An Empirical Investigation of the Effort of Creating Reusable, Component-Based Models for Performance Prediction

Anne Martens[1], Steffen Becker[2], Heiko Koziolek[3], and Ralf Reussner[1]

[1]Chair for Software Design and Quality
Am Fasanengarten 5, University of Karlsruhe (TH), 76131 Karlsruhe, Germany
[2]FZI Forschungszentrum Informatik, Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany
[3]ABB Corporate Research, Wallstadter Str. 59, 68526 Ladenburg, Germany
{martens,sbecker,koziolek,reussner}@ipd.uka.de

Abstract. Model-based performance prediction methods aim at evaluating the expected response time, throughput, and resource utilisation of a software system at design time, before implementation. Existing performance prediction methods use monolithic, throw-away prediction models or component-based, reusable prediction models. While it is intuitively clear that the development of reusable models requires more effort, the actual higher amount of effort has not been quantified or analysed systematically yet. To study the effort, we conducted a controlled experiment with 19 computer science students who predicted the performance of two example systems applying an established, monolithic method (Software Performance Engineering) as well as our own component-based method (Palladio). The results show that the effort of model creation with Palladio is approximately 1.25 times higher than with SPE in our experimental setting, with the resulting models having comparable prediction accuracy. Therefore, in some cases, the creation of reusable prediction models can already be justified, if they are reused at least once.

Keywords: Performance Prediction, Empirical Study, Controlled Experiment.

1 Introduction

As current applications always ask for maximum performance, performance problems are continuously prevalent in many software systems [20]. Model-based prediction methods [1] try to tackle these problems during early design phases to avoid the problem of implementing architectures which are not able to fulfil certain performance goals. They counter the still popular ”fix-it-later” attitude towards performance problems. Many of these methods use designer-friendly UML-based models for software developers, and transform them into formal models (e.g., queueing networks, stochastic Petri-nets, stochastic process algebras), from which performance measures (e.g., response times, throughput) can be derived analytically or via simulation.

During the last decade, researchers have proposed several monolithic prediction approaches (such as SPE [20], uml2LQN [15], umlPSI [2], survey in [1]) and several component-based (CB) prediction approaches (such as CB-SPE [7], ROBOCOP [8], and Palladio [6], survey in [5]). CB approaches try to leverage the benefits of componentry in the sense of Szyperski [21] by reusing well-documented component specifications. This is of particular interest for performance prediction methods, as CB software designs limit the degree of freedom for implementation by (at least partially) reusing existing components. This can also lead to higher performance prediction accuracy. In addition, reusable component prediction models can be composed isomorphically to the software architecture, thereby lowering the effort of performance modelling.

Palladio features highly parametrised component performance specifications, which are better suited for reuse than those of other approaches, because they include more context dependencies (i.e., dependencies to external service calls, usage profile, resource environment). The effort for creating such parametrised, CB models is naturally higher than for throw-away models. However, until now this higher effort has not been investigated systematically. Therefore, it is an open question when it is justified.

Based on this observation, we conducted a controlled experiment comparing the effort of applying SPE (as an example for a method with throw-away models) and Palladio (as an example for a method with reusable models). In this paper, we present the results for the following question: (Q1) What is the duration of modelling and predicting with both methods? As we wanted to assess the effort of applying the methods without bias, we let 19 computer science students apply the methods in an experimental setting. They analysed two CB software systems and assessed the performance impact of additional five design alternatives (e.g., introducing caches, replication, etc.). By using tools accompanying the methods (SPE-ED and PCM-Bench), they predicted response times for two different usage profiles. Therefore we assessed the effort for the combination of applying the method and the corresponding tools.

Our results show that modelling the whole task (that is the initial system and five additional design alternatives) took in average 1.25 times longer with Palladio than with SPE. Interestingly, modelling only the initial architecture took in average 1.81 times longer. The students spent most of the time modelling the control flow and debugging their models, to make them valid for the analyses tools.

In a second paper [13], we further analysed the accuracy of the predictions achieved by the students compared to a sample solution. Additionally, we searched for reasons for the achieved prediction accuracy by analysing the models created during the experiment and evaluating questionnaires filled out by the participants after the experiment. For reasons of self-containedness, sections 2.3, 3.2 - 3.4, 5.1 and 6 are similar in both papers, as they describe and discuss the common experiment setting.

The contributions of this papers are (i) the design of an experimental setting for comparing performance prediction methods, allowing the replication of the study, and (ii) a first quantification of the effort required to produce reusable prediction models.

This paper is organised as follows. Section 2 presents the basics of model-driven performance prediction and briefly introduces SPE and Palladio. Afterwards, Section 3 explains the experimental design, before Section 4 illustrates the results. Section 5 discusses the validity of the empirical study and provides potential explanations for the results. Related work is summarised by Section 6, while Section 7 concludes the paper and sketches future work.

2 Model-Driven Performance Prediction

2.1 Background

Several model-driven performance prediction approaches have been proposed [1], all of which follow a similar process model (Fig. 1). First, developers annotate plain software design models (e.g., UML models) with estimated or already measured performance properties, such as the execution time for an activity or the number of users concurrently issuing requests.

