Continuous Integration of Performance Model

Manar Mazkatli
Institute for Program Structures and Data Organization
(IPD) - Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
manar.mazkatli@kit.edu

Anne Koziolek
Institute for Program Structures and Data Organization
(IPD) - Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
koziolek@kit.edu

ABSTRACT

Applying model-based performance prediction requires that an up-to-date Performance Model (PM) is available throughout the development process. Creating such a model manually is an expensive process that is unsuitable for agile software development aiming to produce rapid releases in short cycles. Existing approaches automate the extraction of a PM based on reverse engineering and/or measurements techniques. However, these approaches require to monitor and analyse the whole application. Thus, they are too costly to be applied frequently, up to after each code change. Moreover, keeping potential manual changes of the PM is another challenge as long the PM is regenerated from scratch every time.

To address these problems, this paper envisions an approach for efficient continuous integration of a parametrised performance model in an agile development process. Our work will combine static code analysis with adaptive, automatic, dynamic analysis covering updated parts of code to update the PM with parameters, like resource demands and branching probabilities. The benefit of our approach will be to automatically keep the PM up-to-date throughout the development process which enables the proactive identification of upcoming performance problems and provides a foundation for evaluating design alternatives at low costs.

CCS CONCEPTS

• **Software and its engineering** → *Model-driven software engineering*; *Software performance*;

KEYWORDS

model-based performance engineering, continuous/incremental performance management, incremental reverse engineering, parametric performance model

ACM Reference Format:
Manar Mazkatli and Anne Koziolek. 2018. Continuous Integration of Performance Model. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3185768.3186285

1 INTRODUCTION

Performance is an essential quality attribute of most software systems. Many software systems fail or cannot be used as they are initially implemented due to performance problems. Avoiding these problems requires careful attention to performance goals throughout the development. Software Performance Engineering (SPE) aims to achieve that through predicting whether the proposed software will meet the performance goal or not. It is a common approach to model a system capturing the system behaviour in addition to performance-relevant aspects. Then it solves it by analytical solvers or simulation engines to predict the performance metrics (response times, utilisation and throughput) and support the proactive reactions and design decision making.

However, building such this model takes a great effort and requires keeping it consistent with the source code along the development life cycle [20]. Modern agile software development processes have the goal to start with the implementation as early as possible and thus may skip the design step. For such methodologies, the agile developer use either (1) measurement-based performance evaluation to ensure the performance or (2) extraction of PM and workload model [3]. The first approach provides the actual state of the performance but does not support design decisions. The second approach enables the Model-based Performance Prediction (MbPP) techniques but existing works have two shortcomings:

(A) They do not support the incremental development in the mean of updating the extracted PM. If they were used iteratively, the extraction of the full model would have to be repeated after each iteration (or in larger intervals). Each extraction would have a high monitoring overhead, which would be infeasible in the case of huge Enterprise Applications EAs. Moreover, the modifications of the extracted PM would not be saved to the next iteration.
(B) Most available techniques do not extract the information how Performance Model Parameters (PMPs) (like Resource Demands RDs, loop execution number, the probability of selecting a branch) depend on impacting factors such as input data (so-called *parametric dependencies* [1]).

This paper envisions to reconcile MbPP and agile development by providing more automation. We envision an approach for incrementally extracting the PM and the parametric dependencies. Our approach uses the Continuous Integration (CI) and Continuous Deployment (CD) of the source code and extends them with the Continuous Integration of a Performance Model (CIPM). CIPM is based on the work of Langhammer et al. [15, 16, 14], which incrementally reverse-engineers the behavioural architectural models in terms of the Palladio Component Model (PCM) [19, 18] and keeps it consistent with source code based on Vitruvius platform [11, 4] using user-defined change-driven consistency preservation rules. Langhammer introduced one scenario for enriching the co-evolved model with performance parameters to show that the co-evolved models can be used in principle for the MbPP [14]. However, he does that in a separate step independent of the consistent incremental development process of VITRUVIUS using a measurement-based batch process that suffer also from the aforementioned shortcomings (A) and (B).

