A Generic Methodology to Derive Domain-Specific Performance Feedback for Developers

Dennis Westermann
SAP Research, Vincenz-Priessnitz-Strasse 1, 76131 Karlsruhe, Germany
Email: dennis.westermann@sap.com

***Abstract*—The performance of a system directly influences business critical metrics like total cost of ownership (TCO) and user satisfaction. However, building responsive, resource efficient and scalable applications is a challenging task. Thus, software engineering approaches are required to support software architects and developers in meeting these challenges. In this PhD research abstract, we propose a novel performance evaluation process applied during the software development phase. The goal is to increase the performance awareness of developers by providing feedback with respect to performance properties that is integrated in the every day development process. The feedback is based on domain-specific prediction functions derived by a generic methodology that executes a series of systematic measurements. We apply and validate the approach in different development scenarios at SAP.**

I. Problem and Motivation

The performance (i.e. resource usage, timing behavior, and throughput) of a system directly influences the total cost of ownership (TCO) as well as the user satisfaction which are highly business critical metrics. In [6], the founders of companies like PayPal and Hotmail report on the large efforts they had to undertake in order to keep their first application versions responsive and make it scalable with the growing user base. To avoid these last-minute efforts, it is essential to integrate performance evaluation into the overall software engineering lifecycle and ensure early and continuous performance awareness.

In recent years, component-based and service-oriented development have become established approaches for the production of software applications. Today, most software vendors do not develop the whole application but use sophisticated third party components/services or even ready-to-use platforms. Hence, the performance of the application developed by a software vendor depends on the performance characteristics of existing elements ***(Working Hypothesis (WH) #1)***. Although many platform providers offer flexible scalability mechanisms, software vendors still have to ensure that their applications can leverage from these mechanisms. Moreover, with the often applied pay-per-use pricing model the resource efficiency of applications becomes an important cost factor.

In order to enable software developers to build responsive and resource-efficient applications, they have to be provided with information about the performance behavior of the platform on which their applications run and the components and services they use ***(WH #2)***. Generally speaking, it is essential that layers or components of a system not only provide the functional interface but also their capabilities in terms of non-functional behavior like performance.

In the software performance engineering research community many approaches have been introduced that focus on performance prediction based on abstract models of the architecture and behavior descriptions of an application [5]. These approaches show that if the model covers all relevant aspects of the system, they can provide accurate and valuable performance predictions [1]. However, based on our observations the acceptance of these approaches in practice (especially outside of the early architectural design phase) is still low due to the large effort and special knowledge necessary to create and maintain these models ***(WH #3)***. Hence, many real world software development projects still focus on providing functionality first and conduct performance evaluations as part of the overall quality assurance by applying measurements on benchmark scenarios [10].

In order to cover the whole software engineering lifecycle, future software performance engineering approaches have to combine the early-cycle predictive model-based approaches with the late-cycle measurement-based approaches [19]. Furthermore, from our practical experiences we believe that especially in the development phase of an application the performance awareness of developers has to be increased. Today, performance evaluation in the development phase mostly comes with large additional effort for the developers. They are provided with specific literature or can attend trainings and are encouraged to use measurement tools to assess the performance of their developed component. For that reason, the number of developers that continuously evaluate the performance of their developed artifacts is very low. Hence, we require an engineering approach that allows developers to keep track of the performance properties of their developed artifacts without a lot of overhead ***(WH #4)***.

Based on the four working hypotheses outlined in this section, we formulate the following main research question that is the subject of the PhD thesis described in this paper:

How can we simplify the performance evaluation process in the development phase of a software product and increase the performance awareness of software developers?

II. Approach and Research Goals

In this section, we first describe the research goals that drive our work and that are necessary to successfully solve the research question stated above. After that, we outline the main idea of our approach as well as the research goals and challenges that we have to meet in order implement the approach, and briefly introduce two application scenarios.

With the research described in this paper, we aim at answering the research question stated in the previous section by achieving the following goals.

RG1: Simplicity Checking the performance properties of an artifact should be possible with minimal effort and without special trainings.

RG2: Goal Orientation Developers should not be cluttered with tables full of numbers which they have to analyze and interpret. Hence, the feedback should focus on one or two basic target metrics.