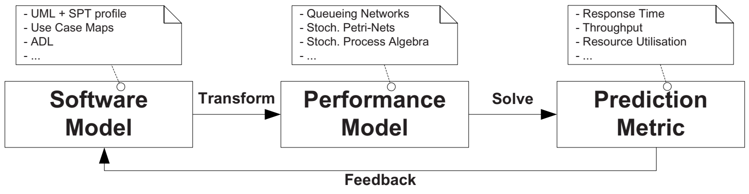


Fig. 1. Performance Prediction Process

Second, model transformations automatically convert the annotated software models into established performance formalisms such as queueing networks (QN), stochastic Petri nets (SPN), or stochastic process algebras (SPA). Existing analytical or simulation-based solution techniques then automatically derive and report performance measures, such as response times for specific use cases, maximum throughputs, or the utilisation of resources, which is crucial for identifying performance bottlenecks. Developers compare the predicted results to their requirements and decide whether to change their design or to start implementation. Only a few approaches implement an automated feedback of the prediction results into the software design model.

For our experiment, we compared our component-based Palladio method [6] with the mature, monolithic Software Performance Engineering (SPE) method [20]. We chose SPE as it has been applied in practice and provides a reasonably usable tool support, unlike many other approaches [11] solely proposed by academics. The following two sections briefly describe the two approaches, which both follow the process model sketched above.

2.2 SPE

The SPE method was the first elaborated, practically applicable comprehensive approach for early, design-time performance prediction for software systems [19]. SPE primarily targets software architects and performance analysts during early development stages. They identify key scenarios (i.e., use cases critical to the overall system performance) and set performance goals for the scenarios (e.g., max. response time) based on the requirements.

Afterwards, developers use a software execution model (Execution Graph, EG) to describe steps within such a performance-critical scenario. EGs are similar to UML activity diagrams and allow annotating each step with resource requirements, for example the number of needed CPU instructions.

With a so-called overhead matrix, software resource requirements in EGs (e.g., a database access) can be mapped to system resources (e.g., 10 ms for a hard disk access per database access). Several scenarios and the corresponding user arrival rates on different machines can be combined to form a system execution model.

EGs do not necessarily reflect actual componentisation of a system, but provide an abstraction of the most performance-relevant steps in a scenario. This is useful for conducting performance analyses as early as possible during the life-cycle of a system, when many details are still unknown. It also limits the developers' effort for initial modelling. However, dependencies on the specific project context are not made explicit, but are mixed with component specifics. Thus, it is usually not possible to readily reuse the resulting performance models when reusing the software components. Additionally, the models cannot be used for model-driven development, as their performance-related abstraction does not provide enough information for other purposes like code generation.

The SPE methodology has been applied in industrial settings. Several anonymised case studies are provided in [20].

2.3 Palladio Component Model

The Palladio Component Model (PCM) [6] is a meta-model for specifying and analysing component-based software architectures with focus on performance prediction.

This meta-model is divided among the separate developer roles of a component-based development process: The component developer produces independent, reusable component specifications. The other roles (software architects, system deployers, domain experts and quality-of-service analysts) provide information on the project-specific context, such as binding of the components, their allocation to hardware and their usage. The meta-model provides each role with a domain-specific language suited to capture their specific knowledge [6].

To support the creation of reusable component performance models, the component specifications are parametrised by influence factors whose later values are unknown to the component developer. In particular, these are the performance measures of external service calls, which depend on the actual binding of the component's required interfaces (provided by the software architect), the actual resource demands which depend on the allocation of the components to hardware resources (provided by the system deployer), and performance-relevant parameters of service calls (provided by the domain expert).

The parametric behavioural specification used in the PCM as part of the software model is the *Resource Demanding Service Effect Specification* (RD-SEFF) which is a control and data flow abstraction of single component services, also similar to UML activity diagrams. It specifies control flow constructs like loops, or branches only if they affect external service calls. Additionally, they abstract component internal computations in so called *internal actions* which only contain the resource demand (e.g. reading 100 Bytes from a hard drive) of the action but not its concrete behaviour. Calling services and parameter passing are specified using *external call actions*, which only refer to the component's required interfaces to stay independent of the component binding. Hence, unlike EGs, RD-SEFFs reflect the componentisation of the system and allow to create component specifications that can be reused in other project contexts. In this

Table 1. GQM plan overview

Goal	**Empirically investigate the effort to create and analyse performance prediction models using Palladio and SPE**	
Question 1	What is the duration of predicting the performance?	
Metric 1.1	Average duration of a prediction	$avd_a = avg(\{d_p \,\vert p \in P_a\})$
Metric 1.2	Break down of the duration	$avdact_{a,i} = norm_{d_a}(avg(\{dact_{i,p} \,\vert p \in P_a\}))$
Hypothesis 1.1	A Palladio prediction needs 1.5 as long as an SPE prediction	$avd_{Pal} = 1.5 \cdot avd_{SPE}$
Hypothesis 1.2	For both approaches, the largest time fraction is needed to model the system	

experiment, we thus measure the additional effort required to reflect the componentisation in the Palladio models (in contrast to the SPE models).