CIPM aims to extend Langhammer’s work by the incremental automatic change-driven enrichments of the PM with PMPs considering the parametric dependencies. For this goal, we add a change-driven adaptive monitoring that monitors only the changed parts of code to minimize the overhead. Then, we keep the measurements consistent with both the PM and the code and use them to estimate the PMPs and update PM. Besides, we define a self-validation process of the estimated PMPs. The resulting PM will enable MbPP that supports design decision making for what-if scenarios like the scalability and sizing questions (examples are found in [13]).

For the experimental evaluation, CIPM will use Kieker monitoring tool [7] that supports adaptive monitoring and defining customizable probes.

The next section gives an overview of the foundations. An overview of the envisioned CIPM process is described in section 3. The section 4 describes how our approach keeps the consistency between the PM, measurements and source code. The section 5 describes the CIPM activities in details. The planned evaluation is described in section 6. The related works will be discussed in section 7. The conclusion will follow in section 8.

2 FOUNDATIONS

This section gives a brief overview of the main foundation that our work is based on.

2.1 DevOps

DevOps [3] is an agile *enterprise* software development (iterative and incremental development). DevOps aims to integrate tightly development (Dev) and operations (Ops) teams to continuously adapt EAs to business environment changes, like configuration management, metrics and monitoring schemes, virtualization etc.

2.2 Palladio

Palladio [18] is a software architecture simulation approach that analyses the software at the model level for performance bottlenecks, scalability issues, reliability threats, and allows a subsequent optimisation.

PCM [19] can be used for model-based performance predictions. For this goal, the SEFFs [1] are used to specify the behaviour of component services. A SEFF describes the behaviour of a component service in an abstract level using different control flow elements, such as *internal actions*: a combination of internal computations that do not include calls to required services, *external call actions*: calls to a required service, *loops* and *branch actions*. Both SEFF loops and branch actions include at least one external call, otherwise they will be ignored and combined into an internal action in order to increase the level of abstraction.

2.3 Co-evolution approach

The approach of co-evolution of component-based software architecture and source code [15, 14] is based on the VITRUVIUS view-based approach [11, 4]. It helps software developers and architects evolving their software system by keeping the architecture and the code consistent during the evolution. This approach applies change-driven consistency preservation process based on user-defined change-driven consistency preservation rules between the architectural model and source code. The consistency preservation rules describe the overlap and the model-based transformations that will be used to reinstantiate the consistency. The reactions language of VITRUVIUS can be used to describe the consistency rules. This language consists of two constructs, reactions and routines. A reaction specifies, in reaction to which kind of change some consistency repair logic, defined in a routine, has to be executed. A routine specifies which information has to be extracted and how consistency is repaired. This approach gives a view of the software behaviour through creating the SEFFs.

2.4 Kieker

Kieker [7] is an extensible open-source application performance management tool. It allows to capture and analysis execution traces from distributed software systems. It allows predefinition and customizing of probes as well as the dynamic and adaptive monitoring. Jung et al. [9] introduce an Instrumentation Aspect Language (IAL) to specify where monitoring probes are injected in the code based on available structural information codified in PCM. Moreover, it specifies which attributes have to be monitored using Instrumentation Record Language (IRL), which describes the data structures to represent the monitored information. Heinrich et al. [6] propose a measurements metamodel based on IRL, which stores and manages the measurements traces.

3 CIPM PROCESS

The CIPM is an extension of the iterative, incremental process of the agile as well as DevOps development process. Currently, both agile and DevOps teams rely on automated build, test automation, CI and CD. CIPM aims to increase this level of automation in the development process by providing fast feedback about the performance and enabling MbPP after each iteration. Executing MbPP using the Palladio simulator requires having repository, system, resource environment, allocation and usage models. Most of these models can be extracted using static analysis of source code and test cases or through the dynamic analysis of the monitoring data. Keeping the extracted PM up-to-date along the iterative development process will avoid the overhead resulting from the unnecessary monitoring or analysis. To achieve that CIPM extends the consistent co-evaluation approach as described in section 4.

This extension automates four activities (described in section 5): First, the static analysis of the changed parts of the source code in order to update the static PM, the monitoring probes and the usage model structure. Second, the monitoring of the affected parts of the code to extract the needed monitoring traces. Third, the dynamic analysis of the resulting monitoring traces to estimate PMPs and feed them to the PM. Finally, the self-validation of the extracted PMPs. These activities will enable applying MbPP at anytime to avoid the performance problems and answer the questions of what-if scenarios and sizing investigations.

