Empirical Software Engineering — An International Journal manuscript No.
(will be inserted by the editor)

From Monolithic to Component-based Performance Evaluation of Software Architectures

A Series of Experiments Analysing Accuracy and Effort

Anne Martens · Heiko Koziolek · Lutz Prechelt · Ralf Reussner

Abstract *Background:* Model-based performance evaluation methods for software architectures can help architects to assess design alternatives and save costs for late life-cycle performance fixes. A recent trend is component-based performance modelling, which aims at creating reusable performance models; a number of such methods have been proposed during the last decade. Their accuracy and the needed effort for modelling are heavily influenced by human factors, which are so far hardly understood empirically.

Objective: Do component-based methods allow to make performance predictions with a comparable accuracy while saving effort in a reuse scenario? We examined three monolithic methods (SPE, umlPSI, Capacity Planning (CP)) and one component-based performance evaluation method (PCM) with regard to their accuracy and effort from the viewpoint of method users.

Methods: We conducted a series of three experiments (with different levels of control) involving 47 computer science students. In the first experiment, we compared the applicability of the monolithic methods in order to choose one of them for comparison. In the second experiment, we compared the accuracy and effort of this monolithic and the component-based method for the model creation case. In the third, we studied the effort reduction from reusing component-based models. Data were collected based on the resulting artefacts, questionnaires and screen recording. They were analysed using hypothesis testing, linear models, and analysis of variance.

Results: For the monolithic methods, we found that using SPE and CP resulted in accurate predictions, while umlPSI produced over-estimates. Comparing the component-based

Anne Martens
Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany
E-mail: martens@kit.edu

Heiko Koziolek
ABB Corporate Research, 68526 Ladenburg, Germany
E-mail: heiko.koziolek@de.abb.com

Lutz Prechelt
Freie Universität Berlin, 14195 Berlin, Germany
E-mail: prechelt@inf.fu-berlin.de

Ralf Reussner
Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany
E-mail: reussner@kit.edu

method PCM with SPE, we found that creating reusable models using PCM takes more (but not drastically more) time than using SPE and that participants can create accurate models with both techniques. Finally, we found that reusing PCM models can save time, because effort to reuse can be explained by a model that is independent of the inner complexity of a component.

Limitations: The tasks performed in our experiments reflect only a subset of the actual activities when applying model-based performance evaluation methods in a software development process.

Conclusions: Our results indicate that sufficient prediction accuracy can be achieved with both monolithic and component-based methods, and that the higher effort for component-based performance modelling will indeed pay off when the component models incorporate and hide a sufficient amount of complexity.

Keywords Empirical Study · Software Architecture · Performance Evaluation · Performance Modelling · Performance Prediction

1 Introduction

1.1 Problem Statement

Quantitative software architecture evaluation methods aim at supporting the design decisions of software architects. Architectural performance evaluation methods (Balsamo et al., 2004a) rely on formal models, such as queueing networks or stochastic Petri-Nets, derived from software architecture models (described with, e.g., Architecture Description Languages (ADLs) (Medvidovic and Taylor, 1997)). Software architects can predict performance metrics, such as response time, throughput, and resource utilizations, and assess different design alternatives. Shifting the performance evaluation task to the architectural level and assessing performance during design counters the ”fix-it-later” attitude (Smith and Williams, 2002) towards the performance of software systems and avoids architecture-related performance problems, which are expensive to fix after a system is implemented. Most of these architectural performance evaluation methods model the system as a whole within a single development process. We therefore call these methods “monolithic methods”.

A recent trend in the area of architectural performance evaluation methods is component-based performance evaluation, which divides the modelling task among component developer and software architects and create models that can be reused together with the implemented components. The claimed advantage over monolithic methods is to save the effort of creating performance models from scratch each time a component is used.

Human factors influence the successful application of architectural performance evaluation methods to a large extent. Most of the methods rely on developer estimations of resource demands of software components (e.g. how many seconds of CPU are required for an operation) and the anticipated workload of the system. The estimations affect the prediction accuracy to a great extent. Furthermore, developers need to create the performance models in the formalism of the performance evaluation method. Software architects need to deal with understanding the methods, using the often complicated tools, and interpreting their results.

While researchers have proposed many methods in this area, there are almost no empirical studies systematically investigating the achievable accuracy and needed effort of these

methods. Published case studies are often carried out by the method authors themselves, thus not allowing conclusions for third-party applicability. Additionally, it is still unknown whether the component-based methods can deliver the same prediction accuracy as classical, monolithic methods and whether the potential additional effort when creating reusable models can pay off when reusing these models.

1.2 Research Objective

The question driving our empirical investigation was: Do component-based methods allow to make predictions with accuracy comparable to monolithic methods while saving effort in a reuse scenario? Formulated in the goal definition template of (Wohlin et al., 2000), the goal of our work was to

Object	Analyse monolithic performance evaluation methods and a component-based performance evaluation method
Purpose	For the purpose of evaluating their applicability in reuse scenarios and creation scenarios
Quality focus	With respect to achieved accuracy and required effort
Perspective	From the point of view of the method user
Context	In context of performance prediction early in the component-based software development process for business information systems in a graduate university course.

We ran a series of three experiments involving 47 computer science students over the course of three years. In the first experiment, we investigated the accuracy of different monolithic model-based performance evaluation methods (Software Performance Engineering (SPE) (Smith and Williams, 2002), UML Performance Simulator (umlPSI) (Marzolla, 2004), and Capacity Planning (CP) (Menascé et al., 2004)). Based on the results, we selected one of the monolithic methods, SPE, and compared it to a component-based method, Palladio Component Model (PCM) (Becker et al., 2009), investigating accuracy and effort. Finally, we assessed the effort for reusing the component-based performance models of PCM in a third experiment.

1.3 Context

The experiments were conducted in a university environment with student participants at Masters level. The analysed systems were simplified versions of realistic component-based business information systems in that they provided only part of the functionality a corresponding realistic system would need to provide (e.g. limited exception handling, no session management). We studied the performance evaluation step embedded in the component-based software development process. The task descriptions were specifically designed to reflect the information and artefacts available at this step of the software development process. The participants assumed the role of performance analysts and evaluated the given design for performance (i.e. response time of use cases for a given workload). We asked them typical performance questions for this step, such as to evaluate the performance effect of different design decisions. They solved the tasks in a series of supervised experiment sessions with time constraints.

1.4 Contribution and Outline

Three former publications (Koziolek and Firus, 2005; Martens et al., 2008b; Martens et al., 2008a) have reported on the design and results of the first two experiments. The contributions of this paper are (i) the addition of a third experiment to analyse the effort for model reuse and (ii) the analysis of the results of all experiments from a holistic viewpoint. We have partially re-evaluated the data of the former experiments and documented the results in a consolidated form. Our results indicate that a sufficient prediction accuracy can be achieved both with most monolithic methods and the component-based method. The effort for creating parametrised, reusable models from scratch is in average almost 75% higher than for monolithic models, but the effort can pay off as the reuse of the models induces only limited efforts.

The remainder of this paper is organised based on the reporting guidelines in (Jedlitschka et al., 2008). Section 2 provides background on performance evaluation methods for software architectures, about empirical validation of performance evaluation methods, and presents related studies. Section 3 describes the overall planning of our series of experiments and explains their commonalities. The next three sections describe the experiment-specific planning, analysis and results of the three experiment in the series: Section 4 covers the comparison of accuracy and effort for the three monolithic methods SPE, umlPSI, and Capacity Planning, Section 5 covers the comparison of accuracy and effort for the monolithic method SPE vs. the component-based method PCM, and Section 6 covers the analysis of effort for reuse vs. model creation for the component-based method PCM. Section 7 then discusses the results from a holistic perspective, names threats to validity, and outlines future research directions. Section 8 concludes the paper.

2 Background

In this section, we first describe the background of model-driven performance evaluation methods of software architectures: In Section 2.1, we present the model-based performance evaluation methods that target to model the system as a whole (thus, we call them monolithic methods). In Section 2.2, we describe the newer field of component-based performance evaluation methods.

In Section 2.3, we set the context for the empirical validation of performance evaluation methods in general. We identify different types of validation and classify the studies presented in this work. Finally, in Section 2.4, we present and discuss the few existing empirical studies in the field of model-driven performance evaluation methods.

2.1 Monolithic Methods

More than 20 model-based performance evaluation methods have been surveyed by Balsamo et al. (Balsamo et al., 2004a). The goal of these methods is to quantitatively support software architects in selecting among different design alternatives. Because of the uncertainty about the system during design, these methods often involve estimations. Thus, these methods often cannot predict the *absolute* performance metrics, such as response time, throughput, and resource utilization with a high accuracy compared to measurements. Instead, their accuracy is often sufficient to evaluate the *relative* differences between different design alternatives, as the errors made during modelling apply for all modelled design alternatives.

Most of the methods follow a similar, coarse process models as depicted in Fig. 1. First, the software architect has to create a software model annotated with performance properties, such as resource demands of software components and the anticipated user workload. The modelling task can start from existing architectural models, which can then be enhanced with performance-related information. Different notations, such as annotated ADLs or use case maps have been proposed for this purpose (Balsamo et al., 2004a). Recently, a UML performance modelling profile called MARTE (Object Management Group (OMG), 2006) has been standardized by the OMG to capture the performance related information in stereotypes and annotations. Software architects can estimate resource demands (e.g., execution times for components) based on experience with former systems or measure small prototypes of the designed architecture. Workload descriptions are often based on requirements documents and for example state the number of users concurrently interacting with a system.

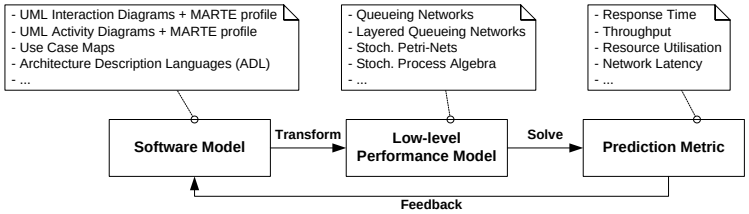


Fig. 1 Model-Driven Performance Evaluation Using Monolithic Software Models. Boxes represent artefacts, arrows represent activities. The annotations list existing realisation examples for the artefacts

Once the software architect has build a complete, annotated software model, model transformations can map it into classical, low-level performance models, such as (layered) queuing networks (Lazowska et al., 1984; Rolia and Sevcik, 1995), stochastic Petri nets (Heitmann and Moldt, 2007), or stochastic process algebra (Hermanns et al., 2002). The distinction between the developer-friendly software model and the mathematically-oriented low-level performance model keeps the complexity of the underlying mathematical concepts hidden from the software architect. In the ideal case the model transformations and performance models are completely encapsulated into tools, so that software architects can use them transparently.

For the classical low-level performance modelling notations, a large body of theory and many tools exists, which solve the models and derive the desired performance metrics, such as response time or throughput (Jain, 1991). To keep the low-level performance model hidden from the software architect, the prediction results should be fed back into the original software model.

Examples of these methods are Software Performance Engineering (SPE) (Smith and Williams, 2002), UML Performance Simulator (umlPSI) (Marzolla, 2004), and Capacity Planning (CP) (Menascé et al., 2004). SPE provides a comprehensive process model. It uses so-called execution graphs or annotated message sequence charts as software models and transforms them into extended queueing networks using the tool SPE Editor (SPE·ED). SPE·ED can then also solve these models with numerical techniques or with simulation. SPE has been applied to several industrial case studies (Williams and Smith, 2002).

umlPSI is based on the UML profile for Schedulability, Performance, and Time (SPT) (Object Management Group (OMG), 2005) and uses annotated UML activity and deployment diagrams for creating the software models. The tool umlPSI then transforms the models in

a discrete event simulation and can also feed back the results into the UML models. No industrial application of this method in practice is known, and the tool is less stable compared to the other two.