3 Empirical Investigation

For the empirical investigation, we formulated a goal, one question and derived metrics using the Goal-Question-Metric approach [4]. The goal of this work is:

> **Goal**: Empirically investigate the effort to create and analyse performance prediction models using Palladio and SPE.

For each metric, hypotheses were formulated to support the evaluation of the metrics and answering the question. The same metrics can also be used when repeating this experiment. Details are presented in section 3.1.

We conducted the investigation as a controlled experiment. Section 3.2 presents the experiment's design, section 3.3 describes the preparation of the participants. The tasks and the experiment execution are presented in section 3.4 and 3.5, respectively.

3.1 Questions and Metrics

For each metric, we have formulated hypotheses to support the evaluation of the metrics and answer the question. After an informal explanation, we give a formal description for the metrics. Table 1 summarises goal, question, metrics, and hypotheses.

Q1: What is the duration of predicting the performance? To evaluate the effort for making a prediction, we looked at the time needed, i.e. the duration, because time (in terms of person-days) is the major factor of effort and costs. For an empirical study of the effort of any software development technique, it is inevitable to include the used tools. Thus, here we measured the effort for the combination of applying the method (SPE and Palladio) and the corresponding tools (SPE-ED and PCM-Bench).

Metric 1.1 is the average duration of making a performance prediction. The duration includes reading the specification (ra), modelling the control flow (cf), adding resource demands (rd), modelling the resource environment (re), modelling the usage profile (up), searching for errors (err) and analysing (ana). Metric 1.2 breaks down the overall duration into the duration of the different activities of a performance prediction mentioned above.

Our hypothesis 1.1 was that a Palladio prediction needs 1.5 times as long as an SPE prediction. We based this hypothesis on experience from the field of code reuse cost models, where a median relative cost of writing for reuse of 1.5 over several studies was detected by [16, p.29], with a standard deviation of 0.24. Furthermore, hypothesis 1.2 is that the entire modelling, including cf, rd, re, and up, is the largest fraction of the duration with both approaches, which should be the case as the analysis is automated. Still, as the tools are not equally matured and Palladio uses simulation, which takes more time than SPE's analytical solution, the hypothesis is not beyond doubt. Additionally, we did not know whether the results can be readily interpreted by the users, and we wanted to check this assumption.

In the following, the metrics are defined formally. Each variable is defined only once and keeps that definition throughout this work. Let $A = \{SPE, Pal\}$ be the approaches under study. With $a \in A$, let P_a be the set of participants applying approach a.

Metric 1.1: For each participant $p \in P_a$, the duration d_p of making a performance prediction is measured. The duration is averaged over all participants. To do so, the function avg is defined as the arithmetic mean of a set of real values.

Metric 1.1: For $a \in A : avd_a = avg(\{d_p \,|p \in P_a\})$

Metric 1.2: Let $Act = \{ra, cf, rd, re, up, err, ana\}$ be the set of different performance prediction activities mentioned above. We measured the duration of each of the single steps $i \in Act$ for each participant $p \in P_a$ and named it $dact_{i,p}$. We averaged it over all participants and normalised it, i.e. gave it as a percentage of the overall duration avd_a.

Metric 1.2: For $i \in Act, a \in A$:
$avdact_{a,i} = norm_{d_a}(avg(\{dact_{i,p} \,|p \in P_a\}))$

3.2 Experiment Design

The study was conducted as a controlled experiment and investigated the effort with participants who are not the developers of the approaches. The participants of this study were students of a master's level course (see section 5.1 for the discussion of student subjects). In an experiment, it is desirable to trace back the observations to changes of one or more independent variables. Therefore, all other variables influencing the results need to be controlled. The *independent variable* in this study was the approach used to make the predictions. Observed *dependent variables* were the duration of making a prediction and the quality of the prediction to ensure a minimum quality.

The experiment was designed as a changeover trial as depicted in figure 2. The participants were divided into two groups, each applying an approach to a given task. In a second session, the groups applied the other approach to a new task. Thus, each participant worked on two tasks in the course of the experiment (inter-subject design) and used both approaches. This allowed to collect more data points and balanced potential differences in individual factors such as skill and motivation between the two experiment groups. Additionally, using two tasks lowered the concrete task's influence and

