Understanding Microservice Runtime Monitoring Data for Anomaly Detection with Structural Equation Modeling[⋆]

Monika Steidl[1][0000−0002−3410−7637], Michael Leitner[2][0009−0001−8876−0033], Pirmin Urbanke[3][0009−0005−1868−3078], Marko Gattringer[2][0000−0003−1659−3624], Michael Felderer[1,4,5][0000−0003−3818−4442], and Sashko Ristov[1][0000−0003−1996−0098]

[1] University of Innsbruck, Austria
{Monika.Steidl, Sashko.Ristov}@uibk.ac.at
[2] Gepardec IT Services GmbH, Austria
{Michael.Leitner, Marko.Gattringer}@gepardec.com
[3] Software Competence Center Hagenberg GmbH, Austria
{Pirmin.Urbanke}@scch.at
[4] German Aerospace Center (DLR), Institute of Software Technology, Germany
{Michael.Felderer}@dlr.de
[5] University of Cologne, Cologne, Germany

Abstract. Microservices reliability is critical, but runtime anomalies are increasingly common due to system complexity. Rule-based and AI-based anomaly detection methods assist practitioners in analyzing runtime monitoring data (logs, traces, metrics) to identify anomalies. However, these methods rely on high-quality datasets and deep domain knowledge to deliver accurate results. Thus, a significant challenge lies in the lack of consensus on which runtime monitoring parameters effectively represent the system and microservices, reliably indicate anomalies, or distinguish deviations that genuinely signal anomalies. A thorough understanding of the dataset, key monitoring parameters, and microservice dependencies is crucial to minimize bias and false positives, ultimately improving the effectiveness of anomaly detection methods.

Thus, we investigate whether structural equation modeling can describe the system's or microservices' behavior via indicators extracted from runtime monitoring data and identify their causal relationships. We used EvoMaster to simulate user behavior in TrainTicket and extract runtime monitoring data to test our model. Our results show that the identified indicators effectively describe microservices' behavior, but network indicators alone are insufficient for describing the whole system's behavior. The model can also identify microservices that significantly influence the whole system's behavior.

Keywords: anomaly detection during runtime · monitoring data · logs · traces · metrics · microservice · structural equation model · PLS-SEM

[⋆] This work was supported by the Austrian Research Promotion Agency (FFG) in the frame of the project ConTest [888127] and the MATISSE project (KDT Joint Undertaking) under the grant agreement No. 101140216.

1 Introduction

Frequent software changes are crucial for fixing bugs, adding features, and improving performance. These changes in microservices must be tested to preserve reliability and functionality during runtime. However, microservice systems' complexity, diverse languages, and independent components make quality assurance challenging, as comprehensive test suites often fail to cover all runtime deviations [15,11,12].

Thus, anomaly detection techniques, such as rule-based or AI-based methods for microservices, are increasingly researched to find unexpected deviations during runtime, called anomalies [11,10]. These methods use runtime monitoring data, such as logs, traces, and metrics, to analyze the system's and microservice's behavior [11]. Their effectiveness and accuracy in reducing false positives depend on the dataset quality and understanding of the respective runtime monitoring parameters. For instance, increased user requests or Kubernetes scaling can affect response times without indicating anomalies if the relationship of specific parameters remains stable [11,10]. Several authors emphasize the need to retain key monitoring parameters for anomaly detection while removing irrelevant ones, making it challenging to identify the right combination that captures diverse anomalies and their impact on the system and microservice [9,14,5,17].

Therefore, this paper aims to apply statistical methods to model the complex causal relationships between multiple microservices and their influence on the whole system's behavior. Furthermore, we want to identify essential parameters, also called indicators, that depict the behavior of the system and microservices. Therefore, this paper provides preliminary results if this research goal can be achieved via specifying a Structural Equation Modeling (SEM).

1.1 Structural Equation Modeling (SEM)

We opted for **SEM** because it is a robust statistical method for analyzing complex, multivariate data. It is particularly useful for exploring relationships and patterns among constructs, which are not directly measurable but can be estimated via multiple indicators [2,3,8]. For instance, the overall system's behavior cannot be observed via a single parameter. Still, it can be estimated using a combination of runtime monitoring data, such as resource utilization, response time, and error rate. SEM allows us to model the interconnection through paths (or hypotheses) of constructs and their multiple indicators, and the complex relationships are not limited to a single type [2,3,4].