4 PRESERVING THE CONSISTENCY

CIPM extends the co-evaluation approach to configure the monitoring process and keep also the PCM models consistent with the last up-to-date measurements based on Vitruvius platform. This requires (1) extending the VSUM-repository with an Instrumentation MetaModel (IMM) that describes and manage the instrumentation points as well as the weaving information based on Aspect-Oriented Programming (AOP) [10], (2) Measurements MetaModel MMM that describes the data structures of the different monitored data, and (3) defining the mapping as well as the consistency rules between the MMM, IMM PCM metamodel and source code metamodel.

Regarding (1,2), our approach will extend the measurement metamodel described by Heinrich et al. [6] to describe and manage the needed fine-grained probes (IMM) and implement new trace events for them (MMM). Regarding (3), the needed consistency rules can be defined using Vitruvius reactions and mapping languages. For example, a reaction routine may specify how to react to changes in the internal action of a service. This routine retrieves the internal action ID, archives the old monitoring records related to this ID and insert/activate the measurement points needed to observe this internal action again.

Figure 1 describes the defined reaction routines to keep the models consistent. First, in reaction to code changes, the first reaction routine will execute a change-driven static analysis (see subsection 5.1) and trigger CD automatically. This will trigger the second reaction routine, which starts the monitoring. The monitoring traces will update the Measurements Model MM. The changes in MM will trigger the third reaction routine, which deactivates the related probes, re-estimate the related PMPs and update the PM, as soon as the sufficient number of traces are collected. This number will be determined later according to experiments. The fourth defined routine will be triggered in the reaction of enriching the PM with the calculated performance parameters. This routine will start the simulation to validate the extracted performance parameters and sequentially delete or activate the related probes.

Besides, the developer can manipulate the resulting PCM models at anytime. His changes will be saved during the iterative development process and also be reflected in the corresponding models.

5 CIPM ACTIVITIES

CIPM can be divided into four activities that are applied in each iteration of the consistent DevOps/ agile development process. The following subsections describe them in detail:

5.1 Static extraction of performance model

The goal of the static analysis is to update the PCM repository model (which contains the models of components with their SEFFs) and the usage model as well as the probes needed in the following CIPM step.

As mentioned before, updating the repository model is mainly based on Langhammer et al. [14], which extracts the components with their behaviours in terms of SEFFs. We extract the usage model from the test cases incrementally similar to another work of Langhammer et al. [16] and update it based on monitoring information as explained in subsection 5.3.

The reaction routines generate the appropriate monitoring probes as a reaction to the changes that may have an impact on performance, if it has not already generated by another similar change. Changes with no impact on performance like updating a name of a method will have no reaction.

The CIPM process is independent of the monitoring tool or the adopted dynamic analysis. Therefore, it allows choosing the granularity of the instrumentation based on the approach used in the next step to estimate the PMPs. For example, RD estimation approaches used by LibreDE [23] require coarse-grained monitoring data, like end-user transaction response times and total resource utilization. In this case, service-level probes will be generated based on AOP and stored in the Instrumentation Model IM similar to the mechanism applied by IAL. Moreover, the static analysis will define the record structures based on IRL. Our work, however, aims to also estimate individual RDs of internal actions as well as the parametric dependencies, like the dependency between input data and control flow or data passed to other components and RDs. In our case, service-level instrumentation is required to capture input parameters and to update the usage model, but it is not fine-grained enough to estimate the RDs of the SEFF parts. Therefore, we insert instrumentation points compatible with the SEFF abstraction level based on AOP, if they have been changed from the previous iteration and the instrument points are not already inserted by another change. For examples, the probes can be (a) around the code of an internal action that has been changed, (b) before as well as after an external call to capture input and return data, (c) inside a loop that has been changed to capture the executions number, or (d) inside a branch that has been changed to capture selected branches in addition to the passed parameter.

5.2 Monitoring the changed parts of source code

The monitoring step provides the data required to estimate the PMPs and update the usage model. The monitoring stage will be triggered by CD regardless where the code is deployed (testing, production, canary test environment [8] etc.). This step is not related to specific monitoring tool.