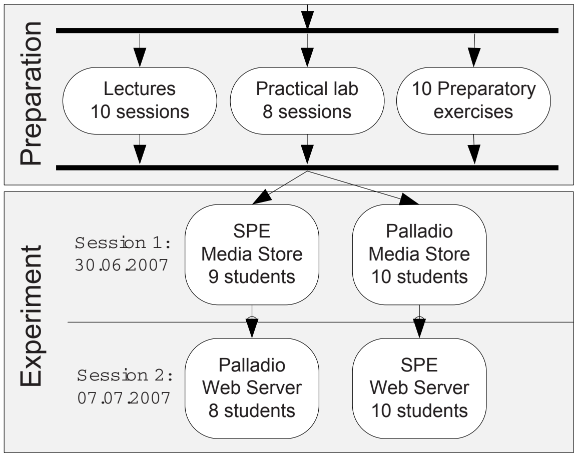


Fig. 2. Experiment design

increased the generalisability. We balanced the grouping of the participants based on the results in the preparatory exercises: We divided the more successful half randomly into the two groups, as well as the less successful half, to ensure that the groups were equally well skilled for the tasks. We chose not to use a counter-balanced experiment design, as we would need to further divide the groups, which would disturb the balancing between the groups. We expected a higher threat to validity from the individual participant's performance than from sequencing effects.

Before handing in, the participants' solutions were checked for minimum quality by comparing the created models to the respective reference model. This acceptance test included the comparison of the predicted response time with the reference model's predicted response time as well as a check for the models' well-formedness.

3.3 Student Teaching

The 19 computer science students participating in the experiment were trained in applying SPE and Palladio during a one-semester course covering both theory and practical labs. For the theory part, there was a total of ten lectures, each of them took 1.5h. The first three lectures were dedicated to foundations of performance prediction and CBSE. Then, two lectures introduced SPE followed by five lectures on Palladio. The three additional lectures on Palladio in comparison to SPE were due to its more complex meta-model which allows for reusable prediction models. Note, that this also shows that reusable models require more training effort. In parallel to the lectures, eight practical labs took place, again, each taking 1.5h. During these sessions, solutions to the accompanying ten exercises were presented and discussed. Five of these exercises practised SPE and five Palladio.

The exercises had to be solved by the participants as homework. We assigned pairs of students to each exercise and shuffled frequently to get different combinations of students work together and exchange knowledge. This was assumed to lower the influence of individual performance in the experiment. Each exercise took the students 4.75h in average to complete.

Overall, the preparation phase was intended to ensure a certain level of familiarity with the tools and concepts, because participants who failed two preparatory exercises or an intermediate short test were excluded from the experiment.

3.4 Experiment Tasks

To be applicable for both SPE and Palladio, the experiment tasks can only contain aspects that can be realised with both approaches. For example, the tasks cannot make use of the separate roles of Palladio, because these roles are not supported by SPE. Thus, each participant needs to fulfil all roles.

For reasons of compatibility, both experiment tasks had similar set-ups. The task descriptions contained component and sequence diagrams documenting the static and dynamic architecture of a CB system. The sequence diagrams also contained performance annotations. The resource environment with servers and their performance properties was documented textually. The detailed task description is available on-line in [12]. For each system, two usage profiles were given, to reflect both a single-user scenario (*UP1*) and a multi-user scenario leading to contention effects (*UP2*). Additionally, they differed in other performance relevant parameters (see below).

In addition to the initial system, several design alternatives were evaluated. This reflects a common task in software engineering. Four design alternatives were designed to improve the system's performance, and the participants were asked to evaluate which alternative is the most useful one. Three of these alternatives implied the creation of a new component, one changed the allocation of the components and the resource environment by introducing a second machine. With the final fifth alternative, the impact of a change of the component container, namely the introduction of a broker for component lookups, on the performance should be evaluated.

The two systems were prototypical systems specifically designed for this experiment. In the first session, a performance prediction for a web-based system called Media Store was conducted. This system stores music files in a database. Users can either upload or download sets of files. The size of the music files and the number of files to be downloaded are performance-relevant parameters. The five design alternatives were the introduction of a cache component that kept popular music files in memory, the usage of a thread pool for database connections, the allocation of two of the components to a second machine, the reduction of the bit rate of uploaded files to reduce the file sizes and the aforementioned usage of a broker.

In the second session, a prototypical Web Server system was examined. Here, only one use case was given, a request of an HTML page with further requests of potential embedded multimedia content. Performance-relevant parameters were the number of multimedia objects per page, the size of the content and the proportion of static and dynamic content. The five design alternatives were the introduction of a cache component, the aforementioned usage of a broker, the parallelisation of the Web Server's logging, the allocation of two of the components on a second machine and the usage of a thread pool within the Web Server.

The participants using the Palladio approach were provided with the initial repository of available components without RD-SEFFs. It made the tasks for SPE and Palladio more comparable, because the participants still had to create the RD-SEFFs with the performance annotations, which is similar to the creation of an EG in SPE.

3.5 Experiment Execution

The group of 19 computer science students was divided into two groups as shown in figure 2. We conducted two sessions, each with a maximum time constraint of 4.5 hours. One participant did not attend the second session due to personal reasons, thus, only 18 students took part. The participants were asked to document the duration of the activities given in metric 1.2 and to fill in a questionnaire with qualitative questions at the end of the session.