Capacity Planning is usually used for the performance analysis and prediction for already implemented systems. Developers describe the system's architecture informally and collect performance information through system monitoring and testing. A queueing network is constructed from this and can be solved using various tools. CP has been applied in industry, for example for client/server systems (Menasce et al., 1994) and e-business systems (Menascé and Almeida, 2000).

The methods described above do not exhibit concepts for parametrised, reusable model elements. They model the system as a whole. Thus, the created models are often used only once after they have been built. While it is in principle possible to model component-based system with these monolithic methods, they do not exploit the benefits of componentisation, such as division of work, separation of concerns, and reuse.

2.2 Component-based Methods

Recently, more than 10 component-based performance modelling methods have emerged (Koziolek, 2010). The key idea of these methods is to build parametrised, reusable performance models for each component, thus dividing the modelling task among software architects and component developers. The performance of a software component is influenced by the parameter values it is called with, the hardware/software platform it is deployed on, and the performance of required services (Becker et al., 2006). All these factors are unknown to the component developer when creating the component performance model. Therefore special modelling notations are required, which allow to express these influence factors explicitly as parameters of the model.

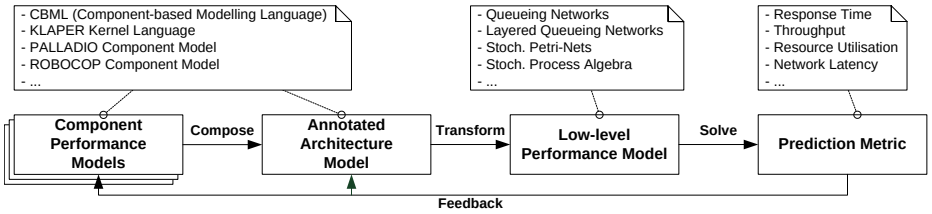


Fig. 2 Model-Driven Performance Evaluation Using Component-based Models. Boxes represent artefacts, arrows represent activities. The annotations list existing realisation examples for the artefacts.

Fig. 2 shows the simplified process model of component-based performance modelling methods. The parametrised component performance models (i.e., behavioural specification with resource demands) are retrieved from a repository and composed by a software architect depending on the desired application architecture. Additionally, the system-level workload of the application as well as the hardware resource environment has to be described to complete the annotated architectural model. Afterwards, the same model transformation and model solution techniques as for the monolithic methods can be applied. While the effort for creating parametrised models is intuitively higher than for throw-away models, the component-based methods promise that the effort can pay off after the model has been

reused only a few times. As a more concrete example, Fig. 3 shows an instantiation of the process for a simplistic component-based architectural model.

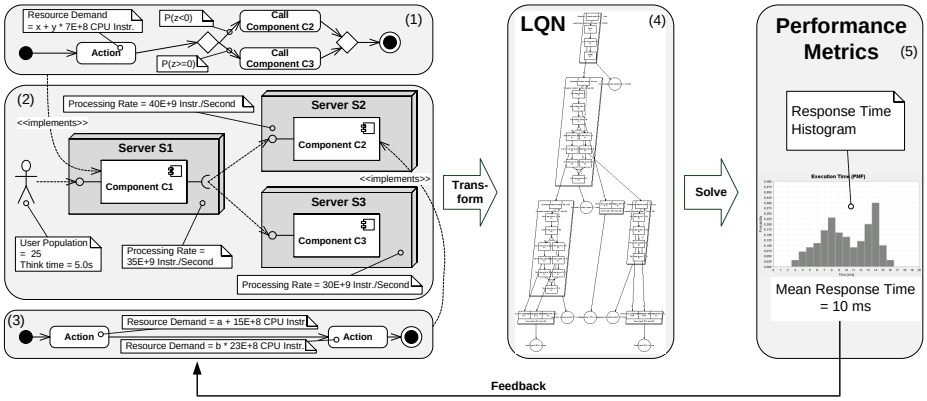


Fig. 3 Component-based Performance Evaluation Example (UML + LQN). The left part shows the software system model in UML: Component developers have supplied parametrised, behavioural specifications for the component services in the form of annotated UML activity diagrams (1) and (3). These specification abstractly describe the control flow through the component as a graph of actions. Calls to required services are included (e.g. in (1)). The resource demands for component internal actions can be specified in dependency to input parameters from the component signature. For example, the resource demand of component C1 depends linearly on the values of input parameters x and y (first action in (1)). Furthermore, the branch conditions inside the component depend on the value of input parameter z. The architecture diagram (2) contains processing rates to solve the parametrisation over hardware resources as well as the expected user population of the system. Such a model contains enough information for a model transformation to a Layered Queueing Network (LQN) (Franks et al., 2009) (4) and the subsequent derivation of performance metrics (5).

Of the 13 component-based performance evaluation methods surveyed in (Koziolek, 2010), the only two methods supporting all described levels of parametrisation are PCM (Becker et al., 2007) and ROBOCOP (Bondarev et al., 2004).

PCM features a meta-model based on the Eclipse Modelling Framework (EMF) and provides specialised modelling languages for component developers and software architects. The component performance models used by PCM are parametrised for different usage profiles, hardware resources, and required services as described above. After different component performance models have been composed into an architectural model, they are transformed into a discrete event simulation (SimuCOM) to derive the desired performance metrics. The PCM has been applied in various industrial projects (Brosig et al., 2009; Huber et al., 2010).

ROBOCOP is based on the ROBOCOP component model and features similar parametrisation concepts as PCM. ROBOCOP targets schedulability analysis for embedded systems and therefore transforms the created models into an according simulation model. In (Bondarev et al., 2007), the authors themselves applied ROBOCOP to an industrial case study on a JPEG decoder.

For our experiments, as component-based method, we chose the PCM, because ROBOCOP does not target our domain of interest of business information systems, but embedded systems.

While component-based performance modelling methods have been proposed in research, they are uncommon in industrial practice. They promise the reuse of models and

thus to save efforts for performance modelling. However, it has been unknown if these methods can deliver a similar prediction accuracy as monolithic methods. While it is intuitively clear that the effort for creating parametrised, reusable models is higher than for monolithic methods, it is not known whether the effort can pay off.

2.3 Empirical Investigation in Context: Validation Types

Our empirical investigation should be viewed in context of other types of research validations. For model-based prediction methods, we have classified the different forms of validation into three distinctive types (Böhme and Reussner, 2008):

- **Type I (Model Accuracy):** Type I studies validate the accuracy of the models, theory, and analysis tools. This is the simplest form of evaluation, in which it is assumed that all input information is available and accurate. The authors of a method create a model, conduct the predictions supported by a tool, and compare them to performance measurements. It requires an implementation of the model language, the analysis tools, and the system under study. The performance annotations are usually based on measurements. Examples can be found in (Liu et al., 2005; Kounev, 2006; Koziolek et al., 2008).
- **Type II (Applicability):** Type II studies evaluate the applicability of a method, when it is used by the targeted developers instead of the method's authors. This includes investigating the maturity of the accompanying software tools and the interpretability of the results delivered by the tools. Type II studies can also check whether third parties are able to estimate performance annotations with sufficient accuracy. Such a study may involve students or practitioners. It is desirable to let multiple users apply the method to reduce the influence of the individual and gain general results.
- **Type III (Cost Benefit):** This form of evaluation is a cost-benefit analysis. Applying a prediction method leads to higher up front costs for modelling and analysis during early stages of software development. Model-based prediction methods claim that these higher costs are compensated by the reduced need for revising the architecture after implementation or fixing the code. A Type III study checks this claim by conducting the same software project at least twice, once without applying the prediction method and accepting the costs for late life-cycle revisions, and once with applying the prediction method thereby investing higher up front costs. Comparing the respective costs for both projects enables assessing the benefit of a prediction method in terms of business value. Due to the high costs for such a study, it is very seldom conducted.

A second dimension for the classification is the external validity of a method. If the authors of a method create their own example systems under laboratory conditions, they will tend to choose systems that emphasise the benefits of their method. All three types of validations can be performed under laboratory conditions or in the field. However, field experiments with industry systems are expensive and time consuming.

Together, the two dimensions of classification are shown in Fig. 4. The studies presented in this work have been performed under laboratory conditions and address Type II (applicability) validation.

2.4 Related Studies

Most empirical validations of model-based performance evaluation methods are Type I validations of a single method, e.g. in the proceedings (Dujmović et al., 2004; Dumke et al.,

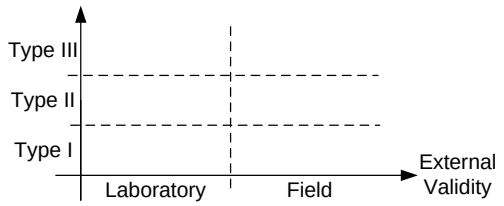


Fig. 4 Evaluating Model-Based Prediction Methods. The three types of validation are labelled on the y-axis. The dimension of external validity is shown on the x-axis.

2001). Some methods do not offer stable tool support (Koziolek, 2010) and are therefore difficult to evaluate empirically with third-party users. In some few studies, e.g. (Balsamo et al., 2004b) and (Bacigalupo et al., 2004; Bacigalupo et al., 2005) (see below), several performance evaluation methods were compared and aspects of applicability were considered. However, in these studies, the evaluation was still carried out by the authors themselves, which limits generalisation to third-party users.

(Balsamo et al., 2004b) compared two complementary performance evaluation methods (Æmilia, based on analytic performance evaluation and umlPSI, based on simulation) using several criteria (performance model derivation, software model annotation, generality, performance indices, feedback, scalability, and integration). It was found that both approaches were overall feasible with an "affordable effort" (no exact numbers were given), while there were further advantages and disadvantages of each method in detail. However, the authors of the methods carried out the predictions themselves. Thus, the results cannot be generalised to applicability by third-party users.

Bacigalupo et al. compared their performance evaluation technique HYDRA (Bacigalupo et al., 2003) to the Layered Queueing Network (LQN) method in two case studies (Bacigalupo et al., 2004; Bacigalupo et al., 2005). HYDRA extrapolates from historical performance data for different classes of workload and is used for grid applications. LQNs explicitly models the control flow, the resource demands and the resources in the system. In both cases, benchmarks were used to generate loads for measurements and to validate the predicted values. Both methods were found to be applicable for the system under study and result in accurate predictions. LQNs required less effort and expertise for modelling. Again, the authors applied the methods themselves.

As a first step towards Type III validation, (Williams and Smith, 2003) present the use of business case analysis to analyse the cost-benefit of SPE projects and exemplarily calculate the return-of-investment for a small case study. However, the estimations are not validated against data from real projects.

3 Overview of Our Three Experiments

In this section, we present the goal of this work in more detail as well as the common aspects of the three successively conducted experiments. Section 3.1 refines the overall goal of this study presented in the introduction resulting in one concrete goal per experiment. Section 3.2 characterises the participants. Section 3.3 describes the experiment material and Section 3.4 summarises the tasks for the participants. Finally, in Section 3.5, we present the preparation of the participants. Information on the parameters, hypotheses, experiment execution, and analysis is given separately for each experiment in Sections 4 to 6.

3.1 Goals

We gradually identified three concrete goals for three separate experiments based on the main research goal presented in the introduction. The viewpoint remains the point of view of the method user in all three goals, so we omitted it from the tables in the following.

First of all, the applicability of all model-driven performance evaluation methods when used by average users (in contrast to the developers of the methods) was unclear. A major factor for applicability is the achievable accuracy. Because monolithic methods do not explicitly support model reuse, we assumed that all input models were created from scratch based on existing design documents.

