analytics to Software Practice: A Systematic Literature Review. In *Proceedings of the First International Workshop on BIG Data Software Engineering (BIGDSE '15)*. IEEE Press, Piscataway, NJ, USA, 30–36. http://dl.acm.org/citation.cfm?id=2819289.2819300

[3] Aytaj Aghabayli. 2019. *Software run time data: visualization and integration of development information - case study.* Master's thesis. University of Tartu. https://comserv.cs.ut.ee/ati{_}thesis/datasheet.php?id=66914{&}year=2019

[4] Richard Berntsson Svensson, Robert Feldt, and Richard Torkar. 2019. The Unfulfilled Potential of Data-Driven Decision Making in Agile Software Development. 69–85. https://doi.org/10.1007/978-3-030-19034-7_5

[5] Cesar Couto, Christofer Silva, Marco Tulio Valente, Roberto Bigonha, and Nicolas Anquetil. 2012. Uncovering Causal Relationships between Software Metrics and Bugs. In *2012 16th European Conference on Software Maintenance and Reengineering*. 223–232. https://doi.org/10.1109/CSMR.2012.31

[6] Hennie Huijgens, Davide Spadini, Dick Stevens, Niels Visser, and Arie van Deursen. 2018. Software Analytics in Continuous Delivery: A Case Study on Success Factors. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18)*. ACM, New York, NY, USA, 25:1–25:10. https://doi.org/10.1145/3239235.3240505

[7] ISO/IEC 25010:2011. 2011. Software engineering âĂŞ Product quality. https://www.iso.org/standard/35733.html

[8] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2018. Data Scientists in Software Teams: State of the Art and Challenges. *IEEE Transactions on Software Engineering* 44, 11 (2018), 1024–1038. https://doi.org/10.1109/TSE.2017.2754374

[9] Florian Lautenschlager and Marcus Ciolkowski. 2018. Making Runtime Data Useful for Incident Diagnosis: An Experience Report: 19th International Conference, PROFES 2018, Wolfsburg, Germany, November 28âĂŞ30, 2018, Proceedings. 422–430. https://doi.org/10.1007/978-3-030-03673-7_33

[10] Lidia López, Silverio Martínez-Fernández, Cristina Gómez, Michał Choraś, Rafał Kozik, Liliana Guzmán, Anna Maria Vollmer, Xavier Franch, and Andreas Jedlitschka. 2018. Q-Rapids Tool Prototype: Supporting Decision-Makers in Managing Quality in Rapid Software Development. Springer, Cham, 200–208. https://doi.org/10.1007/978-3-319-92901-9_17

[11] Silverio Martinez-Fernandez, Andreas Jedlitschka, Liliana Guzman, and Anna Maria Vollmer. 2018. A Quality Model for Actionable Analytics in Rapid Software Development. *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* 732253 (2018), 370–377. https://doi.org/10.1109/SEAA.2018.00067

[12] Silverio Martínez-Fernández, Petar Jovanovic, Xavier Franch, and Andreas Jedlitschka. 2018. *Towards Automated Data Integration in Software Analytics.* https://doi.org/10.1145/3242153.3242159

[13] Silverio Martínez-Fernández, Anna Maria Vollmer, Andreas Jedlitschka, Xavier Franch, Lidia López, Prabhat Ram, Pilar Rodríguez, Sanja Aaramaa, Alessandra Bagnato, Michał Choraś, and Jari Partanen. 2019. Continuously assessing and improving software quality with software analytics tools: a case study. *IEEE Access* (2019), 1. https://doi.org/10.1109/ACCESS.2019.2917403

[14] Tim Menzies and Martin Shepperd. 2019. âĂIJBad smellsâĂİ in software analytics papers. *Information and Software Technology* 112 (aug 2019), 35–47. https://doi.org/10.1016/J.INFSOF.2019.04.005

[15] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. 2006. *Mining metrics to predict component failures.* Vol. 2006. 452–461 pages. https://doi.org/10.1145/1134349

[16] Hilmer Rodrigues Neri and Guilherme Horta Travassos. 2018. Measuresoftgram: A Future Vision of Software Product Quality. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18)*. ACM, New York, NY, USA, 54:1—-54:4. https://doi.org/10.1145/3239235.3267438

[17] Jan Reimann. 2015. *Generic Quality-Aware Refactoring and Co-Refactoring in Heterogeneous Model Environments.* Ph.D. Dissertation.

[18] Colin Shearer. 2000. *The CRISP-DM model: the new blueprint for data mining.* Vol. 5. 13–22 pages.

[19] Uthayasankar Sivarajah, Muhammad Mustafa Kamal, Zahir Irani, and Vishanth Weerakkody. 2017. Critical analysis of Big Data challenges and analytical methods. *Journal of Business Research* 70 (jan 2017), 263–286. https://doi.org/10.1016/J.JBUSRES.2016.08.001

[20] Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, and Tao Xie. 2013. *Software Analytics in Practice.* Vol. 30. 30–37 pages. https://doi.org/10.1109/MS.2013.94