Four members of our chair were present to help with tool problems, the exercise, and the methods, as well as to check the solutions in the acceptance tests. This might have distorted the results, because they might have influenced the duration. The more problems were solved by the experimentators, the less time the participants might have spent on solving them themselves. To avoid this effect, the participants were asked to first try to solve problems on their own before consulting the experimentators. To be able to assess a possible influence of this help, we documented all help and all rejections in the acceptance tests [12].

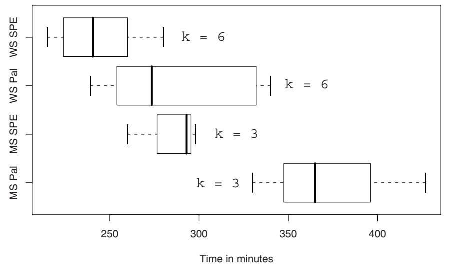
Because many participants did not finish the task within 4.5 hours in both sessions, the time restriction was loosened afterwards and they were allowed to work another 2.5 hours (session 1) and 2 hours (session 2). In both sessions, three (session 1) respectively two (session 2) participants were not properly prepared, as they needed a lot of basic help or were not able to finish even the initial system prediction. Thus, the results of these three / two participants could not be used. All other participants modelled the initial system and at least one design alternative. Because two participants failed using both approaches, omitting their results does not advantage one of the approaches. Additionally, the time constraints did not distort the results for the initial system prediction, because every remaining participant finished the initial prediction well before the end of the experiment.

Overall, in session 1, three of the remaining seven participants using Palladio and seven of the nine participants using SPE were able to finish all design alternatives. In session 2, the eight participants using SPE finished all design alternatives, as well as six of the eight participants using Palladio. The acceptance test ensured that the created models were meaningful. As a result, the average deviation of the predicted response time from a reference solution was only about 10%.

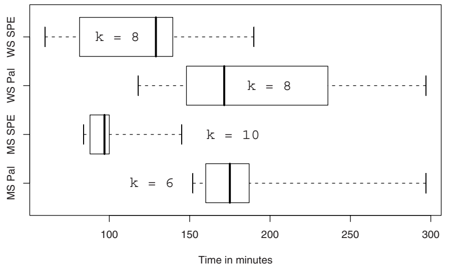
4 Results

4.1 Metric 1.1: Average Duration of Making a Prediction

First, we evaluated metric 1.1 for the whole experiment task (=: scope wt), thus the duration d_p includes the duration of analysing the initial system and all design alternatives. In neither session, all participants were able to finish the respective task within the extended time constraints, especially for Palladio. We first looked at those participants who finished the whole task with one approach a: Let k_a be this number of participants. To not favour one approach, only the results of the $k = max(k_{Pal}, k_{SPE})$ fastest participants from both groups were evaluated for metric 1.1, so that for both groups, the slower participants were left out.



(a) Metric 1.1: Duration of whole task (wt) for both approaches and both systems



(b) Metric 1.1: Duration of only the initial system (is), for both approaches and both systems

Figure 3(a) shows the results of metric 1.1 for the four combinations of approaches and systems in a boxplot, showing the minimum, the lower quartile, the mean, the upper quartile and the maximum for all groups and systems. The number of evaluated results is $k = 3$ for the Media Store (MS) and $k = 6$ for the Web Server (WS).

To get more data points, metric 1.1 was also evaluated for the analysis of the initial system only without design alternatives (=: scope is), now considering all participants (except the aforementioned excluded ones). Figure 3(b) shows the resulting boxplot, including the time to read the exercise.

Table 2 shows the average metric 1.1 for all aforementioned combinations. Additionally, we compared how much longer it takes in average to make the Palladio prediction compared to making the respective SPE prediction. These factors are shown as avd_{Pal}/avd_{SPE}. In average over both scopes, the duration for a Palladio prediction was 1.4 times the duration for an SPE prediction.

We tested our initial hypotheses using Welch's t-test [22], as we cannot assume identical variances for the distributions, and chose a significance level of 0.05. For the initial system, the hypothesis 1.1 is not rejected in a two sided test (p=0.15). Using one sided tests, we found that it is unlikely that students using Palladio needed less that 1.5 times the effort than students using SPE (p=0.08), although not significantly. Overall, it is

Table 2. Metric 1.1: Duration of making a prediction in minutes

	Whole task MS	Whole task WS	Avg	Initial system MS	Initial system WS	Avg	Avg
avd_{Pal}	374	285	329.5	203	191	197	263
avd_{SPE}	284	243	263.5	99	119	109	186
$\frac{avd_{Pal}}{avd_{SPE}}$	1.32	1.17	1.25	2.05	1.61	1.81	1.41

significant that students using Palladio did need more effort than students using SPE for the initial system only (p=$1.7 * 10^{-4}$). The statistical power of these tests were larger than 0.9 and thus satisfactory. For the whole system, i.e. the actual experimental task, hypothesis 1.1 is rejected (p=0.009). Students using Palladio needed significantly less than 1.5 the time than students using SPE, as the opposite is rejected with p=0.004. The statistical power of these two tests is 0.65 and 0.78, respectively, and barely sufficient [18]. Still, students using Palladio needed significantly more time than students using SPE for the whole task as well, as the opposite is rejected (p=0.01,power 0.78).

