Noname manuscript No.
(will be inserted by the editor)

Performance and Reliability Prediction for Evolving Service-Oriented Software Systems

-Industrial Experience Report-

Heiko Koziolek · Bastian Schlich · Steffen Becker · Michael Hauck

Received: date / Accepted: date

Abstract During software system evolution, software architects intuitively trade off the different architecture alternatives for their extra-functional properties, such as performance, maintainability, reliability, security, and usability. Researchers have proposed numerous model-driven prediction methods based on queuing networks or Petri nets, which claim to be more cost-effective and less error-prone than current practice. Practitioners are reluctant to apply these methods because of the unknown prediction accuracy and work effort. We have applied a novel model-driven prediction method called Q-ImPrESS on a large-scale process control system from ABB consisting of several million lines of code. This paper reports on the achieved performance prediction accuracy and reliability prediction sensitivity analyses as well as the effort in person hours for achieving these results.

Keywords Software architecture · Performance prediciton · Reliablity prediction · Case study

H. Koziolek
Industrial Software Systems,
ABB Corporate Research, Ladenburg, Germany
E-mail: heiko.koziolek@de.abb.com

B. Schlich
Industrial Software Systems,
ABB Corporate Research, Ladenburg, Germany
E-mail: bastian.schlich@de.abb.com

S. Becker
University of Paderborn, Germany
E-mail: steffen.becker@upb.de

M. Hauck
Forschungszentrum Informatik (FZI), Karlsruhe, Germany
E-mail: hauck@fzi.de

1 Introduction

Distributed, service-oriented software systems within companies or on the web are constantly evolved due to new customer requirements, failure reports, or technology updates. Such evolution scenarios require architectural changes, for which there are often multiple alternatives (e.g., make-or-buy or selection of technologies). Software architects usually cannot quantify the tradeoffs of these alternatives concerning quality attributes, such as performance, reliability, and maintainability, before implementing them. While current practice often relies on prototyping or former experience to assess design alternatives, researchers have proposed several model-driven prediction methods [1,24,28] to quantitatively evaluate evolution alternatives. These methods claim to be more cost-effective and less error-prone than current practice.

A major challenge to conduct model-driven evolution scenario predictions is first to create a suitable model to evaluate the quality attributes of the current system [59]. Such a model must resemble the architecture, so that architectural evolution scenarios (e.g., replacing a service) can be represented. To reflect performance characteristics, it must include dynamic properties, i.e., control and data flows through the architecture as well as resource demands [28]. To reflect reliability characteristics, it must additionally allow modeling service and/or environmental failure probabilities [24]. Creating such models is currently tedious and error-prone because of sparse tool support and missing step-by-step guidelines [59].

There is only a limited number of documented case studies and industrial experience reports for model-driven prediction approaches. Practitioners often question the accuracy of the model predictions and fear the

potentially expensive effort for creating such models. Existing case studies for performance prediction (e.g., [27,12,22]) and reliability prediction (e.g., [15,45]) often analyze small-scale systems and are carried out by the authors of the methods under controlled laboratory conditions. Their value to practitioners is therefore limited. Additionally, most of these studies have not reported on the effort for creating the models and making the predictions.

The contribution of this paper is a large-scale, industrial case study investigating (i) the achievable prediction accuracy and (ii) the required work effort for model-driven quality predictions. We applied a recently developed model-driven prediction method called Q-ImPrESS (Quality Impact Predictions for Evolving Service-oriented Systems)[46]. The method has been developed in the past three years as a combined effort by several academic and industrial partners within an EU project. It integrates multiple formerly disconnected prediction methods in a single modeling environment.

This paper extends two former publications [31,32] with a survey of related studies, more detail about the Q-ImPrESS method, a much more detailed description of the data collection, extended performance predictions, effort estimations for both performance and reliability predictions as well as a detailed discussion of the modeling, implementation, and process issues of Q-ImPrESS.