Our approach reduces the downside of the used fine-grained monitoring through monitoring only the source code changes in each iteration on one hand and deactivate the probes after collecting the desired amount of traces on the other hand (see subsection 5.3).

In addition to monitoring the defined probes the resource information, such as the CPU utilization and the memory footprint will be also monitored.

5.3 Dynamic analysis and model parametrization

This step aims to extract the performance parameters as well as to update the usage model. For the first goal, we define a reaction routine that reacts to change in the MM. This routine checks, first, whether the count of the gathered traces is enough. If not, no action will be performed, otherwise, the routine will deactivate the

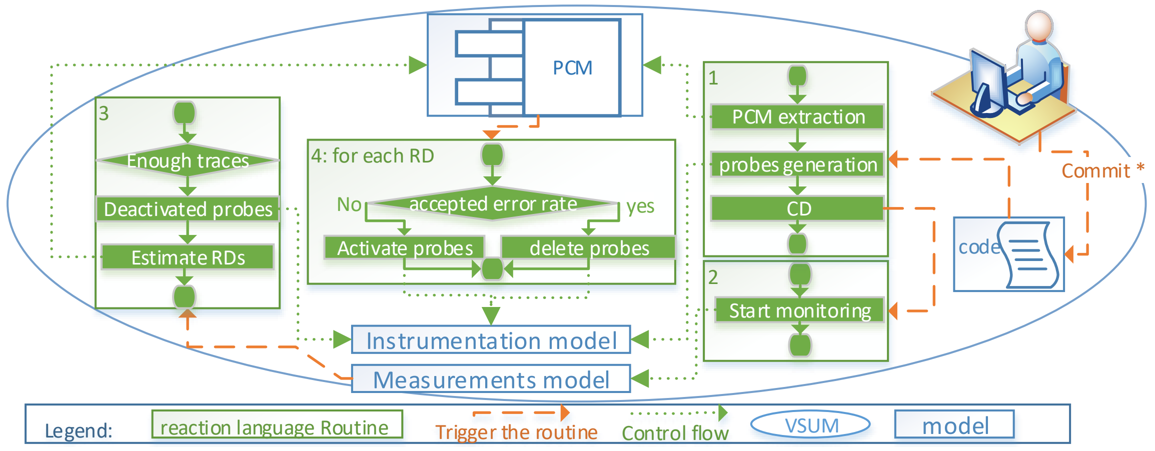

Figure 1: CIPM activities

defined probes and start the dynamic analysis. The dynamic analysis extends the approach of Brosig et al. [2] to estimate the RDs in the case of adaptive monitoring. This approach approximates RDs with measured response times in the case of low resource utilization, typically 20%. Otherwise, they estimate the RD of internal action i (a part of SEFF, see subsection 2.2) for resource r ($D_{i,r}$) based on service demand law [17] shown in equation (1). Here, $U_{i,r}$ the average utilization of resource r due to executing internal action i and C_i is the total number of times that internal action i is executed during the observation period of fixed length T:

$$D_{i,r} = \frac{U_{i,r}}{C_i/T} = \frac{U_{i,r} \cdot T}{C_i} \tag{1}$$

Brosig et al. measure the C_i and estimate $U_{r,i}$ by using the weighted response time ratios of the total resource utilization, which is not applicable in our adaptive case where not all internal actions are monitored. Therefore, we extend their approach to estimate $U_{i,r}$ and as a result $D_{i,r}$ based on the available measurements and the old RDs estimations.

Our new approach is based on the fact that the called internal actions are either Monitored Internal Actions (MIAs) or Not Monitored Internal Actions (NMIAs), which have been already observed in a previous iteration and have, consequentially, an estimation of their RDs. To estimate RDs of MIAs, we estimate the resource utilization due to executing MIAs ($U_{r,MIAs}$), estimate the utilization $U_{i_j,r}$ due to executing each $i_j \in MIAs$ based on the weighted response time ratios of $U_{r,MIAs}$ and estimate $D_{i_j,r}$ based on the service demand law. Then we apply further analysis to extract the stochastic formulation of the RD considering the potential dependency on the input parameters.