More specifically, we opted for Partial Least Square-Structural Equation Modeling (**PLS-SEM**) because compared to other SEM approaches, like Covariance based-Structural Equation Modeling (CB-SEM), Partial Least Square-Structural Equation Modeling (PLS-SEM) is a causal-predictive approach to SEM and is particularly suited for exploratory research and theory development where initially it is unclear how various constructs relate. PLS-SEM is more effective at identifying significant relationships and allows for greater flexibility in introducing or omitting constructs [4,8]. PLS-SEM allows us to simultaneously estimate complex models with many constructs and indicators in one coherent model and provides causal explanations [2,3,4]. Also, the data does not need normal distribution [2,3].

2 Research Approach

We follow the systematic approach by Russo and Stol [8] for a PLS-SEM analysis and review in software engineering, consisting of 2.1) specification of structural model, 2.2) specification of measurement model, 2.3) data collection, 2.4) assessment of measurement model, and 2.5) assessment of structural model. In the following subsection, we will concisely state the applied methodology and preliminary results and discuss limitations that should be resolved by future work. Please refer to our replication package [13] for insights, including details about the specification of the models and their assessment and analysis.

2.1 Specification of Structural Model

Method: This step identifies relevant constructs and their relationships by establishing hypotheses. These should be based on prior theory or empirical observations [8]. In previous work [11], we conducted a systematic literature review of 92 papers, including 36 with industry-related use cases and 15 interviews on runtime anomaly detection. For this paper, we used them to extract information relevant to the structural and measurement models inductively.
Results: It is crucial to look at microservices individually, as not all anomalies are apparent at the system-wide level [14,6]. Half of the interview participants stated that understanding each microservice's impact helps identify critical microservices and their role in anomalies. Therefore, we treat the entire system and its microservices as constructs and hypothesize that a microservice significantly affects the system's behavior ($H1_a$).

An anomaly in one observed microservice might influence other microservices [14,6]. Anomaly propagation can be unclear due to the subtle and non-obvious ways anomalies spread [14]. Therefore, examining various anomalies and their effects on microservices helps understand the system's behavior. We hypothesize that the anomalous microservice impacts other connected microservices ($H1_b$).

2.2 Specification of Measurement Model

Method: In this step, we extract multiple indicators to measure the previously defined constructs (system's and microservices' behavior), which cannot be measured directly [8]. These indicators are based on key runtime monitoring parameters from the literature review and interviews on anomaly detection summarized in our previous work [11]. The identified parameters cover logs, traces, and metrics. Logs are semi-structured messages with timestamps, verbosity levels (e.g., INFO, WARN, DEBUG, ERROR), and natural language descriptions. Traces show the request's execution path through microservices, with spans as trace segments containing metadata like start and end times. Metrics are numeric time series data reflecting system or microservice performance [11].

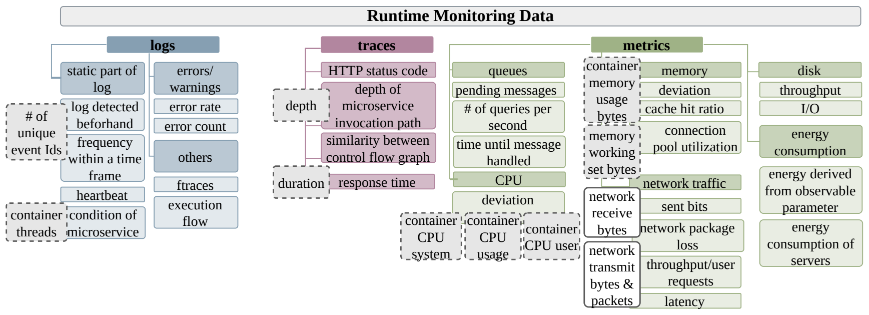

Fig. 1. Runtime monitoring data [11] with whole system indicators (solid & white box) and microservice indicators (dashed & grey box)

Results: Figure 1 shows the indicators extracted from Logger (logs), Jaeger (traces), and Prometheus (metrics), and how they are mapped to the previously discovered parameters. We chose multiple indicators to reduce standard error and iteratively refine the measurement model by discarding inadequate ones [8]. Our replication package provides further description of the extracted indicators [13]. We opted for reflective indicators because a change in the construct (whole system's or microservice's behavior) causes a change in its indicators. Furthermore, the indicators are interchangeable (e.g., several indicators describe CPU usage), where dropping an indicator does not alter the meaning of the construct [8,2].