4.2 Metric 1.2: Break Down of the Duration

We first looked at the break down of the duration as measured in metric 1.1 into the different activities for the initial system only (scope is), because it represented a creation of the models from scratch and we had more data points for it.

Reading (ra) was only an initial reading of the task description, all participants had to read excerpts of the task again while modelling, which was included in the modelling time. For SPE, the participants did not give a separate time for the annotation of resource demands (rd) , but included this time into modelling of the control flow (cf) or of the resource environment (re). Each experiment task contained two usage profiles, so the duration of their modelling, searching for errors and analysing was measured separately for each usage profile and then averaged.

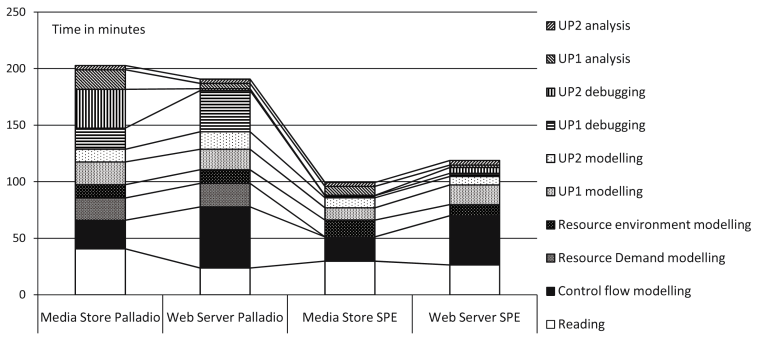


Fig. 3. Metric 1.2: Break down of the duration for the initial system (scope is)

Figure 3 shows the break down of the duration of making a prediction for the initial system, without design alternatives (scope is). It is visible that the entire modelling, including cf, rd, re, and up, was the major activity for both approaches, as expected. The results indicate that hypothesis 1.2 can be held.

Notable results are found for the time needed for searching for errors (err) and the analysis (ana). However, participants using Palladio spent much more time on searching for errors, i.e. fixing wrong or missing parameters: 20%. Here, the participants using SPE only spent 2% (Media Store) and 6% (Web Server), of their time. The proportion of the analyses was fairly constant for the approaches and differs only in the system under study: For the Media Store system, participants spent about 10% of their time in average for the analyses, for the Web Server, only 4%.

The duration of the whole task, i.e. modelling all design alternatives (scope wt) was also composed down to these aspects. The duration of reading $avdact_{a,ra,wt}$ was relatively smaller, because it had just been queried once at the beginning of the task. The other ratios stayed approximately the same. Due to space limitations, we omit the charts here.

5 Discussion

5.1 Threats to Validity

To enable the reader to assess our study, we list some potential threats its validity in the following. We look at the internal, construct, and external validity (a more thorough discussion can be found in [12]).

The *internal validity* states whether changes of an experiment's independent variables are in fact the cause for changes of the dependent variables [23, p.68]. Controlling potential interfering variables ensures a high internal validity. In our experiment, we evaluated the pre-experiment exercises and assigned the students to equally capable groups based on the results to control the different capabilities of the participants. A learning effect might be an interfering variable in our experiment, as the students finished the second experiment session faster than the first one.

A potential bias towards or against Palladio was threatening the internal validity in our experiment, as the participants knew that the experimenters were involved in creating this method. However, we did not notice a strong bias from the collected data and the filled-out questionnaires, as the participants complained equally often about the tools of both approaches.

The *construct validity* states whether the persons and settings used in an experiment represent the analysed constructs well [23, p.71]. Palladio and SPE are both typical performance prediction methods involving UML-like design models. The SPE approach has no special support for component-based systems, and was chosen for the experiment due to its higher maturity compared to existing CBSPE approaches. To allow a comparison, we designed the experimental tasks so that not all specific component-based features of Palladio (e.g. separation of developer roles in component-based development, performance requirements using quantiles) were used.

While our experiment involved student participants, we argue that their performance after the training sessions was comparable to the potential performance of practitioners. Most of the students were close to graduating and will become practitioners soon. Due to the training sessions, their knowledge about the methods was more homogeneous than the knowledge of practitioners with different backgrounds. With a homogeneous group of participants, the significance of the results is even improved. Studies, such as [10], suggest the suitability of students for similar experiments.

The *external validity* states whether the results of an experiment are transferable to other settings than the specific experimental setting [23, p.72]. While we used medium-sized, self-designed systems for the students to analyse, we modelled these system designs and the alternatives after typical distributed systems and commonly known performance patterns [20], which should be representative for the usually analysed systems.

We tried to increase the external validity of our study by letting the participants analyse two different systems, so that differences in the results could be traced back to the systems, and not the prediction methods. Effects that are observed for both tasks are thus more likely to be generalisable to other settings.