RG3: Dependability If a developer does not trust the evaluation results he will not use the feedback to improve his code. Thus, the results should be stable and reliable meaning that there should not be much inaccuracy or ambiguity in the results.

In summary, the main goal of the PhD thesis is to provide a practical performance evaluation process for the development phase of a software product that allows software developers to be continuously aware of the performance of the developed artifacts. Our idea is to provide the software developer with immediate feedback about the performance properties of the artifact under development by integrating a domain-specific prediction function into the development environment of the developer. The domain-specific prediction functions are derived by performance analysts that run a series of systematic experiments on dedicated reference environments.

As the three goals stated above are very domain-specific, we do not aim at a generic solution. For example, the development environment of a web designer differs from the environment of a C programmer which is why one has to decide from case to case how a simple interface for the developer looks like (RG1). The same is true for the definition of the target metrics (RG2). Here, a group of domain experts and performance analysts has to decide in a specific scenario. However, we provide exemplary solutions to these challenges for two development scenarios at SAP. Based on a questionnaire we will gather feedback for these examples from SAP developers. The questionnaire will also assess whether the developers trust the predictions (RG3).

However, in order to implement our idea we need to support the performance analyst with an efficient engineering approach and appropriate methods and tools to derive the feedback functions. In the following, we outline the research goals we want to achieve, along with the corresponding challenges that we face, when implementing this approach.

RG4: Adaptability The fact that we want to provide domain-specific prediction functions brings along that a multitude of scenarios have to be investigated in order to support the majority of developers in an organization. Hence, the approach has to be applicable with different technologies, tools, and abstraction levels as well as on different system layers.

RG5: Accuracy Providing accurate prediction results is the key prerequisite for the approach as it is the most important factor for the reliability of the feedback (see RG3). This comprises that all relevant parameters are included in the function and that the functions correctly quantify the influence of these parameters on the target metric.

RG6: General Applicability As we have to support multiple scenarios (see also RG4), the methods that we apply to derive a prediction function should not make domain-specific assumptions about the structure of the relationship between parameters of the function(e.g., it being linear).

RG7: Efficiency We distinguish between two types of efficiency. The first is concerned with the required manual effort to execute the experiments for a certain domain. If we do not manage to automate a majority of steps, the effort necessary to derive a prediction function for a domain would hinder the practicability. The second type of efficiency is the concerned with the time required to execute systematic measurements in the reference environment. Due to the size of the parameter space spanned by the performance relevant parameters of a domain, it is not feasible to measure all possible points in this multidimensional space. Thus, our approach has to infer accurate multidimensional prediction models with the least possible amount of measurements.

Before we describe the approach by which we aim to achieve the goals and meet the challenges outlined above, we briefly introduce two motivating scenarios that we use to demonstrate and validate our approach.

Scenario 1: SAP's product standards set the goal of sub-second response times in order to achieve a competitive application quality. We observed that for web-based business applications developed with modern UI technologies the rendering time of an application largely contributes to the end-to-end response time and thus requires careful consideration during development. The rendering time is defined as the processing time required by the browser process to parse the HTML and Java-Script code and to render the page. Our goal is to provide a means that allows UI designers and developers to be continuously aware of the expected rendering times of the UI under development so that they can immediately react with countermeasures when the expected rendering time exceeds a certain threshold.

Scenario 2: In the second case study, we apply our approach to a completely different scenario. In this scenario, we target developers that use the Java Persistence API (JPA) to consume database services as part of a platform as a service environment (such as the High Replication Datastore within

Google App Engine). The goal is to provide developers with an estimation function that predicts the database service demands (and thus the expected service costs) generated by the developed entities and the expected usage profile. Using the provided function, developers are continuously aware of the resource efficiency of the current data model and can compare different design alternatives.

In the remainder of this section, we describe our generic approach to support the performance analyst in deriving the domain-specific feedback models. In order to meet RG4, we develop a plugin based framework called Software Performance Cockpit (SoPeCo) [12], [13], [14] that targets at supporting typical steps necessary to conduct performance evaluations. The framework orchestrates different plugins for load generation, monitoring and data analysis while implementing best practices of performance measurements (e.g. ensure that enough data is sampled to derive statistically significant statements). The plugin based architecture allows reuse of existing components and eases the automated repetition of experiments in similar settings (RG7).