Focusing on relevant indicators and removing irrelevant ones enhance dataset quality and anomaly detection methods [5,11,10]. For instance, it is unclear if high granular or specific indicators help identify anomalies more reliably [16]. In addition, indicator usefulness varies by anomaly type, often requiring domain knowledge for selection. Some anomalies manifest through subtle indicator shifts, and single monitoring data types may be insufficient for detection [5,11]. Statistical assessment of indicator impact on the system or microservice can guide their selection process [16]. In assessing the measurement model, we aim to determine if the proposed indicators impact the system or microservice ($H2_a$).

Furthermore, understanding the relationship of indicators is required [16,11]. By identifying correlations, we can pinpoint specific indicators that share similarities, allowing us to detect changes in their relationship that may signal an anomaly [8,5,16]. For example, indicators such as CPU user usage, CPU system usage, and CPU wait, describe and influence CPU performance [5,16]. Thus, we determine if the measurement model identifies a correlation between its indicators of the runtime monitoring data ($H2_b$).

Figure 2 illustrates a conceptual version of the described structural and measurement model. The structural model consists of the constructs (whole system's and microservices' behavior) and their paths (hypotheses). The measurement model includes each construct's currently used measurable indicators (extracted from runtime monitoring data).

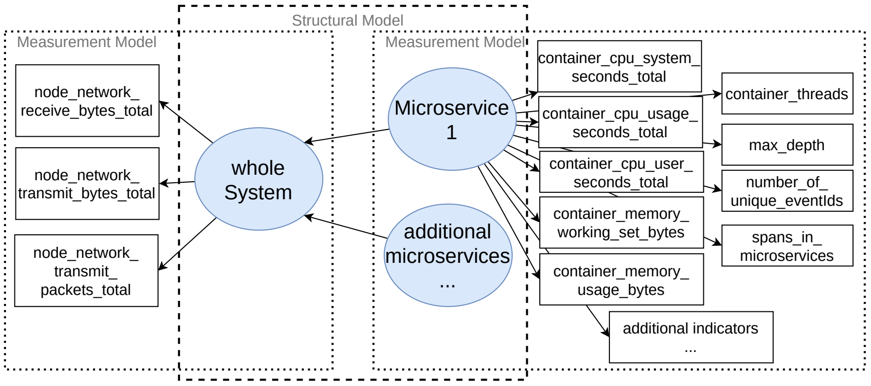
Structural Model
Measurement Model
Measurement Model
node_network_ receive_bytes_total
node_network_ transmit_bytes_total
node_network_ transmit_ packets_total
whole System
Microservice 1
additional microservices ...
container_cpu_system_ seconds_total
container_cpu_usage_ seconds_total
container_cpu_user_ seconds_total
container_memory_ working_set_bytes
container_memory_ usage_bytes
additional indicators ...
container_threads
max_depth
number_of_ unique_eventIds
spans_in_ microservices

Fig. 2. PLS-SEM with constructs (circles) and reflective indicators (rectangles)

2.3 Data Collection

Method: We extracted runtime monitoring data from TrainTicket[6], a widely studied microservice benchmark application for anomaly detection [1,17]. To simulate the load, we used EvoMaster, an open-source AI-driven tool that automatically creates and executes system-level tests [7]. We ran TrainTicket without anomalies as a baseline to assess the feasibility of PLS-SEM for runtime monitoring data and identify anticipated relationships between constructs. We created ten runtime monitoring datasets with injected anomalies for future work (see [13]).
Results: The collected runtime monitoring data consists of metric data with ratio measurements required for the indicators in PLS-SEM [8,4,2]. However, we had to apply data transformation to merge logs, traces, and metrics where one sample is represented in one row. Please refer to the GitHub repository for the code that extracts these system and microservice indicators[8]. This code scans the input, prints statistics about the input data, and clears the output folder. After that, it parses logs (via logparser [18][9]), Jaeger traces, and Prometheus metrics associated with the whole system or specific microservices and concatenates them based on their timestamps. Prometheus collects metrics at regular intervals, while Jaeger and the Logger record traces and logs as they occur. As a result, logs and traces may not align perfectly with Prometheus timestamps, requiring rounding or an asof-join. For better visualization, we generate a graph of