Goal 1	
Object	Analyse SPE, umlPSI, and CP
Purpose	for the purpose of evaluating their applicability for model creation from scratch
Quality focus	with respect to achieved accuracy
Context	in the context of a software performance engineering course.

Component-based methods allow to create reusable component performance models and target to save effort when reusing them. It was unclear whether the same accuracy as with monolithic methods is achievable for third-party users, how much larger the effort to create the models is and how much effort can be saved when reusing the models.

In the second experiment, we studied the differences in accuracy and effort for creating new performance models with the monolithic method SPE and the component-based method PCM. From the monolithic methods, we chose SPE: First, because SPE and umlPSI are better suited for our context of business information systems, and second, because SPE was superior to umlPSI with respect to prediction accuracy in the first experiment.

Again, we compared the methods for the setting that the models are created from scratch, without reusing parts of them. As SPE does not explicitly support reuse, only this setting allows to compare both methods. The advantage of reusable models is covered by Goal 3.

Goal 2	
Object	Compare SPE and PCM
Purpose	for the purpose of evaluating their applicability for model creation from scratch
Quality focus	with respect to achieved accuracy and effort
Context	in the context of a software performance engineering course.

To finish the analysis of the applicability of the component-based method, we analysed the claim that having reusable component performance models saves effort in the reuse case. As the reuse case is not explicitly supported for monolithic models, we did not compare the component-based method with monolithic methods here.

Goal 3	
Object	Compare the model reuse case and the model creation case (both with PCM)
Purpose	for the purpose of determining whether reuse pays off
Quality focus	with respect to the effort
Context	in the context of usage by students.

3.2 Participants

The participants of all experiments were computer science students enrolled in a Masters program [1]. Overall, 47 students participated in the series of experiments. No participant took part in more than one experiment. The participants of the first two experiments signed up for a course on software performance engineering and then chose to voluntarily participate in the experiment. Most of the participants had no experience with software performance engineering before the training, two had a little. For the third experiment, we asked 2 student research assistants, 1 student working on a Masters thesis, and 1 first-year PhD student. All 4 participants had limited, but comparable PCM experience. Overall, all participants had previous software development experience or had attended software engineering courses (or both); however, the length of their previous experienced ranged from less than one to 14 years. Details on the participants can be found in (Koziolek, 2004a; Martens, 2007; Martens et al., 2009).

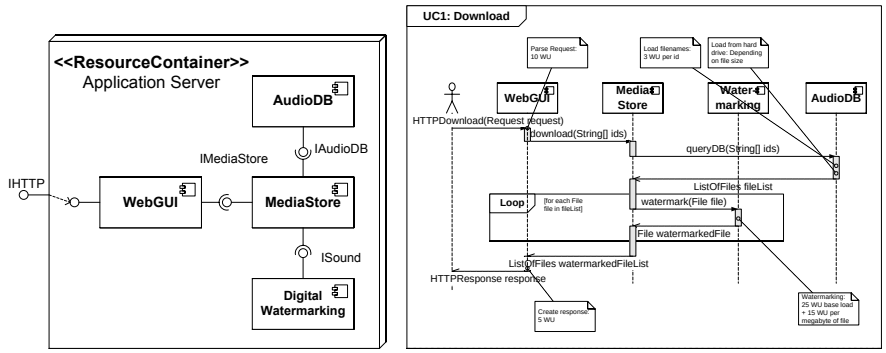
Because of the rather inhomogeneous group, participants were assigned to the experiment groups of the first and second experiment based on their performance in the training exercises (cf. numbers in Fig. 6, p. 15 and Fig. 8, p. 19). For the first experiment, we used the scores $s_{SPE}, s_{umlPSI}, s_{CP}$ achieved by the participants in the training exercise of the respective approach. The participants were assigned to three treatment groups $t_{SPE}, t_{umlPSI}, t_{CP}$ so that each group t_i had a similar number of participants with better-than-average score s_i. In the second experiment, we created two treatment groups based on the participants' overall score in the training. We assigned the participants to two treatment groups so that each group had a similar number of participants with better-than-average score. This stratified randomization reduces accidental inter-group qualification differences and hence strengthens the internal validity of the experiments. In the third experiment, all participants solved the same task (same treatment).

3.3 Experimental Material

All material to conduct a performance evaluation was made available to the participants as if they conducted performance evaluation early in the software development process. This included the needed performance evaluation tools, a description on the software system under study, and documentation on the tools and methods (accessible over the web).

The description of the software system under study included three parts: A description of the systems architecture, information about the system environment, and information about the performance properties of the system. The architecture description included static, behaviour, and allocation views of the system. The views were visualised using UML diagrams and described in natural language. The system environment included the resource environment as required by the performance evaluation method (e.g. what hardware is available, what is the processor speed, what is the throughput of hard disks) and a description of the usage of the system (how many users use the system concurrently, what input data is passed to the system, e.g. file size of input files). Finally, performance properties of the system were described (e.g. when is the disk accessed or how many CPU cycles are needed for a certain operation). For experiment 1 and 2, we also provided a list of possible design alternatives to the initial system design.

[1] to be more precise: *Hauptdiplom* part of the German *Diplom* program, which is similar to a Masters program.



(a) Static and allocation view

(b) Behavioural view with annotated performance properties: WU denotes an abstract work unit measure on the CPU. The CPU speed is then defined in terms of WU/sec.

Fig. 5 Architecture description of the initial Mediastore system visualised with UML.

Our experiments use two example systems, both created for this purpose in different versions. The first is a Webserver that can generate dynamic pages or deliver static ones (Koziolek, 2004a, Sec. 3.2.1),(Martens, 2007, Sec. 4.2.2). The second is a Mediastore for downloading and uploading MP3 and video files (Martens, 2007, Sec. 4.2.2). Part of the description as provided in the second experiment for the Mediastore system is shown as an example in Fig. 5.

3.4 Tasks

In all three experiments, the task of the participants was to predict performance metrics for the system described in the experiment material. The task included the following subtasks:

1. To model the architecture as required by the method,
2. To model the system environment as required by the method,
3. To estimate performance properties (experiment 1 only) and/or insert given performance properties (all experiments),
4. To run the performance evaluation tools,
5. To interpret the predicted response time metrics.

For experiment 1 and 2, the participants were additionally asked to model and evaluate a number of design alternatives, as this is a common task when performing performance prediction. For each design alternatives, the additional subtasks were

6. To vary the created models to represent the design alternative,
7. To run the performance evaluation tools for the design alternative,
8. To interpret the predicted response time metrics.

Finally, for experiment 1 and 2 only, participants were asked

9. To assess and rank all design alternatives based on the mean response time

Goal 1	**Analyse Accuracy of Monolithic Methods**
Q1	*What is the accuracy of monolithic methods?*
M1.1	*fracOfActual*: Predicted mean response time as a fraction of the actual mean response time, ratio scale with rational values > 0
M1.2	*score*: Normalised number of wrong relations in predicted ordering compared to reference ordering, ratio scale with rational values $[0,1]$
PEM	Factor: Used performance evaluation method with levels $\{SPE, CP, umlPSI\}$
a	Independent variable: Design alternatives with levels $\{0, 1a, 1b, 2, 3, 4\}$
accuracy	Dependent variable, measured with metric M1.1 and M1.2

Table 1 Summary of Goals, Questions, Metrics, and Variables for Experiment 1

3.5 Preparation

Before the experiment sessions, participants were trained in the performance evaluation methods. They learned how to use the methods and tools, but were not required to understand the underlying theory in detail. The training included lectures on the used performance evaluation methods (experiment 1 and 2 only) and self-study (experiment 3). Exercises were assigned and had to be completed before the experiment session. The exercises were checked by us and errors were discussed with the participants.

The specific preparation phases for each experiment are depicted in Figures 6 (p. 15), 8 (p. 19) and 12 (p. 25). Details for the experiments are presented in (Koziolek, 2004b, Section 5.4), (Martens, 2007, Section 4.1.1), and (Martens et al., 2009).

4 Experiment 1: Accuracy and Effort of Monolithic Methods

4.1 Planning

4.1.1 Tasks

In this experiment, participants were asked to predict the performance for the initial Web-server system (0) and for five design alternatives. The five design alternatives were (1a) a cache for static HTML-pages, (1b) a cache for dynamically generated HTML-pages, (2) a single-threaded version of the server, (3) application of a compression-algorithm before sending HTML-pages and (4) clustering of the server on two independent computers.

The required subtasks to perform the performance predictions are described in Section 3.4. The original task description and information on the example system can be found in (Koziolek, 2004a, Section 3.2.3).

4.1.2 Hypotheses, Parameters, and Variables

Based on the GQM template of (Basili et al., 1994), we derived questions and metrics for the goals of this experiment, and formulated hypotheses. The metric formulas have been updated for this paper to allow a common presentation of all experiments. Table 1 summarises the goals, questions, metrics, and variables.

The factor (in terms of controlled independent variable) for the first two experiments was the used performance evaluation method *PEM*. The unordered set of considered values was $\{SPE, CP, umlPSI\}$. An additional independent variable for the first two experiments

was the evaluated design alternative a. The levels of the design alternative variable a were $A = \{0, 1a, 1b, 2, 3, 4\}$.

The exploratory question we asked was Q1: *What is the accuracy of monolithic methods?* We identified two metrics to measure the accuracy of the method. The results will be shown as descriptive statistics.

First, the performance models should deliver values that are similar to the actual, correct values. Here, the predicted mean response time *prt* was the relevant performance metric. For goal 1, we used measurements of the implemented system to obtain the actual mean response time *art*. To assess for which method the *prt* was closer to the *art* of the system, we looked at the fraction:

$$\text{M1.1: } fracOfActual = \frac{prt}{art}$$

For early design stages, it is difficult to estimate the absolute values of mean response time. At the same time, it is more interesting to assess the relative difference between mean response time of design alternatives than to obtain absolute values (simple-model strategy, (Smith and Williams, 2002, p.18)). Thus, to assess the accuracy of a method, we asked the participants to order the design alternatives based on the response time they predicted: The smaller the mean response time, the better. Then, we compared the design alternative ordering created by the participants with a reference design alternative ordering created based on the measurements of the implemented system. If two design alternatives lead to very similar results, we considered them to be equivalent in the reference ordering.

We defined a total preorder $R_{ref} \subset A \times A$ on design alternative based on the measured actual mean response time *art* that reflects "better or equivalent to". $(a_1, a_2) \in R_{ref}$ means that design alternative a_1 is better or equivalent to design alternative a_2. If $(a_1, a_2) \in R_{ref}$ and $(a_2, a_1) \in R_{ref}$, then a_1 is equivalent to a_2. If, on the other hand, $(a_1, a_2) \in R_{ref}$ and $(a_2, a_1) \notin R_{ref}$, then a_1 is better than a_2.

The participants were asked to define a strict total order $R_{pred} \subset A \times A$. We counted how many wrong relations have been established by the participants that were not contained in our reference ordering:

$$\text{Count of wrong relations } \#w = \left| R_{pred} - R_{ref} \right|$$

Let us consider an example here. The reference ranking for the Webserver system is that alternative 3 is ranked best, alternatives $1a$ and $1b$ share the second rank, and alternative 4 is ranked worst. Then, $R_{ref} = \{(3,1a),(3,1b),(3,4),(1a,1b),(1b,1a),(1a,4),(1b,4)\}$. Assume that a participant ranks alternative 1a first, followed by 3, 1b, 4 in that strict order. Then, $R_{pred} = \{(1a,3),(1a,1b),(1a,4),(3,1b),(3,4),(1b,4)\}$. The difference set is $R_{pred} - R_{ref} = \{(1a,3)\}$, thus, we count one wrong relation.

