Commit-Based Continuous Integration of Performance Models

Martin Armbruster (martin.armbruster@kit.edu)
KASTEL - Institute of Information Security and Dependability, Karlsruhe Institute of Technology

1 Introduction

Architecture-level performance models (aPM) such as the *Palladio Component Model* (PCM) [5] can be used for, e.g., performance predictions to explore design alternatives and combines the aspects of architecture and performance models. An up-to-date architecture model can support the software maintenance by reducing the architectural degradation or guide the software evolution. At the same time, performance models allow the investigation of the software performance without the need to implement or change the system. However, keeping them up-to-date requires manual effort which hinders their adoption. Especially in the agile software development which is characterized by incremental and iterative development cycles, no or short design phases prevent manual modeling activities.

2 Foundations

PCM The PCM is a metamodel and framework for describing and analyzing component-based software architectures [5]. In *Repository* models, components and their interfaces with offered and required services are defined. Furthermore, a component includes *Service Effect Specifications* (SEFF) which abstractly model the behavior of services in terms of actions including, e.g., calls to other services. Performance model parameters (PMP) such as a resource demand can be attached to actions which are then employed during PCM simulations to predict the performance.

There are approaches such as the *Reconstructive Integration Strategy* [3] which extract a PCM from source code. However, they do not support incremental updates and would extract a complete PCM for every source code change. In addition, the *Coevolution approach* [3] is able to incrementally update a PCM based on code changes. Nevertheless, it assumes the availability of editors that record the changes during the development.

Continuous Integration of Performance Models (CIPM) Addressing the aforementioned issues of keeping aPMs up-to-date with automatized activities, the CIPM approach proposes a Continuous Integration (CI) pipeline [4]. As first step in the pipeline, a commit-based integration strategy extracts changes from a commit and incrementally updates an aPM. At the same time, relations between code elements and their corresponding elements in the aPM are established. To estimate the PMPs, the source code is adaptively instrumented, i.e., only the parts of the source code which have changed are instrumented, and monitored. The taken measurements are used to calibrate the aPM. As a consequence of the adaptive instrumentation and monitoring, the monitoring and calibration overhead can be reduced. Moreover, unaffected PMPs are not estimated again because they have not change.

The CIPM approach's realization supports Java as source code and targets the PCM as aPM. Parts of the pipeline were prototypically implemented and evaluated in previous work. In particular, a delta-based extraction of changes from a commit was implemented and evaluated with an artificial Git history. In a second and third work, the adaptive instrumentation was implemented twice which inserts the text-based instrumentation statements into a copy of the source code. Both implementations derive the insertion locations from an instrumentation model which stores the SEFF actions to instrument as instrumentation points. Moreover, a calibration pipeline was implemented in the third work and evaluated with three case studies.

3 Approach

This master thesis [1] presents an approach building upon the previous work with these two main goals: (1) closing the gaps by completing the pipeline for the aPM extraction and instrumentation, and (2) evaluating the pipeline with a real Git history. As a result, in the approach, the Java source code in the state of a new commit is parsed into a code model. By a state-based comparison with the code model of the previous commit, a delta-based change sequence is obtained which describes how the code model of the previous commit can be transformed into a code model conforming to the state of the new commit.

Afterwards, the changes are utilized to incrementally update the PCM, i.e., only the PCM elements affected by the source code changes are updated while the other PCM elements remain unchanged. The PCM update is defined by technology- and project-specific rules which specify how a change in the Java code model is reflected as changes in the PCM. Thus, the rules allow to obtain an architecture model with respect to the applied technologies and to project-

specific conventions in the code.

In this thesis, the implemented rules are targeted at extracting the architecture of Microservice-based applications where a Microservice is modeled as a component. Thus, in a first step, Microservices are found by the organization of the source code in modules of the build system which are recognized by build files. In addition, the rules cover *Jakarta Servlets* and *Jakarta RESTful Web Services* (JAX-RS) to identify the Microservices' REST APIs in a second step which are represented in PCM interfaces. As an example, if a module of the build system is removed, the corresponding component in the PCM is also deleted. On the other hand, if a class with a corresponding PCM interface is renamed, the PCM interface is renamed, too.

In case of changes in a method with a corresponding SEFF, the SEFF is updated by adding or removing actions and revising the relations to the statements in the code. Those changes in the actions are reflected one-to-one in the instrumentation model by adding or removing instrumentation points.

As last step in the thesis' approach, the source code is adaptively instrumented. During the instrumentation, a copy of the code model is extended with the instrumentation statements whose insertion locations are based on the instrumentation model and the relations between the SEFF actions and the corresponding source code statements. The instrumented code model is printed as source code which can be compiled and executed to take the required measurements for the calibration.

4 Evaluation

The evaluation of the presented approach aims at finding out how accurate the generated models are, how accurate the code is instrumented, and how large the reduction of the monitoring overhead is compared to a full instrumentation of the source code. Therefore, the evaluation is executed with the *TeaStore*[1] which is a web-based store for tea and related products [2]. Its commits between version 1.1 and 1.3.1 are divided into four intervals and propagated within the intervals to simulate the development. All intervals span 292 commits of which 58 commits include changes in 144 Java files with overall 10030 added and 8270 removed lines.

To assess the accuracy of the generated models, the updated Java code model is compared to a newly parsed model corresponding to the specific commit. The automatically updated PCM is compared to a manually updated one. In order to obtain a manual PCM, the changes of a commit are manually analyzed and applied on the previous PCM according to the rules defined in the approach.

[1] `https://github.com/DescartesResearch/TeaStore`

For the instrumentation, several indicators are considered. They include a counting of the statements before and after the instrumentation, a compilation of the instrumented code, and a partly manual inspection of the instrumentation statements. At last, the reduced monitoring overhead is calculated as the ratio of the actual instrumented instrumentation points to all potential instrumentation points.

The evaluation results indicate that the generated models are accurately updated and the source code is accurately instrumented. The reduction of the monitoring overhead ranges between 60.5% and 97.6%. As a consequence, the thesis' approach leads to an improved usability of the CIPM approach and a reduced effort through automatization.

References

[1] Martin Armbruster. "Commit-Based Continuous Integration of Performance Models". Master Thesis. Karlsruher Institut für Technologie, Sept. 14, 2021. DOI: 10.5445/IR/1000154588.

[2] Jóakim von Kistowski et al. "TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research". In: *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. MASCOTS '18. Milwaukee, WI, USA, Sept. 2018.

[3] Michael Langhammer. "Automated Coevolution of Source Code and Software Architecture Models". PhD thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT), 2017. 259 pp. DOI: 10.5445/IR/1000069366. URL: http://nbn-resolving.org/urn:nbn:de:swb:90-693666.

[4] Manar Mazkatli et al. "Incremental Calibration of Architectural Performance Models with Parametric Dependencies". In: *IEEE International Conference on Software Architecture (ICSA 2020)*. 2020. DOI: 10.1109/ICSA47634.2020.00011.

[5] Ralf H. Reussner et al., eds. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016. 408 pp. ISBN: 978-0-262-03476-0.