[6] `https://github.com/FudanSELab/train-ticket`, accessed 13.05.2024
[7] `https://github.com/WebFuzzing/EvoMaster`, accessed 29.07.2024
[8] `https://github.com/moniSt13/ConTest-Parsing.git`, accessed 08.08.2024
[9] `https://github.com/logpai/logparser?tab=readme-ov-file`, accessed 01.08.2024

the microservices via HTML-rendered graphs with neo4j and vis.js. Finally, we transform the data by transposing each trace and its respective log and metric information into a single line (sample), with each span appended to this one-line trace. We were able to extract over 25000 samples, which minimizes the risk of a wrong representation or sample size issue and increases precision, such as consistency [8,2].

2.4 Assessment of (Reflective) Measurement Model

Method: We used SmartPLS[7] to calculate the consistent PLS-SEM algorithm for the monitoring data of the anomaly-free runtime monitoring dataset to validate and improve the established model, and create a baseline regarding constructs and indicators used for future research. For the model's setup, we had to follow three rules: indicators must have non-zero variance and no perfect collinearity for the same construct, and the sample should be ten times the number of structural paths [8]. Thus, we need a minimum of 400 samples because of the potential involvement of 40 TrainTicket's microservices. Nevertheless, we included 25000 samples for our assessment.

We evaluate the reflective measurement model based on Russo and Stol's [8] evaluation criteria, such as internal consistency reliability, convergent validity, and discriminant validity.

Results: The `internal consistency reliability` assesses whether the construct's indicators consistently measure the same underlying concept, ensuring valid interpretations. Therefore, we use Cronbach's alpha to evaluate the consistency of the results across indicators. This is satisfactory and significant (p-value: 0.000) for all microservices that fulfilled the rules to be included in the model, except *ts-cancel-service*. Nonetheless, its value is between 0.6 and 0.7 (0.690), considered acceptable for exploratory research [4,8]. Cronbach's alpha of the whole system equals 1, indicating that its indicators seem redundant and measure the same phenomenon.

For `convergent validity`, we examined the outer loadings of the indicators. We found that most outer loadings for metric-based indicators fulfill the required 0.7, whereas indicators calculated based on logs (number of unique static parts of logs) or traces (trace duration or depth of microservices) do not perform well in explaining the construct. Russo et al. [8] noted that constructs under development often have loadings below 0.7 and may be considered for removal if it improves the Average variance extracted (AVE). Thus, removing the trace duration increased the AVE to over 0.5, indicating that this indicator reduced the construct's overall explanatory power. However, the microservice *ts-cancel-service* still has a AVE of 0.364, indicating that its indicators share only 36% of their variance and require further refinement.

For the `discriminant validity`, we identified that the measured microservices and the whole system are not conceptually similar (conservative cut-off should be below 0.85 for the heterotrait-monotrait ratio (HTMT) and confidence interval via bootstrapping does not contain 1.0), indicating that the identified constructs are unique in relation to each other where the *ts-order-service* and

system-wide seem to be most similar with 0.831.
Discussion, limitations, and future research: Based on the results, we can discuss H2$_a$ regarding the impact of proposed indicators on the system's or microservices' behavior. While construct results for the measurement model seem promising, log-based and trace-based indicators need further improvement due to their low outer loadings. This may be because most traces in the used dataset contain only one span, making the depth of the microservice nearly always 0. This can artificially reduce the outer loading due to low variability. We used the unique static parts of logs to count unique eventIds before a timestamp, noting that this indicator always increases. Thus, implementing a sliding time window could better capture actual behavior for future work. Furthermore, anomalies might manifest in a continuous change of the indicator, making it necessary to consider indicators over time.