The entry point for the performance analyst to SoPeCo is a generic meta-model that allows him (in collaboration with domain experts) to formally describe different domains. This comprises the development artifact (e.g., a web page) and its properties (e.g., the type of stylesheet), individual components (e.g., a table or button) and its properties (e.g. the maximum number of buttons on a page or the maximum size of a table), the relationships between components (e.g., that tables can be nested) as well as the properties of the relationship (e.g., the maximum nesting depth). Moreover, he can specify different execution platforms (e.g., the browser type and version) for which he wants to derive the prediction functions. In addition, he can define different measurement related properties such as the desired accuracy, the maximum measurement time, the analysis technique, or the measurement point selection strategy. Based on the resulting model instance we automatically execute a series of systematic experiments in order to derive the prediction function for the domain. In these experiments, we systematically check which components, relationships, and properties (in the following summarized as parameters) influence the performance metric of interest. Moreover, we test if there are interdependencies between the different parameters. Once we identified those parameters that influence the performance metric of interest and which of them are interrelated, we define different independent parameter groups. Then, we mutually vary the parameter values of one group in order to derive a mathematical prediction function.

At this point, we have to overcome the challenges that come with the research goals RG5, RG6, and RG7 by finding a trade off between providing accurate models, using the least possible number of measurements, and making little assumptions about the structure of the relationship between the parameters involved. Therefore, we developed a methodology that iteratively triggers new measurements, infers a statistical model with the measured data, validates that model and decides if and where to trigger measurements in the next iteration [16]. Using this systematic, iterative combination of measurement and analysis allows us to derive prediction functions with a maximum average relative error of 20% using only up to 10% of all possible measurement points. In [15], we evaluated different measurement point selection strategies and non-parametric statistical analysis methods with respect to their applicability in this iterative approach by applying them in two enterprise application case studies.

Once we derived the prediction function for each of the independent parameter groups we put them together to build the prediction function for the domain. As described earlier, this function can then be integrated into the development environment of the corresponding development groups in order to provide immediate feedback during development.

III. Related Work

Denaro [2] and Gorton [3] deal with performance evaluation in the development phase. Both derive and execute application-specific test cases that they run on existing middleware. In [8], [11] the authors focus on detecting performance anti-patterns in applications. However, in both cases the resulting predictions focus on the overall application and do not directly support the developers but rather the software architect. Approaches that analyze performance differences between two versions of a component [7] support the software developer in finding potential performance problems but do not provide the immediate feedback we aim at in our approach. For the efficient derivation of Software Performance Curves the work of Reussner [9] and Woodside [18] serves as a starting point for our research, but has to be extended to deal with multidimensional parameter spaces.

IV. Planned Validation

We plan to validate our approach in multiple steps using the two application scenarios described in Section II. We demonstrate that the predictions made by our prediction functions conform to the observed reality (RG5). Therefore, we compare for example the rendering time predictions of our functions with the measured rendering time of the corresponding web page. In order to validate whether we met RG1, RG2, and RG3 we plan to prepare a questionnaire that is answered by the SAP development groups which use our prediction functions in their daily work. RG4, RG6, and RG7 are implicitly validated as we show that the approach is applicable in two different scenarios and actually applied in the development groups at SAP. Furthermore, we will report on the time necessary to derive the prediction functions for an existing scenario, as well as the estimated efforts necessary to setup the environment for a new scenario. Finally, we seek to validate that our methodology is superior

to the existing approach at SAP. Therefore, we develop an experimental study with SAP developers that is conducted in the course of a developer training session.