To cope with varying number of ranks (due to equivalent design alternatives or incomplete rankings), we normalised by the maximum possible $\#w$ denoted by *maxW*. The maximum *maxW* could be obtained by calculating $\#w$ for the inverse ordering of R_{ref}. In our example, the inverse ordering of R_{ref} is $\{(1a,3),(1b,3),(4,3),(1b,1a),(1a,1b),(4,1a),(4,1b)\}$. Then, the difference set is $\{(1a,3),(1b,3),(4,3),(4,1a),(4,1b)\}$. Thus, $maxW = 5$ in this example.

MetricM1.2 measured the normalised number of wrong relations:

$$\text{M1.2: } score = \frac{\#w}{maxW}$$

The metric was a ratio scale with rational values in $[0, 1]$, with 0 meaning a correct prediction.

4.1.3 Design

The experiment was designed as a one-factor-3-treatments design as depicted in Fig. 6 in the lower "Experiment" part. Each participant used one method (extra-subject design) to limit the effort for the participants.

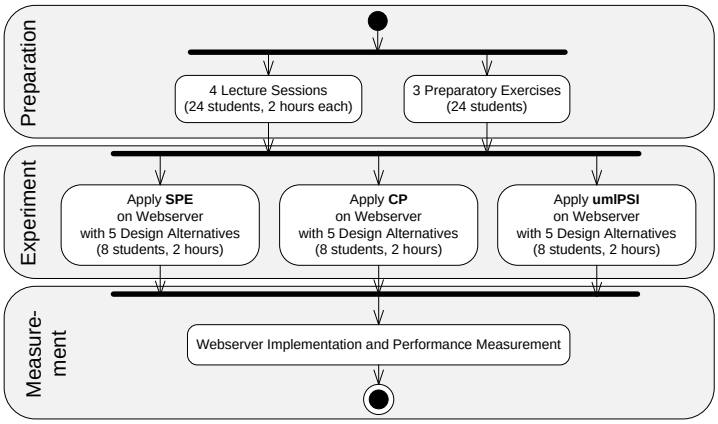


Fig. 6 Design of Experiment 1, visualised as an activity diagram. Nodes represent activities. Activities surrounded by bars represent parallel activity. The activities are grouped into the three phases of the experiment. Maximum time constraints are named for the experiment sessions.

4.1.4 Procedure

The experiment took place in a university computer lab. Participants were not supposed to speak with others during the experiment session. Members of our research group were present to help with tool problems or the task description. They did not influence the performance modelling of the participants.

After completion, the participants wrote down a ranking of all design alternatives (for metric M1.2: misplacements in ranking) and handed in the created models and prediction results, which allowed us spot checks for consistency.

After the experiment, the consistency check indeed revealed some elementary mistakes regarding the experiment task that distorted the results. Thus, the participants were asked to correct their solutions later. Mistakes regarding the performance evaluation methods were not corrected. After the corrections, 22 solutions were valid and could be used for the analysis. Design alternative (2) was omitted in the analysis because only about half of the participants evaluated it.

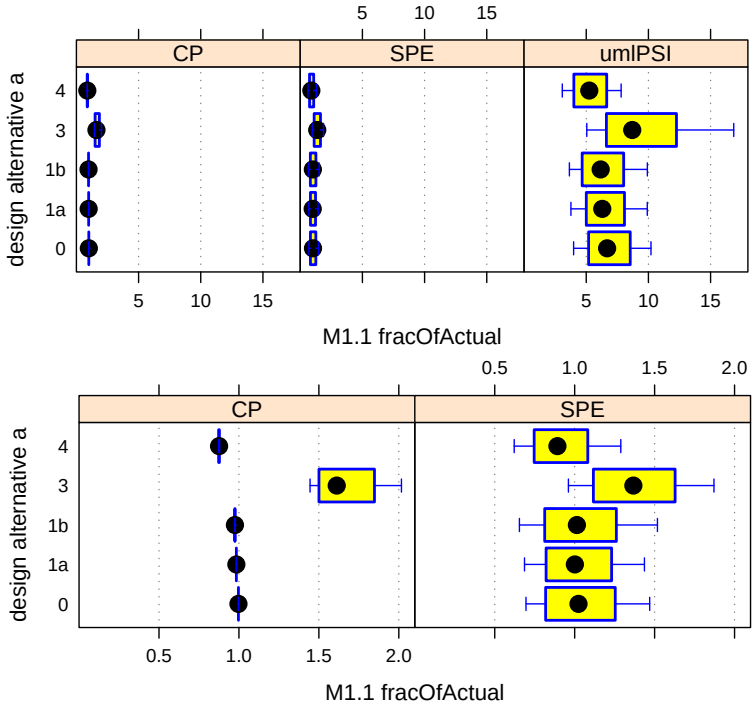


Fig. 7 M1.1: Variability of the predictions (predicted response time) made with different methods. Each boxplot shows the distribution of 1000 resampled 3-pooled predictions p, each computed as the average of a random three subjects' predictions x_i, x_j, x_k (together: x) plus some normally distributed noise for smoothing: $p = \frac{1}{3}(\mathcal{N}(x_i, 0.5\sigma(x)) + \mathcal{N}(x_j, 0.5\sigma(x)) + \mathcal{N}(x_k, 0.5\sigma(x)))$. This amounts to simulating the variability of committee predictions when three people similar to our subjects produce each prediction together but without discussion. It assesses the variability and robustness of prediction for each method. The box marks the 25- to 75-percentile, the fat dot is the median, the whiskers mark the 10- and 90-percentile. The lower figure shows details from the upper one.

4.2 Analysis

For the deviation of the predicted mean response time to the actual measured mean response time (metric M1.1: *fracOfActual*), we assessed the variability of the predictions using resampling as shown and explained in Fig. 7. As we see in the top part, umlPSI produced vast overestimates (a median of 6 means a 500% overestimation is typical) and also much higher variability. The bottom part tells us for CP and SPE that four of the five design alternatives were 'easy' and on average lead to precise predictions with both methods—but with much higher variance (box widths) for SPE. However, for the one 'difficult' alternative, SPE (while still producing higher variance) was more robust: The worst three quarters of all predictions (left box edge) were at least 50% too high for CP, but only at least 12% too high for SPE.

The results for M1.2: *score* are shown in Table 2. We observe that the orderings created with *SPE* were by far most similar to the reference ordering R_{ref} with 13% wrong relations. Orderings created with *umlPSI* were worst, almost half of the relations between the design alternatives were specified wrong by the participants.

Method *PEM*	mean *score*	standard deviation of *score*
SPE	0.13	0.14
CP	0.32	0.26
umlPSI	0.48	0.36

Table 2 M1.2: Mean and variance for the *score* over all participants. Higher values indicate more contradictions to the reference ordering.

5 Experiment 2: Accuracy and Effort of Component-based Methods

5.1 Planning

5.1.1 Tasks

In this experiment, participants were asked to predict the performance for the Webserver system and the Mediastore system, as well as for five design alternatives.

The five design alternatives for the Mediastore were (*server2*) the allocation of two of the components to a second machine, (*pool*) the usage of a thread pool for database connections, (*dynlookup*) the usage of a broker for the component lookup, (*cache*) the introduction of a cache component that kept popular music files in memory, and (*bitrate*) the reduction of the bit rate of uploaded files to reduce the file sizes.

The five design alternatives for the Webserver were (*server2*), (*pool*), (*dynlookup*) and (*cache*) as described above as well as (*logging*) parallelisation of the Webserver's logging.

The required subtasks to perform the performance predictions are described in Section 3.4. The original task description and information on the two example systems can be found in (Martens, 2007, Section 4.2.2).

5.1.2 Hypotheses, Parameters, and Variables

For experiment 2, we use factor $PEM = \{SPE, PCM\}$ and independent variable design alternative a as described above. As an additional independent variable, participants evaluated two different tasks t describing two different systems. The levels of the task variable t were the Mediastore task (MS) and the Webserver task (WS). The levels of the design alternative variable a were $a_{MS} = \{original, server2, pool, dynlookup, cache, bitrate\}$ for the Mediastore and $a_{WS} = \{original, server2, pool, logging, dynlookup, cache\}$ for the Webserver. Table 3 summarises all goals, questions, metrics, hypotheses, and variables.

The first question to ask is Q2: *What is the accuracy of the analysed methods?* The validity of predictions compared to measurements for SPE had been studied in experiment 1. For PCM, it had been shown in (Becker et al., 2009, Sec. 10.2.) that predictions with correct models are close to the measured values. Thus, in the context of the second experiment, we compared the predictions of the participants with predictions made for a reference model created by ourselves. This excludes noise introduced by a comparison with real measurements. With the obtained actual mean response time *art*, we reused both metrics M1.1 *fracOfActual* and M1.2 *score* for this question. For metric M1.1, we will provide descriptive statistics. For metric M1.2, our hypothesis H2.1 was that applying PCM and SPE results in similarly good rankings: $score_{SPE} \approx score_{PCM}$

The second question to ask was Q3: *What is the effort of applying the analysed methods?* To assess the effort, we measured the time needed for the tasks as duration d in metric M3.1. The metric was a ratio scale with rational values > 0. Because the possible time sav-

Goal 2	**Compare Accuracy and Effort for SPE and PCM**
Q2	*What is the accuracy of the analysed methods?*
M1.1	*fracOfActual*: Predicted mean response time as a fraction of the actual mean response time, ratio scale with rational values > 0
M1.2	*score*: Normalised number of wrong relations in predicted ordering compared to reference ordering, ratio scale with rational values $[0,1]$
H2.1	$score_{SPE} \approx score_{PCM}$
Q3	*What is the effort of applying the analysed methods?*
M3.1	d: duration needed for the tasks, ratio scale with rational values > 0
H3.1	$d_{PCM} > d_{SPE}$
PEM	Factor: Used performance evaluation method with levels $\{SPE, PCM\}$
t	Independent variable: Task with levels $\{MS, WS\}$
a_{MS}	Independent variable: Design alternatives for task MS, with levels $\{original, server2, pool, dynlookup, cache, bitrate\}$
a_{WS}	Independent variable: Design alternatives for task WS, with levels $\{original, server2, pool, logging, dynlookup, cache\}$
accuracy	Dependent variable, measured with metric M1.1 and M1.2
effort	Dependent variable, measured with metric M3.1

Table 3 Summary of Goals, Questions, Metrics, Hypotheses, and Variables for Experiment 2

ings when reusable models are reused was not studied here, we expected that just creating the parametrised, reusable models and using them once takes longer. Our hypothesis H3.1 was that using PCM takes longer than using SPE: $d_{PCM} > d_{SPE}$.

5.1.3 Design

The experiment was designed as a one-factor-two-treatments changeover trial as depicted in Fig. 8 in the lower "Experiment" part. Each participant used both methods (intra-subject design), but with two different tasks. This allowed to collect more data points and further balanced potential differences in individual factors like skill and motivation between the two experiment groups. Using two different tasks reduced learning effects in the second session and lowered the influence of the concrete task. Additionally, using two tasks can increase both the internal validity (Prechelt, 2001, p.124) as well as the generalisability, which is most threatened by specific characteristics of the single experimental tasks (Prechelt, 2001, p.154).

Each session had a maximum time constraint of 4.5 hours. After making a prediction, the participants' solutions were checked for minimum quality by comparing the created models to the respective reference model. This acceptance test served to reduce the number of careless mistakes and included the comparison of the predicted response time with the predicted response time of the reference model as well as a check for the well-formedness of the models.

5.1.4 Procedure

The experiment took place in a university computer lab. Participants were not supposed to speak with others during the experiment session. Four adviser (members of our research group) were present to help with tool problems, the exercise, and the methods, as well as to check the solutions in the acceptance tests. To avoid influence on the duration and accuracy, the participants were asked to first try to solve problems on their own before consulting the advisors. To be able to assess a possible influence of this help, we documented all help and all rejections in the acceptance tests (Martens, 2007).