To address H2$_b$, we examined `residuals` to see how indicators correlate with alternative indicators [2,8]. For this dataset, indicators for the same construct often correlate, with memory-related indicators correlating negatively with CPU-related ones. Network indicators also exhibit a strong correlation, indicating that they may not fully represent the entire system's behavior since they share some of their variance. Thus, in the future, we will introduce additional indicators to better explain the constructs. For instance, we could extract error and warning counts, disk-related indicators (like throughput or I/O), or energy consumption to extend potential indicators. After establishing a more comprehensive set of indicators, we will assess this measurement model with anomaly-injected datasets, as indicators react differently to various anomalies. Thus, the interpretation may vary [12,11].

2.5 Assessment of Structural Model

Method: We evaluated the structural model, based on Russo and Stol's [8] evaluation criteria, to gain insights about the predictive power and significance of the relationship between constructs [8].
Results: The `collinearity among constructs` represents the relationship between constructs. We achieved approximately a Variance inflation factor (VIF) value of 1 for all constructs in the structural model, indicating that our chosen constructs do not represent similar concepts and, thus, are not biased. The model's `explanatory power` for the system-wide construct results in 0.712, indicating that 71.2% of the variation in this construct is explained by the microservices pointing to the whole system. Regarding the `path significance`, we identified that one microservice (*ts-order-service*) strongly and statistically significantly relates to the whole system with a path coefficient of over 0.8 and a p-value of 0.000 calculated via consistent bootstrapping. Furthermore, this is the only microservice that has a large effect size (f^2) on the system-wide construct.
Discussion, limitations, and future research: This study demonstrates how SEM provides insights into the causal relationship and influence between the whole system's and microservices' behavior measured via runtime monitoring

data. We can answer $H1_a$ that the construct indicating the behavior of the microservice *ts-order-service* statistically significantly influences the whole system's behavior. Other microservices do not influence the whole system's behavior.

However, we had to exclude some of TrainTicket's microservices as constructs due to zero variance in their indicators in the used dataset. This occurs because EvoMaster executes system tests for one microservice at a time, resulting in some microservices not receiving load for the two-hour test period. However, microservices that are not invoked do not influence the system's behavior or other microservices as they are not required for the observed operations. For future work, we want to use other runtime monitoring datasets to have more complex hypothesized structural relationships between the microservices. Furthermore, since we have assessed the model using a dataset without injected anomalies to establish a baseline for the structural and measurement model, future work will analyze runtime monitoring data with various injected anomalies. This will address $H1_b$ regarding the impact of the anomalous microservice on other connected microservices.

3 Conclusion

Frequent microservice updates are vital for fixing bugs and improving performance, but they must maintain system and microservice reliability. Traditional test suites often overlook runtime anomalies, making advanced anomaly detection techniques necessary. These methods use runtime monitoring data (logs, traces, metrics) to understand the system's and microservices' behavior. The main challenge is selecting relevant indicators to accurately measure these behaviors and identify causal relationships between the system and microservices.

Thus, this paper specifies Partial Least Square-Structural Equation Modeling (PLS-SEM) to model these relationships and improve the understanding of how runtime monitoring data assesses system and microservice behavior.

We identified that metric-based indicators describe the microservice behavior well, where CPU and memory have a statistically significant negative correlation for our generated TrainTicket dataset. Future research should refine indicators based on logs and traces as well as indicators describing the system's behavior. Furthermore, the established PLS-SEM can identify microservices with a strong relationship and influence on the system's behavior. Thus, we plan to utilize additional runtime monitoring datasets to explore more complex hypothesized structural relationships between anomalous microservices.