Still, the systems under study were modelled on a high abstraction level due to the time constraints of such an experiment. More complex systems would increase the external validity, but would also involve more interfering variables thus decreasing the internal validity. Furthermore, the available information at early development stages is usually limited, which would be reflected by our experimental setting.

5.2 Potential Explanations for the Results

Using SPE, the predictions can be done significantly faster. Using Palladio takes 1.17 to 2.05 times longer, depending on the system under study and the nature of the task. The proportion is higher if we look at the prediction of the initial system only. However, this is not a realistic setting, because a usual task in performance engineering is the comparison of several alternatives. For the evaluation of several alternatives, using Palladio only takes 1.17 or 1.32 times longer. To a certain extent, this can be explained by the reuse character of this scenario: For the prediction of design alternatives, the EGs of SPE were copied and adapted, which is faster than creating new models from scratch, but still a considerable effort. However, for Palladio, the RD-SEFFs of the most components can be reused as is due to their parametrisation, and only single components, their assembly and allocation need to be changed.

To a certain extent, the extra time needed for making Palladio predictions could be traced back to the duration of searching for errors. This might be partly caused by the immaturity of the tool and the limited understandability of the error messages. Using Palladio, more problems occur during creation of the model and searching for errors before the simulation, but the number of problems in the later acceptance tests after simulation is lower than with SPE. The PCM-Bench performs more consistency checks on the models than the SPE-ED tool, thus predictions with the PCM-Bench seem more reliable. However, both tools still allow wrong parameter settings or wrong modelling.

Still, the participants using SPE also needed less time to model and analyse the systems. However, in this experimental setting, not yet considering potential time-savings when reusing models in other projects, SPE is favoured, because it allows to create the models on a higher abstraction level and thus faster. The resulting SPE models are not meant for reuse, which is the case for Palladio models. Furthermore, existing components were not reused in the systems under study and no code was generated from the resulting Palladio models, which might have affected the combined effort of design and implementation. The influence of possible reuse on the effort, however, is deliberately not subject of our experiment and needs further studies.

Next to differences of the approaches presented here, we also found that the results differ for the two systems under study. For the Web Server, both the duration of modelling the control flow and the variance of the overall duration is considerably higher for both approaches.

5.3 Implications for Further Research

Our experiment has several implications for further research. The study could be repeated with a larger sample size to allow a better and more precise quantification of the additional effort. Furthermore, the actual reuse of the created parametrised models in terms of applicability, effort and quality need to be studied. Also more complex, and less componentised systems could be evaluated with the approaches. We also plan to investigate whether cost models on the effort of creating reusable code [16] are suitable for assessing the overhead effort of creating reusable performance prediction models.

For comparative studies between different approaches, a component-based reference system can help avoid researchers applying their methods on their own model examples, which are often tuned to show specific benefits but not general applicability. A recent joint effort by more than 15 research groups has taken steps into this direction by specifying CoCoME (Common Component Modelling Example) [17], which could be used for comparative studies.

6 Related Work

Basics about the area of *performance prediction* can be found in [20,14]. Balsamo et al. [1] give an overview of about 20 recent approaches based on QN, SPN, and SPA. Becker et al. [5] survey performance prediction methods specifically targeting component-based systems. Examples are CB-SPE [7], ROBOCOP [8], and CBML [24].

Empirical studies and controlled experiments [23] are still under-represented in the field of model-based performance predictions, as hardly any studies comparable to ours can be found. Balsamo et al. [3] compared two complementary prediction methods (one based on SPA, one on simulation) by analysing the performance of a naval communication system. However, in that study, the authors of the methods carried out the predictions themselves. Gorton et al. [9] compared predicted performance metrics to measurements in a study, but only used one method for the predictions.

Koziolek et al. [11] conducted a study similar to this one. They compare predictions with SPE [20], Capacity Planning [14], and umlPSI [2] with measurements of an implementation. It attested SPE the most maturity and suitability for early performance predictions and influenced our decision to compare Palladio to SPE.

7 Conclusions

We have conducted an empirical investigation to quantify the higher effort for creating reusable, component-based models for performance prediction in relation to create throw-away models. After substantial training, we let 19 computer science students apply the SPE method and the Palladio method to predict the response times of two example systems. We found that the effort for applying Palladio on the whole task was in average 1.25 times the effort for applying SPE. Our results indicate that in some cases, the effort of creating reusable models for performance prediction can already be justified if the models are reused at least once, if the costs for the reuse itself are low. If the models are reused more often, the additional upfront effort pays off even more.

The results are useful for both practitioners and researchers. Practitioners, such as software architects and performance analysts, get a first quantification of the higher effort to create reusable, component-based models, which they could use in front of management to justify higher upfront costs for modelling. Researchers obtain a reusable experimental setting, which is the basis for future replications of the experiment. The results suggest that it is worthwhile to put more research effort into creating reusable models, because their creation can quickly pay off. However, our study cannot give a definite, overall answer to the questions raised, as the results are also confined to our specific experimental setting.