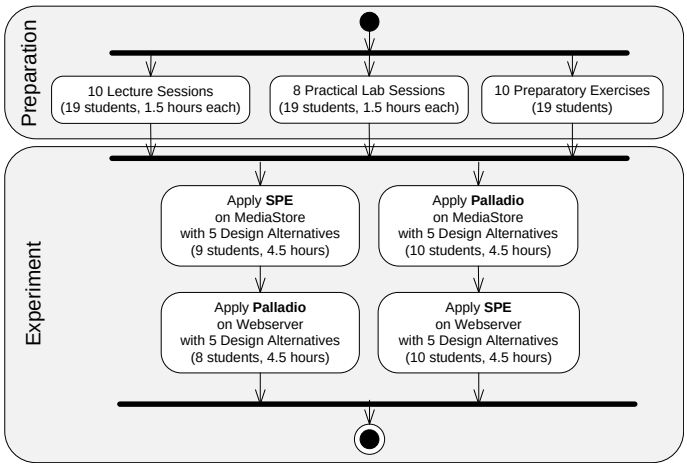


Fig. 8 Design of Experiment 2, visualised as an activity diagram. Nodes represent activities. Activities surrounded by bars represent parallel activity. The activities are grouped into the two phases of the experiment. Maximum time constraints are named for the experiment sessions, however, they were relaxed during the experiment (see Sec. 5.1.4).

The participants noted results and duration for the data collection. For each prediction, they were asked to document the duration (for metric M3.1) and they received a sheet to note timestamps in a predefined way. At the end of the session, they were asked to write down a ranking of all design alternatives (for metric M1.2: misplacements in ranking). Additionally, the created models were handed in that allowed us spot checks of whether the response times have been properly interpreted.

Not all participants finished the prediction for all design alternatives. Additionally, some participants' results could not be used and were dropped without distorting the results (3 in the first session, 2 in the second session, cf. (Martens, 2007, p.74)). One participant could not attend the second session due to personal reasons (cf. Fig. 8).

5.2 Analysis

5.2.1 Accuracy

For the deviation of the predicted mean response time to the reference mean response time (metric M1.1), we assessed the variability of the predictions using resampling as shown and explained in Fig. 9. The variability of both methods varied for different tasks t and design alternatives a. Thus, the achieved accuracy seemed to depend on the system and design alternative under study. There was no distinctive difference between the methods and we observe that most predictions with both methods were close to the reference. Overall, the accuracy of both methods was similar.

The results for M1.2: *score* are shown in Table 4. We observe that the orderings created both *SPE* and *PCM* were of similar quality, with slightly better results for *PCM*. However, due to the small sample size and small effect, there is no statistical significance.

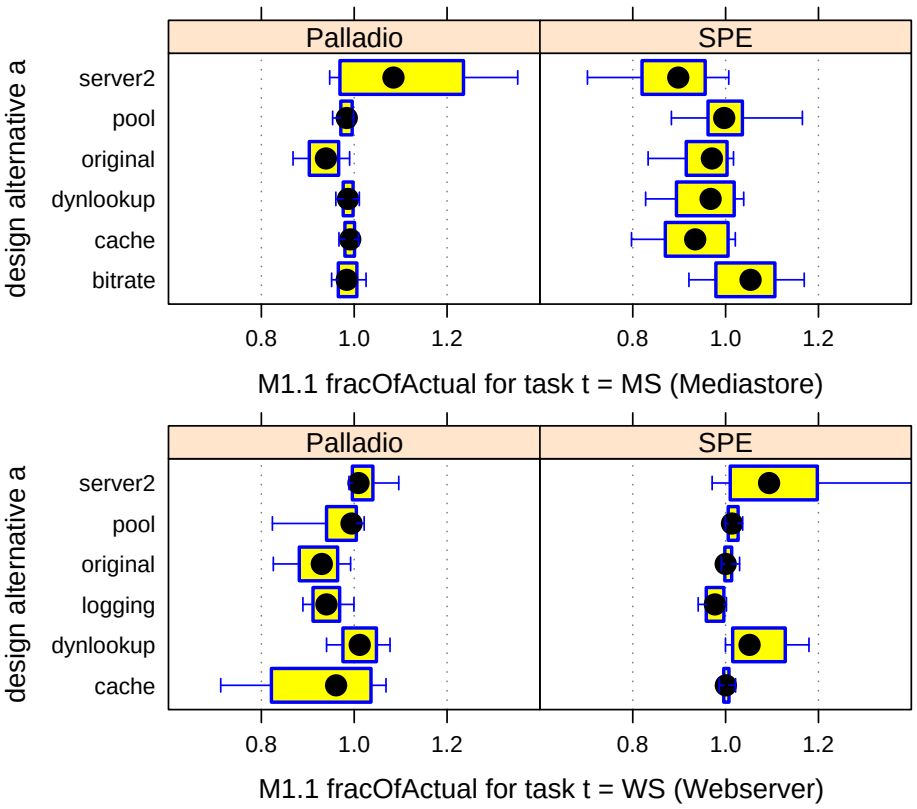


Fig. 9 Variability of the predictions (predicted response time) made with PCM and SPE. Each boxplot shows the distribution of 1000 resampled 3-pooled predictions p, each computed as the average of a random three subjects' predictions x_i, x_j, x_k (together: x) plus some normally distributed noise for smoothing: $p = \frac{1}{3}(\mathcal{N}(x_i, 0.5\sigma(x)) + \mathcal{N}(x_j, 0.5\sigma(x)) + \mathcal{N}(x_k, 0.5\sigma(x)))$. This amounts to simulating the variability of committee predictions when three people similar to our subjects produce each prediction together but without discussion. It assesses the variability and robustness of prediction for each method. The box marks the 25- to 75-percentile, the fat dot is the median, the whiskers mark the 10- and 90-percentile.

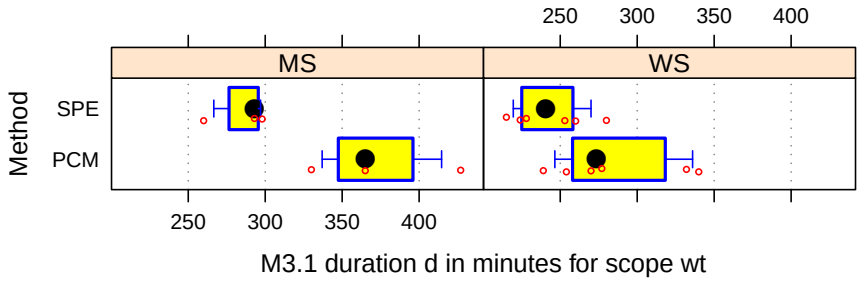
Method *PEM*	mean *score*	standard deviation of *score*
SPE	0.048	0.013
PCM	0.037	0.006

Table 4 M1.2: Mean and standard deviation for the *score* over all participants and all tasks $t \in \{MS, WS\}$. Higher values indicate more contradictions to the reference ordering.

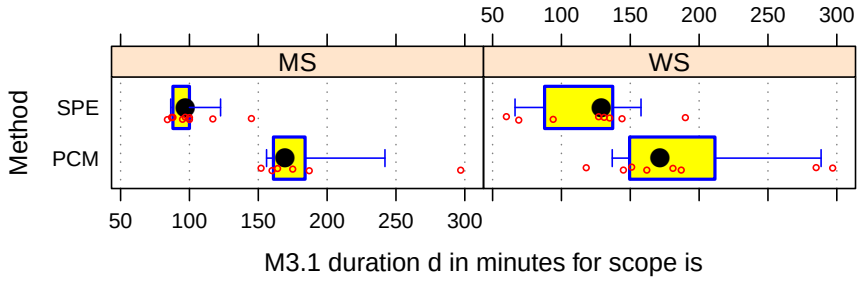
5.2.2 Effort

We evaluated metric M3.1 for the whole experiment task (=: scope *wt*), thus the duration d includes the duration of analysing the initial system and all design alternatives. Additionally, we distinguished by the task $t \in \{MS, WS\}$.

In neither session, all participants were able to finish the whole task *wt* within the extended time constraints, especially for *PCM*. To not favour one method in the comparison,



(a) for the whole task scope (*wt*). The number of evaluated results is $k_{PCM,MS} = k_{SPE,MS} = 3$ and $k_{PCM,WS} = k_{SPE,WS} = 6$.



(b) for the initial system scope (*is*) with $k_{PCM,MS} = 6$, $k_{SPE,MS} = 10$ and $k_{PCM,WS} = k_{SPE,WS} = 8$.

Fig. 10 Metric M3.1: Duration d in minutes for both methods and both systems of the k_t fastest participants. The boxes indicate the 25- and 75-percentile, the whiskers the 10- and 90-percentile, the small circles all single data points. The large dot inside the boxes is the median.

only the results of the k fastest participants from both groups were evaluated for metric M3.1, so that for both groups, the slower participants were left out.

Let $k_{a,t}$ be the number of participants who finished the whole task for one task $t \in \{MS, WS\}$ with one method $a \in PEM$. Then, we used the $k_t = max(k_{PCM,t}, k_{SPE,t})$ fastest participants results when comparing the results for t. Fig. 10(a) shows the results of metric M3.1 for the four combinations of methods and systems.

To get more data points, metric M3.1 was also evaluated for the analysis of the initial system only without design alternatives (=: scope *is*). Because all participants completed the initial system, we did not have to consider only the k fastest as before. Thus, we considered all participants (except the excluded ones mentioned in Section 5.1.4). Fig. 10(b) shows the resulting boxplot.

Table 5 shows the mean and standard deviations for metric M3.1 as well as the number of observations taken into account for all aforementioned combinations. Additionally, we compared how much longer it takes in average to make the PCM prediction compared to making the respective SPE prediction. These factors are shown as avd_{PCM}/avd_{SPE}. In average over both scopes, the duration for a PCM prediction was 1.4 times the duration for an SPE prediction.

	Whole task			Initial system		
	MS	WS	Both	MS	WS	Both
$mean_{PCM,t}$	374	285.33	314.89	189.17	190.75	190.07
$sd_{PCM,t}$	49.122	41.471	60.362	54.22	65.546	58.691
k	3	6	9	6	8	14
$mean_{SPE,t}$	283.67	243.33	256.78	101.44	118.75	109.59
$sd_{SPE,t}$	20.648	25.009	30.07	19.034	42.654	32.502
k	3	6	9	9	8	17
$\frac{mean_{PCM}}{mean_{SPE}}$	132%	117%	123%	186%	161%	173%

Table 5 Metric M3.1: Mean durations (*mean*) and standard deviations (*sd*) for making a prediction for the k fastest participants in minutes

For further analysis, we analyse both systems *MS* and *WS* together by normalising the effort values to a mean value of 100 for each system and scope. We analysed the confidence intervals for our hypothesis H3.1 $d_{PCM} \geq d_{SPE}$. Even though the number of observations k is small, the effect size is large enough to achieve significant results. With probability of 0.95, the mean effort to apply PCM is more than 110% of the mean effort to apply SPE for the whole experiment task (scope *wt*). For the initial system only without design alternatives (scope *is*), the mean effort to apply PCM is more than 144% of the mean effort to apply SPE with the same probability. An upper bound for the confidence interval could not be determined in both cases.

From the results, we also learn that using *PCM* takes a larger initial effort, as the time difference in scope *is* was larger. When studying different design alternatives, most of the system can be reused. As the lead of *SPE* was reduced when considering the whole task (scope *wt*), we suspected that the PCM models are indeed more suitable for reuse, even if this was not studied explicitly here. The next experiment with a specific design was needed to verify this hypothesis.