V. Contributions and Progress

The research question that we investigate is: "How can we simplify the performance evaluation process in the development phase of a software product and increase the performance awareness of software developers?". In this paper, we outlined the concomitant research goals and challenges as well as an introduction to the approach by which we plan to meet them. The major scientific contributions are as follows:

a) A novel engineering approach on how to improve performance awareness of developers.

b) A generic meta-model that allows performance analysts to define different development scenarios and the corresponding performance evaluation information necessary to run automated experiments.

c) A methodology for the automated derivation of performance feedback models by applying automated systematic experiments based on a given domain model instance.

d) A framework that enables the performance analyst to run the aforementioned experiments in different scenarios including different technologies and tools. The framework is not restricted to the derivation of performance feedback models for developers as we showed in two publications where we applied it to derive performance completions for model-driven predictions [4] and to optimize costs for web service hosting [17].

e) A methodology that derives multidimensional Software Performance Curves with the least possible number of measurements and without making assumptions on the structure of the relationship between parameters.

f) Two sets of domain-specific experiment environments (WebDynpro, JPA) that can be used to automatically derive performance prediction functions on multiple platforms (e.g. different browsers or different JPA implementations).

g) A detailed validation of the overall approach in an industrial setting at an SAP development department.

References

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

[2] G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. *SIGSOFT Software Engineering Notes*, 29(1):94–103, 2004.

[3] I. Gorton and A. Liu. Performance Evaluation of Alternative Component Architectures forEnterprise JavaBean Applications. *IEEE Internet Computing*, 7(3):18–23, 2003.

[4] J. Happe, D. Westermann, K. Sachs, and L. Kapova. Statistical inference of software performance models for parametric performance completions. In *Proceedings of QoSA 2010)*.

[5] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 2009.

[6] J. Livingston. *Founders at Work: Stories of Startups' Early Days*. Apress, Berkely, CA, USA, 2008.

[7] N. Mostafa and C. Krintz. Tracking performance across software revisions. In *Proc. of 7th International Conference on Principles and Practice of Programming in Java (PPPJ)*, pages 162–171, 2009.

[8] T. Parsons and J. Murphy. Detecting performance antipatterns in component based enterprise systems. *Journal of Object Technology*, 7(3):55–91, 2008.

[9] R. Reussner, P. Sanders, L. Prechelt, and M. Mueller. SKaMPI: A detailed, accurate MPI benchmark. *Lecture Notes in Computer Science*, 1497:52–??, 1998.

[10] J. Sankarasetty, K. Mobley, L. Foster, T. Hammer, and T. Calderone. Software performance in the real world: personal lessons from the performance trauma team. In V. Cortellessa, S. Uchitel, and D. Yankelevich, editors, *WOSP*, pages 201–208. ACM, 2007.

[11] K. Shen, C. Stewart, C. Li, and X. Li. Reference-driven performance anomaly identification. In *In ACM SIGMETRICS*, pages 85–96, 2009.

[12] D. Westermann and J. Happe. Performance Cockpit: Systematic Measurements and Analyses. In *ICPE'11: Proc. of 2nd ACM/SPEC Intern. Conf. on Performance Engineering*, New York, NY, USA, 2011. ACM.

[13] D. Westermann and J. Happe. Software Performance Cockpit. http://www.software-performance-cockpit.org/, March 2011.

[14] D. Westermann, J. Happe, M. Hauck, and C. Heupel. The performance cockpit approach: A framework for systematic performance evaluations. In *Proc. of 36th EUROMICRO SEAA 2010*. IEEE CS, 2010.

[15] D. Westermann, J. Happe, R. Krebs, and R. Farahbod. Automated inference of goal-oriented performance prediction functions. In *ACM/IEEE Int. Conf. on Automated Software Engineering*, page Submitted, 2012.

[16] D. Westermann, R. Krebs, and J. Happe. Efficient experiment selection in automated software performance evaluations. In *EPEW '11: Proc. of 8th European Performance Engineering Workshop*. Springer LNCS, 2011.

[17] D. Westermann and C. Momm. Using software performance curves for dependable and cost-efficient service hosting. In *Proc. of 2nd Workshop on the Quality of Service-Oriented Software Systems*. ACM, 2010.

[18] C. M. Woodside, V. Vetland, M. Courtois, and S. Bayarov. Resource function capture for performance aspects of software components and sub-systems. In *Performance Engineering, State of the Art and Current Trends*, pages 239–256, London, UK, 2001. Springer-Verlag.

[19] M. Woodside, G. Franks, and D. C. Petriu. The Future of Software Performance Engineering. In *Proceedings of ICSE 2007, Future of SE*, pages 171–187. IEEE Computer Society, Washington, DC, USA, 2007.