# Model-Based Performance Evaluations in Continuous Delivery Pipelines

Markus Dlugi, Andreas Brunnert
fortiss GmbH
Guerickestr. 25
80805 Munich, Germany
{dlugi|brunnert}@fortiss.org

Helmut Krcmar
Technische Universität München
Boltzmannstr. 3
85748 Garching, Germany
krcmar@in.tum.de

## ABSTRACT

In order to increase the frequency of software releases and to improve their quality, continuous integration (CI) systems became widely used in recent years. Unfortunately, it is not easy to evaluate the performance of a software release in such systems. One of the main reasons for this difficulty is often the lack of a test environment that is comparable to a production system. Performance models can help in this scenario by eliminating the need for a production-sized environment. Building upon these capabilities of performance models, we have introduced a model-based performance change detection process for continuous delivery pipelines in a previous work. This work presents an implementation of the process as plug-in for the CI system Jenkins.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: measurement techniques, modeling techniques

## General Terms

Measurement, Performance

## Keywords

Performance Evaluation, Performance Change Detection, Palladio Component Model, Continuous Delivery

## 1. INTRODUCTION

The modern software engineering landscape has been reshaped significantly in recent years by new paradigms such as agile software development. This led to a shift from the traditional model of releasing new features in few, major versions to rapid release cycles pushing new features and bug fixes to the user in increasingly shorter intervals. In order to support such short release cycles and still ensure adherence to all requirements during the entire development process, new concepts like continuous integration (CI), continuous delivery (CD) or continuous deployment have emerged. However, while functional requirements are rigorously evaluated in these systems to avoid breaking the application, non-functional requirements like performance (i.e., response time, utilization and throughput) are only rarely examined. In the case of software performance, one frequent reason for this is the lack of an adequate performance test environment [3]. Performance models such as the Palladio Component Model (PCM) [1] bear the potential to help in this scenario. By automatically generating a performance model and running a simulation, a prediction of an enterprise application's (EA) performance can be made which is independent of the environment used to generate the model [4]. By integrating such a model-based performance evaluation process into a CD deployment pipeline, changes in an EA's performance can be detected for every build of the EA. This capability reduces development costs and risks as performance regressions introduced by new features or bugs can be detected immediately, thus avoiding expensive fixes once the EA is in production.

This work presents the integration of an existing performance change detection process [2] into Jenkins[1] as an exemplary CI system. It builds upon the previous work that demonstrated the feasibility of such an approach and presents an integrated tool for the whole process.



*QUDOS'15*, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3817-2/15/09...
http://dx.doi.org/10.1145/2804371.2804376

## 2. CONTINUOUS INTEGRATION, DELIVERY & DEPLOYMENT

CI describes the practice of ensuring a usable application during all stages of the development process [5]. This means that every time a developer commits their changes, they are instantly integrated with the rest of the application and acceptance tests are performed to guarantee the soundness and stability of the system.

CD takes the idea of CI one step further and demands that the application not only be usable, but also in a deployable state for every successful release candidate [5]. A key concept of CD is a so-called deployment pipeline. It consists of multiple steps such as acceptance testing, capacity testing or packaging to produce a deployable artifact in the end.

Finally, continuous deployment is the practice of actually deploying every application version that has passed the necessary tests to production [5].

## 3. PERFORMANCE CHANGE DETECTION

The tool for performance change detection is realized as a Jenkins plug-in. The process necessary for providing the