6 Experiment 3: Effort for Model Reuse

6.1 Planning

6.1.1 Tasks

The architecture modelling (cf. Section 3.4) was separated into two subtasks. First, the participants were asked to create 3 component performance models (for the `Cache`, `Shop`, and `Watermarking` component). In the second subtask, they assembled the newly created components together with other pre-existing components (`AudioDB`, `Billing`, `Encoding`, `Store`), and predicted the performance for the resulting system. The original task description and information on the example system can be found online at (Martens et al., 2009).

The disguise of the studied research question can reduce unwanted motivation bias (Prechelt, 2001). Example studies are (Briand et al., 1997; Prechelt et al., 2001). In our case, the participants were asked whether a certain performance requirement of 5 seconds response time could be fulfilled to disguise that their effort was the measured metric.

6.1.2 Hypotheses, Parameters, and Variables

For the third experiment, only *PCM* was applied. The factor "how is the model obtained" m distinguished the two cases: (1) create a component performance model and predict the

Goal 3	**Compare Effort for Reuse Case and Model Creation Case in PCM**
Q4	*What is the effort of the two cases "must model" ($m = 1$) and "reuse" ($m = 0$)?*
M4.1	$\hat{d}$: duration to obtain the performance model, ratio scale with rational values > 0
H4.1	$\hat{d}_{m=0} << \hat{d}_{m=1}$
Q5	*How does effort relate to interface complexity and inner complexity?*
H5.1	The additional effort for creating new models ($m = 1$) is explained by the total complexity $c + c_i$, whereas the duration of reuse ($m = 0$) depends only on the interface complexity c_i
m	Factor: Model origin with levels "must model" ($m = 1$) and "reuse" ($m = 0$)
c	Independent variable: Component internal complexity, ratio scale with integer values ≥ 0
c_i	Independent variable: Interface complexity, ratio scale with integer values ≥ 0
effort	Dependent variable, measured with metric M4.1

Table 6 Summary of Goals, Questions, Metrics, Hypotheses, and Variables for Experiment 3

performance ("must model" case: $m = 1$) and (2) reuse a component performance model and predict the performance ("reuse" case: $m = 0$). Additional independent variables were two complexity measures c and c_i. Table 6 summarises all goals, questions, metrics, hypotheses, and variables.

To operationalise the goal, we first ask Q4: *What is the effort of the two cases "must model" ($m = 1$) and "reuse" ($m = 0$)?* We measured the effort to prepare a performance prediction by creating or reusing a component performance model with metric M4.1: duration $\hat{d}$ to obtain the the performance model. We observed the metric for both levels of m on a ratio scale with rational values > 0.

Reusing a component must be considerably easier than recreating it in order for reuse to pay off eventually. Our qualitative hypothesis H4.1 was that the duration to obtain the estimation model is considerably less in the "reuse" case compared to the "must model" case for our example system: $\hat{d}_{m=0} << \hat{d}_{m=1}$.

The metric $\hat{d}$ was very specific for our components under study. In general, to achieve a lower effort for reuse, complexity has to be encapsulated inside the reused component: the interface complexity, dealt with when reusing, needs to be smaller than the inner complexity of the component. For more generalisability, we studied how the effort e was related to the interface complexity and the inner complexity of a component in both reuse and model creation case. We asked the Q5: *How does effort relate to interface complexity and inner complexity?*.

To measure the complexity of components, we defined the following additional independent variables for each of the services s offered by a component. The definitions were derived from our experience of complexity of PCM models. To illustrate our definition, Fig. 11 shows an example behaviour specification of the service *watermark*. We defined the following variables:

- $sp_s \in \mathbb{N}_0$: Number of parameters used in the behaviour description of s (e.g. 1 for `file.BYTESIZE` in service *watermark*).
- $a_s \in \mathbb{N}_0$: Number of actions in the behaviour description of s, without start and stop (e.g. 2 in service *watermark*).
- $lv_s \in \mathbb{N}_0$: Number of local variables used in the behaviour description of s (0 in service *watermark*).
- $cs_s \in \mathbb{N}_0$: Complexity of the parameter dependencies in the probabilistic annotations of actions and control flow of s (e.g. 1 in service *watermark* for the parameter dependency in the InternalAction: A single parameter is used in a linear dependency, which is a simple case).

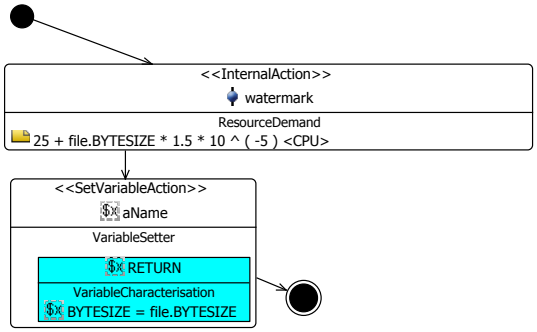


Fig. 11 The behaviour specification of the service *watermark* of the DigitalWatermarking component in the Mediastore system. In this example, the behaviour is a sequence of four actions. Start action and stop action mark the entry point and exit point of the service. The InternalAction denotes the usage of resources. In this example, the CPU is used. The amount of required CPU time has a constant factor (25), but also depends on the size of the *file* parameter in bytes (denoted as `file.BYTESIZE`). In the SetVariableAction, the return parameters of this service are characterised. In this case, the size of the file remains unchanged. Other available constructs used in this study are loops, branches, external calls, and local variable definitions.

Variable	AudioDB	Billing	Cache	Encoding	Shop	Store	Watermarking
c	7	2	7	14	8	9	3
c_i	5	1	4	4	2	5	2

Table 7 Complexity of the Components in Experiment 3

We defined the *interface complexity* of a component with n services as independent variable

$$\text{Interface complexity: } c_i = \sum_{s=1}^{n} sp_s$$

We defined the *inner complexity* of a component with n services as independent variable

$$\text{Inner complexity: } c = \sum_{s=1}^{n} a_s + lv_s + cs_s$$

Both variables were ratio scale with integer values ≥ 0. The complexity of the components is given in Table 7.

For successful reuse, we expected the relation between effort and complexity as follows: One the one hand, the effort for obtaining the estimation model for a given component K should correlate with K's total complexity $c+c_i$ if one must create the model ("must model": $m = 1$). On the other hand, the effort should correlate only with K's interface complexity c_i if one can reuse the model and needs not create it ($m = 0$). The level of m was expected to be the main explanatory factor for the effort variable.

Thus, our second hypothesis was H5.1: The additional effort for creating new models ($m = 1$) is explained by the total complexity $c + c_i$, whereas the duration of reuse ($m = 0$) depends only on the interface complexity c_i.

6.1.3 Design

As we expected to observe a large effect, we chose a simple experiment design with few participants and still expected significant results. We used an unbalanced paired comparison

design (Wohlin et al., 2000, p. 55) as depicted in Fig. 12 in the lower "Experiment" part. The fixed order (first create, then reuse) is required by the nature of the task, as the components cannot be used before they have been modelled. By having the participants create and reuse several component models, we obtained more data points for each factor level.

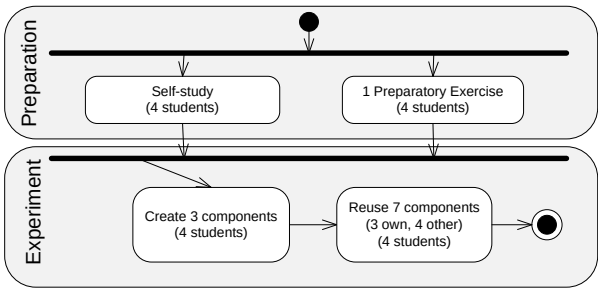


Fig. 12 Design of Experiment 3, visualised as an activity diagram. Nodes represent activities. Activities surrounded by bars represent parallel activity. The activities are grouped into the two phases of the experiment.

6.1.4 Procedure

The experiment took place in a university office. Each participant had a separate appointment. The participants' modelling was monitored by screen recording and the duration $\hat{d}$ was measured afterwards for each component.

One experimentator was present. After the participants had read the (sub-)task description, they were allowed to ask questions related to understanding the task description. After starting with the modelling, they were not supposed to ask any questions. However, some tool problems (crashes, bugs) required intervention of the experimentator. The time spent on problems could be determined from the recordings and was removed from the analysis results.

6.2 Analysis

The results of metric M4.1: Duration $\hat{d}$ is shown in Fig. 13. We see that the duration for the newly created components `Watermarking`, `Shop`, and `Cache` was much higher than for the other reused components. Thus, hypothesis H4.1 was clearly fulfilled.

For further analysis, for each subject and component, we represented the effort by the fraction of subject's s overall work time that he/she spent on component K : $e_{s,K} = t_{s,K}/t_{s,total}$. This way, we removed of the inter-subject working speed differences and focussed on the inter-component effort differences alone. We now normalized the complexity measure c by linearly scaling it such that its mean (over all data points in this experiment) is 1. We did the same for c_i. This way we made c and c_i directly comparable in the effort models we could then construct: We searched for linear models explaining e in terms of c, c_i, and m for all subjects and components. For these computations, we used the 'lm' and 'anova' procedures of R 2.8.1 (R Development Core Team, 2007).

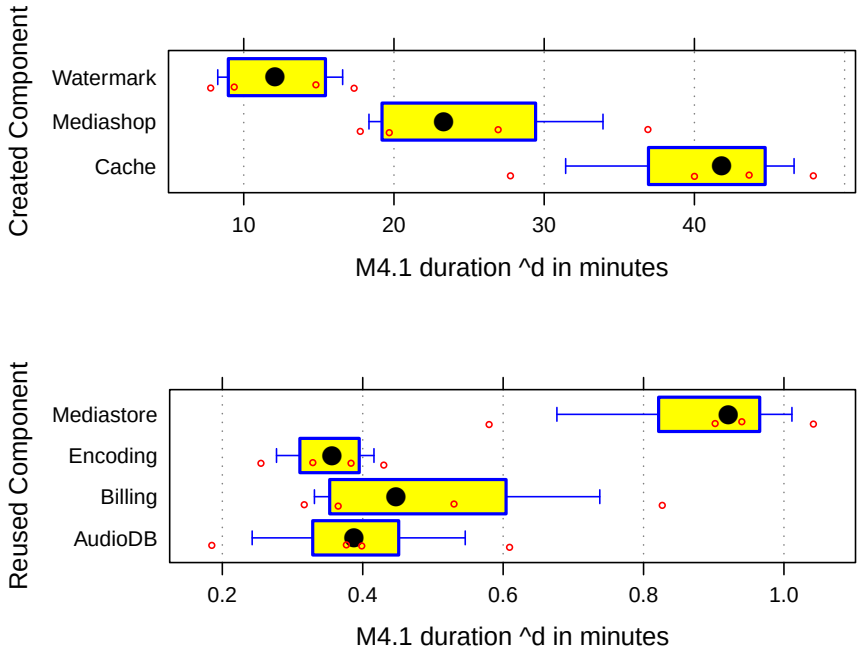


Fig. 13 M4.1: Duration $\hat{d}$ in minutes to obtain the performance model. The upper figure shows the duration for all newly created components (*mustmodel* = 1). The lower figure shows the duration for all reused components (*mustmodel* = 0). Note that the scale in the lower figure is enlarged by almost factor 50. The boxes indicate the 25- and 75-percentile, the whiskers the 10- and 90-percentile, the small circles all single data points; the large dot inside the boxes is the median. The number observations is 4 for each box.

A model providing a reasonably good fit (with low error and acceptable heteroscedasticity) was $\sqrt{e} = 0.729c_i + 5.438c \cdot m$ (see Fig. 14). Both coefficients were highly significant ($p < 0.0003$) and the model explained 95% of the variance, of which the first coefficient bore 38.5% and the second bore 56.5%.