In an observation period T we can determine which services are called and which parameters are passed based on the applied coarse-grained instrumentation. Based on that, we can also estimate which internal actions $i_k \in NMIAs$ are processed in this interval ($0 \leq k \leq m$: the number of $NMIAs$). Furthermore, we can measure the C_{i_k}, query D_{i_k} for each i_k and resolve it based on the captured input parameters. Sequentially, we can estimate the utilization $U_{i_k,r}$ needed to execute i_k based on the equation (1). Hence, we can estimate $U_{r,NMIAs}$ the utilization due to executing the NMIAs as summation of the estimated U_{r,i_k}. See the equation (2).

$$U_{r,NMIAs} = \sum_{k=0}^{m} U_{i_k,r} = \sum_{k=0}^{m} \frac{D_{i_k,r} \cdot C_{i_k}}{T} \tag{2}$$

Accordingly, we estimate the utilization due to executing the MIAs ($U_{r,MIAs}$) using the measured U_r and the estimated $U_{r,NMIAs}$ as shown in equation (3):

$$U_{r,MIAs} = U_r - U_{r,NMIAs} \tag{3}$$

Hence, we can estimate the utilization $U_{i_j,r}$ due to executing each $i_j \in MIAs$ using the weighted response time ratios as shown in equation (4), where R_{i_l} is the response time of $i_l \in MIAs$ ($1 \leq l \leq n$: the number of $MIAs$) and C_{i_l} is the number of executing i_l in T.

$$U_{i_j,r} = U_{r,MIAs} \cdot \frac{R_{i_j} \cdot C_{i_j}}{\sum_{l=1}^{n} R_{i_l} \cdot C_{i_l}} \tag{4}$$

Using $U_{i_j,r}$ we can estimate $D_{i_j,r}$ based on the service demand law presented in equation (1).

Next, we use regression analysis to detect whether there is a relationship between the captured input parameters and the values of ($D_{i_j,r}$) that are estimated in each observation period.

If it is found, we store the RD as a stochastic formulation based on the input parameter values, if not, the average value of $D_{i_j,r}$ will be considered as RD estimation.

To differ between the CPU demands and disk demands we suggest detecting the disk-based services in the static analysis stage, through using specific notation or based on the used libraries. After the RD estimation, the reaction routine will enrich the Palladio model with the estimated values.

Besides, further regression analysis is used to detect the dependencies between input parameters on one hand and each of branch probabilities and loop iteration numbers on another hand. Based on the found results, CIPM will update SEFF with either input-based estimation or probability space for loop execution number as well as branch selection.

For updating the usage model (the second goal of the dynamic analysis), we expand an approach by Jung et al. [9]. This approach is based on Kieker and uses the fact that Kieker is able to gather the call traces and the workloads by monitoring frequency of calls to the system. The statistical analysis of these information can extract the suitable probability distributions for workload specification. Similar to the first part of the analysis, the routine will update the palladio usage model.

5.4 Model-based performance prediction

In this step, the MbPP will be started as a reaction to updating the PM with RDs. The goal of this step is the self-validation of the extracted performance model notation. The validation will compare between response time distribution of measurements and simulation results for the extracted model. If the prediction error of an operation is acceptable, the related fine-grained measurement points are deleted, otherwise, the fine-grained probes will be activated and the desired number of traces will be increased, in order to collect more monitoring traces and do further analysis. However, the desired number of traces must not exceed a given limit, which will be determined by experiments.

The performed MbPP results will support the decision making to plan the next step of the iterative development.

6 PLANNED EVALUATION

The planned evaluation aims to cover two points. The first one is the evaluation of defined MIM, MMM as well as the consistency rules between the source code, PCM, IM and MM. The evaluation goal is to ensure that the automatic monitoring and the enrichment of PM are executed correctly. To achieve this goal, we will develop incrementally different case studies and use CIPM process to extract and update the related PMs. The evaluation performs different scenarios to add/ change the code and then asses that the correct code parts are monitored, the correct amount of traces are collected, the monitoring is stopped automatically and PMPs are updated. Based on these experimental evaluations, we will determine also when the monitoring can stop.