Our investigation opens up future directions for research. We conducted one of the first empirical studies comparing two performance prediction approaches. The study could be repeated with a larger sample size to allow a better quantification of the additional effort as well as a validation of the results. Furthermore, it has to be assessed whether the promised reusability of the models can be achieved in more complex or less componentised systems. Moreover, the analysis of factors influencing the effort, especially the nature of the systems under study, is an issue for future research.

Details on the Experimental Settings and the Results. can be found in [12], available online at `http://sdq.ipd.uka.de/diploma_theses_study_theses/completed_theses`.

Acknowledgements. We would like to thank Walter Tichy, Lutz Prechelt, and Wilhelm Hasselbring for their kind review of the experimental design and fruitful comments. Furthermore, we thank all members of the SDQ Chair for helping prepare and conduct the experiment. Last, but not least, we thank all students who volunteered to participate in our experiment.

References

1. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. IEEE TSE 30(5), 295–310 (2004)
2. Balsamo, S., Marzolla, M.: A Simulation-Based Approach to Software Performance Modeling. In: Proc. of ESEC/FSE, pp. 363–366. ACM Press, New York (2003)
3. Balsamo, S., Marzolla, M., Di Marco, A., Inverardi, P.: Experimenting different software architectures performance techniques. In: Proc. of WOSP, pp. 115–119. ACM Press, New York (2004)
4. Basili, V.R., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering - 2 Volume Set, pp. 528–532. John Wiley & Sons, Chichester (1994)
5. Becker, S., Grunske, L., Mirandola, R., Overhage, S.: Performance Prediction of Component-Based Systems: A Survey from an Engineering Perspective. In: Reussner, R., Stafford, J.A., Szyperski, C.A. (eds.) Architecting Systems with Trustworthy Components. LNCS, vol. 3938, pp. 169–192. Springer, Heidelberg (2006)
6. Becker, S., Koziolek, H., Reussner, R.: Model-based Performance Prediction with the Palladio Component Model. In: Proc. of WOSP, February 5–8, 2007, pp. 54–65. ACM Sigsoft (2007)

7. Bertolino, A., Mirandola, R.: CB-SPE Tool: Putting Component-Based Performance Engineering into Practice. In: Crnković, I., Stafford, J.A., Schmidt, H.W., Wallnau, K. (eds.) CBSE 2004. LNCS, vol. 3054, pp. 233–248. Springer, Heidelberg (2004)
8. Bondarev, E., de With, P.H.N., Chaudron, M.: Predicting Real-Time Properties of Component-Based Applications. In: Proc. of RTCSA (2004)
9. Gorton, I., Liu, A.: Performance Evaluation of Alternative Component Architectures for Enterprise JavaBean Applications. IEEE Internet Computing 7(3), 18–23 (2003)
10. Höst, M., Regnell, B., Wohlin, C.: Using students as subjects - A comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering 5(3), 201–214 (2000)
11. Koziolek, H., Firus, V.: Empirical Evaluation of Model-based Performance Predictions Methods in Software Development. In: Reussner, R., Mayer, J., Stafford, J.A., Overhage, S., Becker, S., Schroeder, P.J. (eds.) QoSA 2005 and SOQUA 2005. LNCS, vol. 3712, pp. 188–202. Springer, Heidelberg (2005)
12. Martens, A.: Empirical Validation of the Model-driven Performance Prediction Approach Palladio. Master's thesis, Carl-von-Ossietzky Universität Oldenburg (November 2007)
13. Martens, A., Becker, S., Koziolek, H., Reussner, R.: An empirical investigation of the applicability of a component-based performance prediction method. In: EPEW 2008, Palma de Mallorca, Spain (accepted, 2008)
14. Menascé, D.A., Almeida, V.A.F., Dowdy, L.W.: Performance by Design. Prentice-Hall, Englewood Cliffs (2004)
15. Petriu, D.C., Wang, X.: From UML description of high-level software architecture to LQN performance models. In: Nagl, M., Schürr, A., Münch, M. (eds.) AGTIVE 1999. LNCS, vol. 1779, Springer. Heidelberg (2000)
16. Poulin, J.S.: Measuring software reuse: principles, practices, and economic models. Addison-Wesley Longman Publishing Co., Inc., Boston (1996)
17. Rausch, A., Reussner, R., Mirandola, R., Plasil, F. (eds.): The Common Component Modeling Example: Comparing Software Component Models. LNCS. Springer, Heidelberg (to appear, 2008)
18. Sachs, L.: Applied Statistics: A Handbook of Techniques. Springer, New York (1982)
19. Smith, C.U.: Performance Engineering of Software Systems. Addison-Wesley, Reading (1990)
20. Smith, C.U., Williams, L.G.: Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley, Reading (2002)
21. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. ACM Press, Addison-Wesley, Reading (1998)
22. Welch, B.L.: The generalization of student's problem when several different population variances are involved. Biometrika 34, 28–35 (1947)
23. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Dordrecht (2000)
24. Wu, X., Woodside, M.: Performance Modeling from Software Components. SIGSOFT SE Notes 29(1), 290–301 (2004)