The model indicates that interface complexity c_i was a relevant predictor of effort for the component model reuse case ($m = 0$) and, more importantly, that internal complexity c was only relevant for the component model creation case ($m = 1$). Our hypothesis H5.1 can be accepted (in the sense that the opposite is rejected).

The second coefficient was 7.5 times larger than the first. This shows that the relative influence of the internal complexity c was *much* stronger, which means that the amount of effort saved by reuse was large. Note that the ratio of 7.5 applies to the square root of effort. For effort itself, the ratio was about 20. However, in that case, the first coefficient was not statistically significant ($p = 0.3$), which was why we present the model for square root of effort instead.

7 Discussion

In the following, we discuss the results of the studies (Section 7.1) and the threats to validity (Section 7.2). We generalise the findings in section 7.3. In Section 7.4 we briefly present

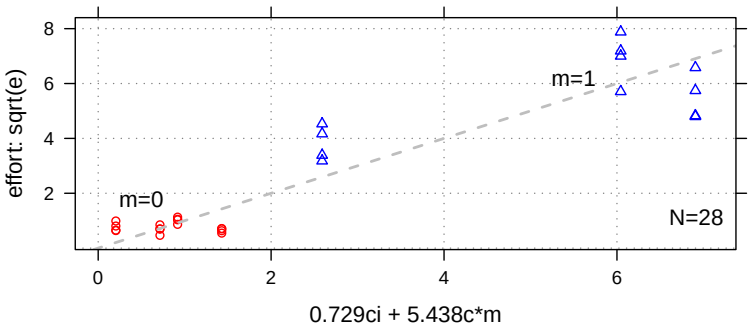


Fig. 14 Prediction quality of the model of reuse effort ($m = 0$ marked with circles) vs. must-model effort ($m = 1$ marked with triangles). The dashed line represents the linear model $\sqrt{e} = 0.729c_i + 5.438c \cdot m$.

future research in the area of component-based performance evaluation as well as discuss possible future studies in the area of model-based performance evaluation, also from an industry point of view.

7.1 Evaluation of Results and Implications

Initial interpretations of the results of each study separately have partially been given in the respective analysis sections. Here, we summarise the overall findings.

The main findings are

1. The accuracy achieved with umlPSI was noticeably worse than the other two methods SPE and CP. Using SPE or CP led to similar accuracy (cf. Fig. 7 and Table 2).
2. Using the component-based PCM or the monolithic SPE led to similar accuracy (cf. Fig. 9 and Table 4).
3. Creating reusable, parametrised models with PCM requires more effort than creating monolithic models with SPE (cf. Fig. 10(b)).
4. Reusing PCM models is much faster than creating them (cf. Fig. 13). Furthermore, we could identify that there is a significant relationship between total component complexity and the effort in the model creation case, whereas the effort only depends on the outer interface complexity in the reuse case (cf. Fig. 14).

The related studies presented in Section 2.4 have different research objective, thus their results cannot be readily compared to ours. However, our results do not contradict any of their findings, thus, they all fit in one overall theory. The comparison of SPE and CP can be related to the comparison of LQN and HYDRA in (Bacigalupo et al., 2005), because SPE and LQN are control-flow-oriented methods, whereas CP and HYDRA use measurement data. In both studies, the applicability of the two respective methods was similar, none was superior.

A discussion of inferences from this results can be found after the discussions of the threats to validity in Section 7.3.

7.2 Threats to Validity

In this section, we discuss the most important threats to validity that remain in our experiment designs. A more detailed discussion for the first two experiments can be found in (Koziolek and Firus, 2005; Martens et al., 2008b; Martens, 2007).

7.2.1 Internal Validity

The internal validity is concerned with the causal relationship between the treatment and the outcome. The observations should be caused by changes of the independent variable (e.g. the applied performance evaluation method) and not by other factors that have not been measured (Wohlin et al., 2000).

We have ensured this by randomized assignment of subjects to treatment groups in experiments 1 and 2, by a cross-over design to counter the learning effect in experiment 2, and by careful and supervised measurement. Some small deviations exist as mentioned in the 'Procedure' sections 4.1.4 and 5.1.4 above. Experiment 3 investigates only correlation, not causation. The quite reasonable quality of the linear model that describes this correlation can roughly be seen from Fig. 14.

7.2.2 Construct Validity

The construct validity is concerned with the relation of observation (treatment and outcome) and underlying theory (cause and effect in the real world) (Wohlin et al., 2000). The observed outcome (metrics for effort and accuracy) must be valid representatives for the studied effect (applicability), and the treatment (factors of independent variables) must be valid representatives for the studied cause (application of the methods in software development).

Levels of constructs: The measure for SEFF complexity used in experiment 3 is solely based on our experience and has not been previously validated. However, as the first idea worked so well without any trial-and-error, it appears that it is in fact valid, at least for the present study.

Experimenter bias: The authors were involved in the development of *PCM*, which could have biased experiment 2. However, we found no evidence of such bias in the setup (e.g. the quality of the method and tool instructions) or the resulting data. For instance, the participants complained equally often about the tools of both methods.

7.2.3 External Validity

The external validity states whether the results of an experiment are transferable to other settings than the specific experimental setting (Wohlin et al., 2000, p.65).

Here is the list of what we consider the most serious threats in this regard:

- The two systems (Webserver and Mediastore) investigated in our experiments are realistic from a performance assessment point of view (Smith and Williams, 2002), but can obviously not cover the space of possible architectures. It is conceivable that some differences between the methods might be different for different kinds of architectures, in particular very complex ones. However, effects observed for both systems probably generalize to a fair degree.
- We partially assumed that the timing behaviour of the software was already known.
- In the reuse scenario, no identification of possible reuse candidates was needed.

– The competence of our student subjects may have been unrealistically low (as they did not have long experience with the methods) or unrealistically high (as they were instructed carefully in the methods used). No substantial numbers of practitioners with deep experience in model-based performance evaluation exist so far, so it is unclear with what to compare our subjects.

7.3 Inferences

The finding that the umlPSI method led to worse accuracy supports the claim that tool support is crucial for empirical validations. At the same time, this poses a threat to the related studies reported in Section 2.4, because the authors themselves applied the methods in those cases. Our findings show that the generalisability to third-party users is not given for those studies. Inferences from the umlPSI results to the applicability of UML methods in general cannot be made. The unstable tool support is more likely the cause.

The third experiment investigated whether we can expect lower effort when reusing components. We did not only analyse this for one specific system, but also tried to generalise by showing that the effort can be explained on the interface complexity only. Thus, regardless of the inner complexity of a component, the models can be reused. This observations also seems to be generalisable to more realistic models which more inner complexity (as the inner complexity plays no role).

Even though we did not study the effect of reuse for monolithic methods, we can make the following observations. When reusing monolithic performance models, the internals of the models need to be adapted manually for every reuse. Thus, here the inner complexity must play a role in the effort or accuracy. Consequently, for well-encapsulated components, monolithic models are more difficult to reuse and component-based models will eventually pay off.

7.4 Future Work

Directions of future work include continuing development of the PCM, improved applicability of the approaches for industry, and future experiment in these areas.

7.4.1 Further Research for Component-based Performance Evaluation

Some open questions for the PCM method and component-based performance evaluation methods in general have arisen during our study. First of all, the required degree of parametrisation is unknown. For example, a component can be modelled with parameters that allow to specify the effects of the component container later. Alternatively, the component can be modelled depending on one specific component container (e.g. JBoss' EJB implementation). In this case, the component model might not be reusable in contexts with other component containers (e.g. Oracle's EJB implementation). A guideline is required to help deciding for a degree of parametrisation and reusability when creating the initial models. Especially for resource parametrisation, it has to be checked whether the right abstraction level is used in the meta models.

It is crucial to decrease the effort to create the parametrised models. On the one hand, model creation can be tool-supported: Reverse engineering implemented components has been suggested in (Kuperberg et al., 2008). Furthermore, automated test bed generation can

help to improve the parametrisation based on varying measurements. On the other hand, a stepwise refinement can reduce the need to create detailed parametrised models from the beginning. When a component performance model is reused in a different context, an automated calibration and validation of its initial parametrisation could be performed.

7.4.2 Industry Interests

For industrial application, it is relevant to cut down the effort for performance modelling as much as possible. More automation, such as automated measurements, automated calibration of models, etc., is required. With model-based method such as SPE, PCM, umlPSI, etc., users create high-level software models instead of low-level performance models of the system (cf. Fig. 1). Empirical studies can validate the claim that high-level models ease the modelling, and can help to spread the use of model-based methods in industry. Finally, the methods need to be accompanied by more systematic processes that structure the activities required for predictions and name patterns and best-practices. Applying the methods needs to evolve from an art to a repeatable engineering practice.

For component-based performance evaluation methods, it needs to be determined under which circumstances their use is beneficial. Product lines with systematic reuse could be used as case studies. As a second issue, legacy systems, that are not divided into components by design, need to be incorporated in the methods.

7.4.3 Interesting Research Questions and Proposed Experiment Set-ups

To evaluate the cost-benefit (type 3 validation) of methods, three different strategies to performance should be compared based on the cost they cause in the development process:

1. Model-based performance evaluation starting at design time: Measure the cost to create the models
2. Active performance testing, i.e. executing load tests as soon as implementations are available: Measure the cost for performance testing and the cost for fixing performance problems
3. Fix-it-later approach, i.e. only handling performance if problems occur in functional testing or even after installation at the customer site: Measure the cost for fixing performance problems.

Furthermore, the applicability of different abstraction levels for the created models should be compared based on achieved accuracy and effort. Abstraction levels are (1) high-level software models or (2) low-level performance models as explained in Section 2.1 (cf. Fig. 1).

The reliability of performance estimation of human users should be analysed with respect to the user's experience to determine which level of expertise is required for accurate estimations.

A relevant study for industry is to analyse the research questions in different contexts in order to determine the factors to choose one method. Possibly, LQNs could be beneficial for distributed systems. Stochastic Petri nets or queueing Petri nets could be best for systems with many synchronization and concurrency issues. PCM could be useful if varying allocation and varying usage profiles need to be studied.

For component-based methods, we should study which factors of the development context influence the break-even point for reusable models. A number of studies in different

contexts can lead to guidelines, which state under which circumstances component-based methods are useful.

Consequently, required setups for future experiments on performance evaluation methods include surveys with performance engineers from practise, experiments with practitioners, experiments in realistic project settings (possibly with students in a full term project), and long term studies (e.g., on performance knowledge engineering).

8 Conclusion

From the experiments reported here, we learned that component-based performance evaluation methods can indeed be more applicable, i.e. as accurate while saving effort, than monolithic models if component performance models are reused.

We reported on a series of three experiments studying the accuracy and effort when the performance evaluation methods SPE, CP, umlPSI, and PCM are applied by student participants. We found that applying SPE and CP produced good results, while applying umlPSI produced over-estimates. Comparing PCM and SPE, we found that they resulted in similar accuracy, while PCM required more effort to create the models from scratch. Finally, the effort for predicting performance with PCM was much reduced when models can be reused. In particular, our findings suggest that the effort for reusing PCM component models only depended on the comparably small interface complexity, not on the high internal complexity encapsulated inside each component. Thus, for well encapsulated components, effort can be saved when reusing the component models.

Future experiments should analyse larger designs in a more realistic development setting. Possibly, field studies could be done for a single method. Additional future work could also compare different component-based methods against each other to evaluate their specific benefits and deficits. Finally, the prerequisites under which component performance models can be successfully reused need to be characterised to support decisions to use a component-based performance evaluation method.