The second point is the evaluation of the proposed PMPs estimation. The evaluation will compare the results (the needed overhead as well as the accuracy of the estimated PMPs) with the other available approaches. To do that, we define a set of metrics that measure (1) the *monitoring overhead* based on the total number of measurements needed to extract PM after each iterationand (2) the *accuracy* of the PM based on calculating the error between the simulation metrics and the measurements [1]. These metrics will compare our approach with the approaches extracting the PM from scratch. We expect that the initial commit of our approach can lead to some overhead on one hand. However, we planned to overcome it by deleting the fine-grained probes after collecting enough number of traces. On another hand, we expect that our approach will avoid later a huge part of unnecessary measurements, by filtering the source code changes, monitoring only the affected parts and self-validation of the up-to-date PM.

For experements-based evaluation, we aim to use and extend CASPA platform [5] with CIPM to do the needed experimental evaluations and comparisons with a higher degree of validity, repeatability, reproducibility, and comparability. We aim also to apply our approach to an EA under development to measure the monitoring overhead regarding to each commit. Otherwise, we can use the historical versions of EA to build and update the PM iteratively and show the expected benefit of our adaptive monitoring. In the first step of the evaluation, we adopt the mRubis[1] case study, which has been used by Langhammer to evaluate the co-evaluation approach [14]. The goal is to annotate the extracted PM with PMPs that we extract incrementally based on the dynamic process described in subsection 5.3. For the monitoring we use Kieker tool with new custom probes, like internal actions probes. In the second step, we will implement the defined reactions to automate the instrumentation, enrichment and self-validation of PM.

7 RELATED WORKS

A number of approaches propose means for constructing the architectural model based on the static analysis, dynamic analysis or both of them. For example, Walter et al. [24] propose a Performance Model Extractor (PMX) tool and provided it as a web service [25]. Their approach extracts an architectural performance model as well as performance annotation based on analysing the monitoring traces. Similarly, Krogmann et al. [13, 12] extract parametrised PCM considering the parametric dependency based on the static and dynamic analysis. Langhammer [14] introduces two reverse engineering tools that extract the static behaviour of the underlying source code. The above-mentioned approaches do not support the iterative extraction of PMs as well as the consistency preservation between PMs and source code. If they were used in an iterative development, they would re-extract the whole model by monitoring and analysing the whole system after each iteration. In addition to the monitoring overhead, the re-generation of PM ignores the changes that may be formerly applied to PM, like the parametrisation. Spinner et al. [21] propose an agent-based approach to updated architectural performance models. In contrast to our work, they focus on model updates at runtime.

Other model extraction approaches drive the resource demands either based on coarse grained monitoring data [22, 23] or fine-grained data [2, 26]. The later approaches give a higher accuracy by the estimation but have a downside effect because of the overhead of instrumentation and the monitoring. Our approach aims to reduce the overhead by the adaptive instrumentation and monitoring.

8 CONCLUSION

Applying MbPP in the agile and DevOps process promises supporting proactive performance management based on what-if analysis.

In this paper, we presented a roadmap to continuous integration of the architectural performance model in the agile/ DevOps development process. We propose a lightweight process based on incremental updates of the PM instead of regenerating it after each change. This process is incremental not only in terms of the generation of the structure of the model, but also in the extraction of the PMPs and parametric dependencies based on adaptive monitoring. As a next step, we will evaluate our aspect using historical versions

[1]see: https://www.hpi.uni-potsdam.de/giese/public/mdelab/mdelab-projects/case-studies/mrubis/

of an enterprise application. We aim to validate the correctness of the extracted PM through the comparison between the PM simulation results and the real monitoring results. Moreover, we aim to compare our approach with those based on full generation of the PM after each change using metrics measuring the accuracy and the monitoring overhead.

9 ACKNOWLEDGEMENTS

Thanks go to Jürgen Walter for comments on a preliminary version of the paper. Manar Mazkatli is supported by the German Academic Exchange Service (DAAD).

REFERENCES

[1] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio component model for model-driven performance prediction". In: *Journal of Systems and Software* 82 (2009), pp. 3–22.

[2] Fabian Brosig, Samuel Kounev, and Klaus Krogmann. "Automated Extraction of Palladio Component Models from Running Enterprise Java Applications". In: *Proceedings of the 1st International Workshop on Run-time mOdels for Self-managing Systems and Applications (ROSSA 2009). In conjunction with the Fourth International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2009).* 2009.