[1] http://jenkins-ci.org

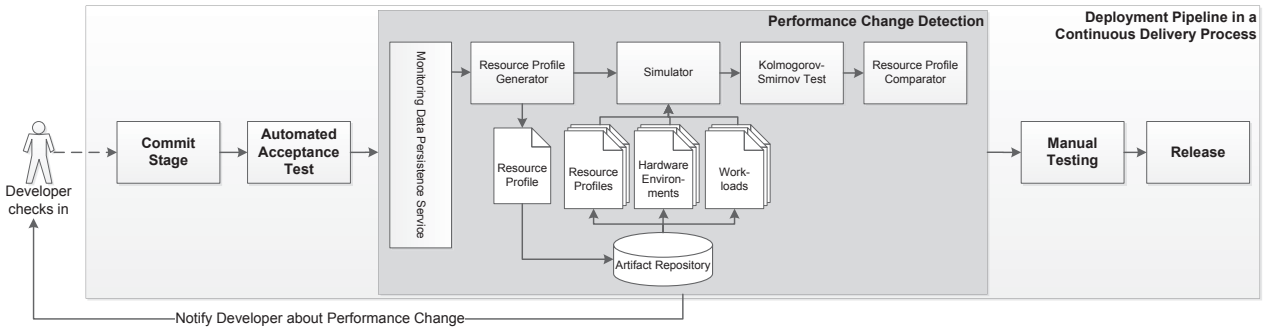


**Figure 1: Performance Change Detection Process (adapted from [2, 5])**

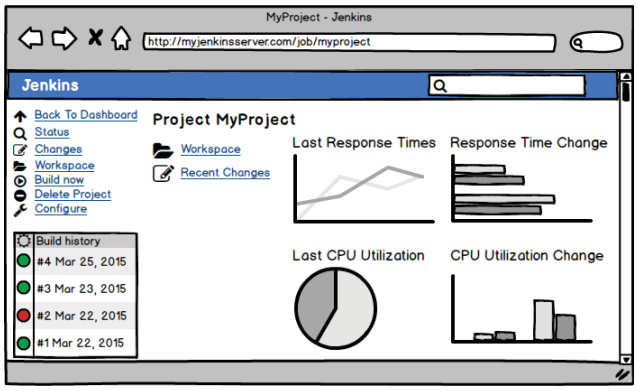


**Figure 2: Mockup of the Web UI**

change detection functionality is illustrated in figure 1.

Once the build process and the automated acceptance tests are completed, the first process step is to generate specifically formed performance models called resource profiles. A resource profile describes the performance-relevant aspects of an EA by depicting the control flow and resource demand of single transactions on the level of single component operations. Users can specify the workload for a resource profile on the level of single transactions and the deployment topology using deployment units. The data necessary for the generation is gathered using application performance monitoring (APM) tools. In order to support different monitoring technologies, the Jenkins plug-in provides a configuration to define which APM solution is used to collect the necessary data. Out of the box, the plug-in provides support for dynatrace[2] as well as a custom monitoring solution. However, it can be extended to additional APM tools.

The resulting resource profile is stored in an artifact repository [2]. The artifact repository is responsible for holding performance model information about each of the EA's revisions and is realized as EMFStore server. Besides the various resource profile versions, it also contains the hardware environments and workloads designed for the performance evaluation; these artifacts need to be manually created prior to the change detection. The Simulator retrieves the generated resource profile as well as the predefined hardware environments and workloads and executes a simulation for each combination. This step is repeated for all resource profile versions to which the latest version should be compared.

After all simulations have finished, the resulting performance metrics of all examined application versions are compared using two-sample Kolmogorov-Smirnov tests to determine whether the samples differ significantly. If a performance regression is found during one of the tests, the Resource Profile Comparator tries to find the reason for it by analyzing the control flow of the business transactions containing a regression, comparing the involved resource profiles using EMFCompare and computing their difference. Finally, a report is generated containing visualizations of the examined performance metrics as well as a list of the methods that are likely to be responsible for the observed regression. A mockup of the information available to the developer is depicted in figure 2.

[2]http://www.dynatrace.com

## 4. CONCLUSION AND FUTURE WORK

This work presented the integration of a model-based performance evaluation process into an exemplary CI system. While the basic process has been implemented and is working as described in section 3, there are still many aspects of the prototype that can be refined and improved. One major task is the development of a configuration interface to enable the developer to easily manipulate the various process parameters as well as the hardware environment and workload models.

## 5. REFERENCES

[1] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[2] A. Brunnert and H. Krcmar. Detecting performance change in enterprise application versions using resource profiles. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '14, pages 165–172, ICST, Brussels, Belgium, 2014.

[3] A. Brunnert, C. Vögele, A. Danciu, M. Pfaff, M. Mayer, and H. Krcmar. Performance management work. *Business & Information Systems Engineering*, 6(3):177–179, 2014.

[4] A. Brunnert, K. Wischer, and H. Krcmar. Using architecture-level performance models as resource profiles for enterprise applications. In *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '14, pages 53–62, New York, NY, USA, 2014. ACM.

[5] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation.* Addison-Wesley Professional, 1st edition, 2010.