Only with applicable tools, the empirical validation of the entire method is possible. In the course of studies like presented here, the conceptional methods and the supporting tools are better understood, improved and made more applicable.

Acknowledgements We would like to thank Steffen Becker, Walter Tichy, Wilhelm Hasselbring, Viktoria Firus, Klaus Krogmann, and Michael Kuperberg for their valuable additions to this work. Furthermore, we thank all students who participated in our studies.

References

[Bacigalupo et al., 2004]Bacigalupo, D. A., Jarvis, S. A., He, L., and Nudd, G. R. (2004). An investigation into the application of different performance techniques to e-commerce applications. In *18th IEEE International Parallel and Distributed Processing Symposium 2004 (IPDPS'04), Santa Fe, New Mexico*. IEEE Computer Society Press.

[Bacigalupo et al., 2005]Bacigalupo, D. A., Jarvis, S. A., He, L., Spooner, D. P., Dillenberger, D. N., and Nudd, G. R. (2005). An investigation into the application of different performance prediction methods to distributed enterprise applications. *The Journal of Supercomputing*, 34(2):93–111.

[Bacigalupo et al., 2003]Bacigalupo, D. A., Turner, J. D., Jarvis, S. A., and Nudd, G. R. (2003). A dynamic predictive framework for e-business workload management. In *7th World Multiconference on Systemics,Cybernetics and Informatics (SCI2003) Performance of Web Services Invited Session, Orlando, Florida, USA*.

[Balsamo et al., 2004a]Balsamo, S., Di Marco, A., Inverardi, P., and Simeoni, M. (2004a). Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310.

[Balsamo et al., 2004b]Balsamo, S., Marzolla, M., Di Marco, A., and Inverardi, P. (2004b). Experimenting different software architectures performance techniques: A case study. In *Proceedings of the 4th International Workshop on Software and Performance*, pages 115–119. ACM Press.

[Basili et al., 1994]Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). The Goal Question Metric Approach. In Marciniak, J. J., editor, *Encyclopedia of Software Engineering - 2 Volume Set*, pages 528–532. John Wiley & Sons.

[Becker et al., 2006]Becker, S., Grunske, L., Mirandola, R., and Overhage, S. (2006). Performance Prediction of Component-Based Systems: A Survey from an Engineering Perspective. In Reussner, R., Stafford, J., and Szyperski, C., editors, *Architecting Systems with Trustworthy Components*, volume 3938 of *Lecture Notes in Computer Science*, pages 169–192. Springer-Verlag Berlin Heidelberg.

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

[Becker et al., 2007]Becker, S., Koziolek, H., and Reussner, R. H. (2007). Model-based Performance Prediction with the Palladio Component Model. In *WOSP '07: Proceedings of the 6th International Workshop on Software and Performance*, pages 54–65, New York, NY, USA. ACM Press.

[Böhme and Reussner, 2008]Böhme, R. and Reussner, R. (2008). *Dependability Metrics*, volume 4909 of *Lecture Notes in Computer Science*, chapter Validation of Predictions with Measurements, pages 7–13. Springer-Verlag Berlin Heidelberg.

[Bondarev et al., 2007]Bondarev, E., Chaudron, M. R. V., and de Kock, E. A. (2007). Exploring performance trade-offs of a JPEG decoder using the DeepCompass framework. In *WOSP '07: Proceedings of the 6th international workshop on Software and Performance*, pages 153–163, New York, NY, USA. ACM Press.

[Bondarev et al., 2004]Bondarev, E., Muskens, J., With, P. d., Chaudron, M., and Lukkien, J. (2004). Predicting real-time properties of component assemblies: A scenario-simulation approach. In *Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, pages 40–47, Washington, DC, USA. IEEE Computer Society.

[Briand et al., 1997]Briand, L. C., Bunse, C., and W.Daly, J. (1997). An experimental evaluation of quality guidelines on the maintainability of object-oriented design documents. In *7th Workshop on Empirical Studies of Programmers*, pages 1–19. ACM Press.

[Brosig et al., 2009]Brosig, F., Kounev, S., and Paclat, C. (2009). Using WebLogic Diagnostics Framework to Enable Performance Prediction for Java EE Applications. Oracle Technology Network (OTN) Article.

[Dujmović et al., 2004]Dujmović, J., Almeida, V., and Lea, D., editors (2004). *Proceedings of the 4th International Workshop on Software and Performance (WOSP'04)*, New York, NY, USA. ACM Press.

[Dumke et al., 2001]Dumke, R. R., Rautenstrauch, C., Schmietendorf, A., and Scholz, A., editors (2001). *Performance Engineering, State of the Art and Current Trends*, volume 2047 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany.

[Franks et al., 2009]Franks, G., Omari, T., Woodside, C. M., Das, O., and Derisavi, S. (2009). Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering*, 35(2):148–161.

[Heitmann and Moldt, 2007]Heitmann, F. and Moldt, D. (2007). Petri nets tool database.

[Hermanns et al., 2002]Hermanns, H., Herzog, U., and Katoen, J.-P. (2002). Process Algebra for Performance Evaluation. *Theoretical Computer Science*, 274(1–2):43–87.

[Huber et al., 2010]Huber, N., Becker, S., Rathfelder, C., Schweflinghaus, J., and Reussner, R. (2010). Performance Modeling in Industry: A Case Study on Storage Virtualization. In *ACM/IEEE 32nd International Conference on Software Engineering, Software Engineering in Practice Track, Capetown, South Africa*, pages 1–10, New York, NY, USA. ACM. Acceptance Rate: 23% (16/71).

[Jain, 1991]Jain, R. (1991). *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley.

[Jedlitschka et al., 2008]Jedlitschka, A., Ciolkowski, M., and Pfahl, D. (2008). *Guide to Advanced Empirical Software Engineering*, chapter Reporting Experiments in Software Engineering, pages 201–228. Springer, London, UK.

[Kounev, 2006]Kounev, S. (2006). Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7):486–502.

[Koziolek, 2004a]Koziolek, H. (2004a). Empirical Evaluation of Performance-Analysis methods for Software Architectures. http://sdqweb.ipd.kit.edu/publications/pdfs/koziolek2004b.pdf. Partial English translation of the original Master's thesis "Empirische Bewertung von Performance-Analyseverfahren für Software-Architekturen", Universität Oldenburg.

[Koziolek, 2004b]Koziolek, H. (2004b). Empirische Bewertung von Performance-Analyseverfahren für Software-Architekturen. Master's thesis, Universität Oldenburg.

[Koziolek, 2010]Koziolek, H. (2010). Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634 – 658. Special Issue on Software and Performance.

[Koziolek et al., 2008]Koziolek, H., Becker, S., Happe, J., and Reussner, R. (2008). *Model-Driven Software Development: Integrating Quality Assurance*, chapter Evaluating Performance of Software Architecture Models with the Palladio Component Model, pages 95–118. IDEA Group Inc.

[Koziolek and Firus, 2005]Koziolek, H. and Firus, V. (2005). Empirical Evaluation of Model-based Performance Predictions Methods in Software Development. In Reussner, R. H., Mayer, J., Stafford, J. A., Overhage, S., Becker, S., and Schroeder, P. J., editors, *Proceeding of the first International Conference on the Quality of Software Architectures (QoSA'05)*, volume 3712 of *Lecture Notes in Computer Science*, pages 188–202. Springer-Verlag Berlin Heidelberg.

[Kuperberg et al., 2008]Kuperberg, M., Krogmann, K., and Reussner, R. (2008). Performance Prediction for Black-Box Components using Reengineered Parametric Behaviour Models. In *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE'08), Karlsruhe, Germany, 14th-17th October 2008*, volume 5282 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag Berlin Heidelberg.

[Lazowska et al., 1984]Lazowska, E., Zahorjan, J., Graham, G. S., and Sevcik, K. C. (1984). *Quantitative System Performance - Computer System Analysis Using Queueing Network Models*. Prentice-Hall.

[Liu et al., 2005]Liu, Y., Fekete, A., and Gorton, I. (2005). Design-Level Performance Prediction of Component-Based Applications. *IEEE Transactions on Software Engineering*, 31(11):928–941.

[Martens, 2007]Martens, A. (2007). Empirical Validation of the Model-driven Performance Prediction Approach Palladio. Master's thesis, Carl-von-Ossietzky Universität Oldenburg.

[Martens et al., 2008a]Martens, A., Becker, S., Koziolek, H., and Reussner, R. (2008a). An Empirical Investigation of the Applicability of a Component-Based Performance Prediction Method. In *Proceedings of the 5th European Performance Engineering Workshop (EPEW'08), Palma de Mallorca, Spain*, volume 5261 of *Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag Berlin Heidelberg.

[Martens et al., 2008b]Martens, A., Becker, S., Koziolek, H., and Reussner, R. (2008b). An Empirical Investigation of the Effort of Creating Reusable Models for Performance Prediction. In *Proceedings of the 11th International Symposium on Component-Based Software Engineering (CBSE'08), Karlsruhe, Germany*, volume 5282 of *Lecture Notes in Computer Science*, pages 16–31. Springer-Verlag Berlin Heidelberg.

[Martens et al., 2009]Martens, A., Koziolek, H., Prechelt, L., and Reussner, R. (2009). Experiment 3: Effort for creating and reusing PCM performance models. http://sdqweb.ipd.kit.edu/wiki/Mobppexp/reuse_experiment.

[Marzolla, 2004]Marzolla, M. (February 2004). *Simulation-Based Performance Modeling of UML Software Architectures*. PhD Thesis TD-2004-1, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Mestre, Italy.

[Medvidovic and Taylor, 1997]Medvidovic, N. and Taylor, R. N. (1997). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26:70–93.

[Menasce et al., 1994]Menasce, D., Almeida, V., and Dowdy, L. (1994). *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems*. Prentice-Hall, New Jersey.

[Menascé and Almeida, 2000]Menascé, D. A. and Almeida, V. A. F. (2000). *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall, Englewood Cliffs, NJ, USA.

[Menascé et al., 2004]Menascé, D. A., Almeida, V. A. F., and Dowdy, L. W. (2004). *Performance by Design*. Prentice Hall.

[Object Management Group (OMG), 2005]Object Management Group (OMG) (2005). UML Profile for Schedulability, Performance and Time (SPT).

[Object Management Group (OMG), 2006]Object Management Group (OMG) (2006). UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP (realtime/05-02-06).

[Prechelt, 2001]Prechelt, L. (2001). *Kontrollierte Experimente in der Softwaretechnik*. Springer-Verlag, Berlin, Germany.

[Prechelt et al., 2001]Prechelt, L., Unger, B., Tichy, W. F., and Votta, L. G. (2001). A controlled experiment in maintenance comparing design patterns to simpler solutions. *IEEE Transactions on Software Engineering*, 27:1134.

[R Development Core Team, 2007]R Development Core Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, Last retrieved 2008-01-06.

[Rolia and Sevcik, 1995]Rolia, J. A. and Sevcik, K. C. (1995). The Method of Layers. *IEEE Transactions on Software Engineering*, 21(8):689–700.

[Smith and Williams, 2002]Smith, C. U. and Williams, L. G. (2002). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley.

[Williams and Smith, 2002]Williams, L. G. and Smith, C. U. (2002). PASASM: a method for the performance assessment of software architectures. In *Proceedings of the 3rd International Workshop on Software and Performance (WOSP'02)*, pages 179–189, New York, NY, USA. ACM.

[Williams and Smith, 2003]Williams, L. G. and Smith, C. U. (2003). Making the Business Case for Software Performance Engineering. In *Proceedings of the 29th International Computer Measurement Group Conference, December 7-12, 2003, Dallas, Texas, USA*, pages 349–358. Computer Measurement Group.

[Wohlin et al., 2000]Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: an Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.