References

1. Fischer, S., Urbanke, P., Ramler, R., Steidl, M., Felderer, M.: An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study. Proceedings - 2024 IEEE/ACM International Conference on Automation of Software Test, AST 2024 pp. 182–192 (apr 2024). https://doi.org/10.1145/3644032.3644445, `https://dl.acm.org/doi/10.1145/3644032.3644445`
2. Hair, J., Hult, T., Ringle, C., Sarstedt, M.: A primer on partial least squares structural equation modeling (PLS-SEM)-Third Edition (2021)

3. Hair, J.F.: Multivariate data analysis: a global perspective. Pearson, 7. ed., global ed edn. (2010)
4. Hair, J.F., Risher, J.J., Sarstedt, M., Ringle, C.M.: When to use and how to report the results of PLS-SEM (2018). https://doi.org/10.1108/EBR-11-2018-0203, www.smartpls.com
5. Lee, C., Yang, T., Chen, Z., Su, Y., Yang, Y., Lyu, M.R.: Heterogeneous Anomaly Detection for Software Systems via Semi-supervised Cross-modal Attention. Proceedings - International Conference on Software Engineering pp. 1724–1736 (feb 2023). https://doi.org/10.1109/ICSE48619.2023.00148, https://arxiv.org/abs/2302.06914v1
6. Nedelkoski, S., Cardoso, J., Kao, O.: Anomaly detection from system tracing data using multimodal deep learning. IEEE International Conference on Cloud Computing, CLOUD **2019-July**, 179–186 (jul 2019). https://doi.org/10.1109/CLOUD.2019.00038
7. Ringle, C.M., Wende, S., Becker, J.M.: Smartpls 4 (2024), https://www.smartpls.com/
8. Russo, D., Stol, K.J.: PLS-SEM for software engineering research: An introduction and survey. ACM Computing Surveys **54**(4) (2021). https://doi.org/10.1145/3447580, https://doi.org/10.1145/3447580
9. Shan, H., Zhang, Y., Chen, Y., Xiao, X., Liu, H., He, X., Li, M., Ding, W.: ϵ-Diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019 pp. 3215–3222 (may 2019). https://doi.org/10.1145/3308558.3313653, https://dl.acm.org/doi/10.1145/3308558.3313653
10. Soldani, J., Brogi, A.: Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. ACM Computing Surveys **55**(3), 39 (feb 2022). https://doi.org/10.1145/3501297, https://dl.acm.org/doi/abs/10.1145/3501297
11. Steidl, M., Dornauer, B., Felderer, M., Ramler, R., Racasan, M.C., Gattringer, M.: How industry tackles anomalies during runtime: Approaches and key monitoring parameters (8 2024), https://doi.org/10.48550/arXiv.2408.07816
12. Steidl, M., Gattringer, M., Felderer, M., Ramler, R., Shahriari, M.: Requirements for Anomaly Detection Techniques for Microservices. Lecture Notes in Computer Science **13709 LNCS**, 37–52 (2022). https://doi.org/10.1007/978-3-031-21388-5_3
13. Steidl, M., Leitner, M., Urbanke, P.: Replication Package: Anomaly detection via runtime monitoring data for structural equation modeling (Aug 2024). https://doi.org/10.5281/zenodo.13324135, https://doi.org/10.5281/zenodo.13324135
14. Wang, P., Xu, J., Ma, M., Lin, W., Pan, D., Wang, Y., Chen, P.: CloudRanger: Root cause identification for cloud native systems. Proceedings - 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018 pp. 492–502 (jul 2018). https://doi.org/10.1109/CCGRID.2018.00076
15. Waseem, M., Liang, P., Shahin, M., Di Salle, A., Márquez, G.: Design, monitoring, and testing of microservices systems: The practitioners' perspective. Journal of Systems and Software **182**, 111061 (dec 2021). https://doi.org/10.1016/J.JSS.2021.111061
16. Xie, Z., Xu, H., Chen, W., Li, W., Jiang, H., Su, L., Wang, H., Pei, D.: Unsupervised Anomaly Detection on Microservice Traces through Graph VAE. ACM Web Conference 2023 - Proceedings of the World Wide Web Conference, WWW 2023 pp. 2874–2884 (apr 2023). https://doi.org/10.1145/3543507.3583215, https://dl.acm.org/doi/10.1145/3543507.3583215
17. Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W., Ding, D.: Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. IEEE Transactions on Software Engineering **47**(2), 243–260 (feb 2021). https://doi.org/10.1109/TSE.2018.2887384
18. Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and Benchmarks for Automated Log Parsing. Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019 pp. 121–130 (may 2019). https://doi.org/10.1109/ICSE-SEIP.2019.00021