We applied the method on a large-scale, distributed process control system from ABB and report our experiences and lessons learned in this paper. We found that reasonable performance prediction accuracy could be achieved, and estimated the effort for third-party applications of the method in the range of one person month per evolution scenario and quality attribute. As most studies before, we were not able to obtain reliability data from monitoring operational reliability and validate predictions. Also, we found that the Q-ImPrESS method still has lots of potential for improving the meta-model and prediction tools. Our experience report provides initial evidence about the costs and benefits of this method to enable third party users assessing the usefulness in their own context.

The remainder of this paper is structured according to the guidelines for case study research [48]: Section 2 surveys the state of the art in determining the prediction accuracy and application effort of model-driven quality prediction methods by analyzing a number of case study reports from literature. Section 3 introduces the Q-ImPrESS method with its process, meta-model, and tools. Section 4 formulates our research questions and details the system under analysis. Section 5 documents both the case study execution and the results of each phase. It also discusses threats to validity. Section 6 discusses modeling, implementation, and process issues, before Section 7 concludes the paper and sketches future work.

2 Related Work

This section surveys related empirical studies and the given evidence to the research questions stated in this paper. Section 2.1 discusses the experience on performance prediction case studies, while Section 2.2 does the same for reliability prediction case studies. Finally, Section 2.3 describes further related studies.

2.1 Experience on Performance Predictions

Early approaches on performance modeling [25,56] focused on low-level formal models, such as Markov chains, queuing networks, stochastic Petri nets, and stochastic process algebra, which reflected the structure of software systems and the interaction of different software components only to a limited extend. The software performance engineering (SPE) approach by Connie Smith [52] was the first approach putting more emphasis on software structures.

Recently many approaches and performance modeling formalism have been proposed, which model the performance of a system on the level of the software architecture and interacting components [59]. Layered queuing networks [47] and queuing Petri nets [3] are among the most popular models in this area. Researchers have also explored mapping UML models to classical performance models [10] and designing performance models specifically for component-based software systems [28]. There is also work on deriving performance models directly from performance measurement traces [40].

In the following, we discuss a number of recent performance modeling case studies and experience report, where the software architecture is explicitly reflected in the models. The emphasis in this discussion is on the performance prediction accuracy achieved by these studies as well as the effort required for these predictions.

Liu and Gorton [34] modeled the performance of the Stock-Online system, a typical EJB application with enterprise beans and a database. They constructed a queuing network model and additionally expressed EJB architectural patterns with UML activity diagrams. After benchmarking the system to determine the service

demands for the queuing model they performed a capacity planning predicting the throughput for a growing number of database connections. The average deviation between predicted and measured response times was below 15 percent, but the effort for constructing the model was not reported.

Xu et al. [60] analyzed a modified version of Duke's bank application, a three-tier application which was used by the J2EE tutorials for explaining the Java Enterprise edition. They build a layered queuing network, which for example represented the thread pools present in the system. To determine service demands, they measured the performance of the system on three computers. The subsequent capacity planning study analyzed the response time and throughput of the system for a growing number of clients. The average error for the response time predictions was below 24 percent. The effort for this study was not reported.

Kounev [27] constructed a queuing Petri net for the specJAppServer2004. It is an enterprise system representing a web platform for an automobile manufacturer and allows browsing cataloged, tracking inventories and purchasing. The author measured the system for different workloads and analyzed the impact of a higher number of application server nodes (i.e., 2,4,6,8) for various performance metrics. The mean deviation between predictions and measurements was below 20 percent for response times, below 7 percent for throughput and below 9 percent for CPU utilization. Again the effort for the study was not discussed.

Jin et al. [26] modeled the performance of a meter-data system for utilities with a layered queuing network model. The system consisted of an Oracle database with 13 applications. They constructed a very large LQN with more than 20 processors and over 100 tasks. After benchmarking the system, they analyzed the throughput of the system for massively higher workloads. The error between predicted and measured throughput was below 8 percent.