[3] Andreas Brunnert et al. *Performance-oriented DevOps: A Research Agenda.* Tech. rep. SPEC-RG-2015-01. SPEC Research Group - DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), Aug. 2015.

[4] Erik Burger. "Flexible Views for View-based Model-driven Development". PhD thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology, July 2014.

[5] T. F. Düllmann et al. "CASPA: A Platform for Comparability of Architecture-Based Software Performance Engineering Approaches". In: *International Conference on Software Architecture Workshops.* IEEE, 2017.

[6] Robert Heinrich et al. "Integrated Observation and Modeling Techniques to Support Adaptation and Evolution of Software Systems". In: *DFG Priority Program SPP1593, 4th Workshop.* Nov. 2014.

[7] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering.* ACM, 2012.

[8] Jez Humble and David Farley. *Continuous Delivery. Reliable Software Releases through Build, Test, and Deployment Automation.* Pearson Education, 2010.

[9] Reiner Jung, Robert Heinrich, and Eric Schmieders. "Model-driven instrumentation with Kieker and Palladio to forecast dynamic applications". In: *Symposium on Software Performance.* Vol. 1083. CEUR, 2013, pp. 99–108.

[10] Gregor Kiczales et al. "Aspect-oriented programming". In: *ECOOP 97—Object-oriented programming* (1997).

[11] Max E. Kramer, Erik Burger, and Michael Langhammer. "View-centric engineering with synchronized heterogeneous models". In: *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling.* VAO '13. Montpellier, France: ACM, 2013, 5:1–5:6.

[12] Klaus Krogmann. *Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis.* Vol. 4. The Karlsruhe Series on Software Design and Quality. KIT Scientific Publishing, 2012.

[13] Klaus Krogmann, Michael Kuperberg, and Ralf Reussner. "Using Genetic Search for Reverse Engineering of Parametric Behaviour Models for Performance Prediction". In: *IEEE Transactions on Software Engineering* 36.6 (2010). Ed. by Mark Harman and Afshin Mansouri, pp. 865–877.

[14] Michael Langhammer. "Automated Coevolution of Source Code and Software Architecture Models". PhD thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT), 2017. 259 pp.

[15] Michael Langhammer and Klaus Krogmann. "A Co-evolution Approach for Source Code and Component-based Architecture Models". In: *17. Workshop Software-Reengineering und-Evolution.* Vol. 4. 2015.

[16] Michael Langhammer et al. "Automated Extraction of Rich Software Models from Limited System Information". In: *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA).* Apr. 2016, pp. 99–108.

[17] Daniel A Menasce et al. *Performance by design: computer capacity planning by example.* Prentice Hall Professional, 2004.

[18] Ralf H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach.* Cambridge, MA: MIT Press, Oct. 2016. 408 pp.

[19] Ralf Reussner et al. *The Palladio Component Model.* Tech. rep. Karlsruhe: KIT, Fakultät für Informatik, 2011.

[20] Connie U. Smith and Lloyd G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison Wesley Longman Publishing Co., Inc., 2003.

[21] Simon Spinner, Jürgen Walter, and Samuel Kounev. "A Reference Architecture for Online Performance Model Extraction in Virtualized Environments". In: *Companion Publication for ACM/SPEC on International Conference on Performance Engineering.* ICPE '16 Companion. ACM, 2016.

[22] Simon Spinner et al. "Evaluating approaches to resource demand estimation". In: *Performance Evaluation* (2015).

[23] Simon Spinner et al. "LibReDE: A Library for Resource Demand Estimation". In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering.* ICPE '14. ACM, 2014.

[24] Jürgen Walter et al. "An Expandable Extraction Framework for Architectural Performance Models". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion.* 2017.

[25] Jürgen Walter et al. "Providing Model-Extraction-as-a-Service for Architectural Performance Models". In: *Symposium on Software Performance: SSP 2017.* 2017.

[26] Felix Willnecker et al. "Comparing the accuracy of resource demand measurement and estimation techniques". In: *European Workshop on Performance Engineering.* Springer. 2015.