Huber et al. [22] built a Palladio Component Model instance for a storage virtualization system from IBM. They measured performance of the system and analyzed the performance for a synchronous and asynchronous re-design using the model. The error between predicted and measured throughput for the current system was below 21 percent. The authors estimated the effort for the case study to approx. four person months and discussed that implementing a prototype to measure the performance for the re-design alternatives would have taken 24 person months.

Concluding, there is already a growing amount of empirical studies on architecture-level performance prediction in literature. Most studies reported a prediction error of less than 30 percent for the various performance metrics. Furthermore, most studies did not report on the effort to achieve this accuracy.

2.2 Experience on Reliability Predictions

Seminal research on software reliability engineering focused on system testing and system-level reliability growth models [42]. Musa provided an approach for the reliability analysis of evolving software systems [41]. However, these approaches did not take the software architecture into account.

Recently, multiple surveys ([17,13,24]) review more than 20 methods for architecture-based software reliability analysis (ABSRA) (e. g., [9]). For example, Singh et al. [50] proposed to transform annotated UML component models into a Bayesian model, which can be used for reliability prediction during the design phase. Goseva [13] pointed out in 2007 that "very little effort has been devoted to the validation of ABSRA techniques". Immonen et al. [24] stated in 2008 "a great lack of publications covering large-scale, industrial application of the methods" and that "validation of the methods is based only on the authors' experiments and evaluations in laboratory circumstances".

The following discusses a number of the most related and up-to-date case studies on reliability prediction with a special emphasis on their data collection methods. Hardly any of these studies has attempted to assess the accuracy of the respective models with a comparison of predicted and measured values. Several authors, however, performed sensitivity analyses with their models to demonstrate their robustness against uncertain input values. There is no evidence on the effort required for these investigations.

Gokhale et al. [14] analyzed the SHARPE tool (35 KLOC, C-code) for stochastic modeling by constructing a DTMC. For estimating component failure rates, they used the enhanced non-homogeneous Poisson process model that incorporates the failure intensity of a component (i.e., 4 errors per 1 KLOC in this study) and the expected time spent in each component. The latter was determined by profiling the system with the ATAC tool while executing 735 test cases from a regression test suite. The study found that the system reliability could be increased from 0.9903 to 0.9950 if the fault density per component was reduced from 4 to 1 error per 1 KLOC.

Goseva et al. [18] performed a case study on a system (10 KLOC, C-code) of the European Space Agency (ESA). They divided the system into three subsystems and constructed a DTMC according to Cheung [9]. For

estimating failure probabilities, faults discovered during integration testing and runtime were re-inserted into the software. The authors then executed random tests and estimated the reliability of each component with the ratio of failed tests versus successful tests. Finally, they used the DTMC model in a sensitivity analysis.

The largest case study reported in literature so far is also from Goseva et al. [15]. Here, the authors divided the implementation of the GCC C-compiler comprising 350 KLOC into 13 software components. They executed 2126 test cases from GCC 3.3.3 on GCC 3.2.3, so that 111 failures could be detected in the old version of the software, for which test cases had been added in the new version. The authors used the tool gprof to record a number of execution profiles into a database and filtered this data to determine transition probabilities between components. Finally, a DTMC was constructed and solved. The authors compared the system reliability prediction (0.9997) to the actual system reliability (0.9724). The same authors applied a similar approach on a smaller system (11 KLOC) [16].

Wang et al. [57] analyzed the so-called stock market system (SMS), which is widely used in industry (13 KLOC, C/C++ code, 15 components). They executed 13,596 test cases against the system and observed 121 failures, which they mapped to component failure probabilities. They recorded transitions between components via manual instrumentation of the code and derived transition probabilities from this data. The authors constructed a DTMC and predicted the system reliability.

Compared to existing case studies, our study analyzes a significantly larger system and thus offers more insight into the industrial applicability of ABSRA methods. Due to the size of the system, we use different methods for determining component failure probabilities and transition probabilities as in former studies. Our focus is on determining the cost-effectiveness of ABSRA.

2.3 Other studies

Martens et al. [36] conducted a series of experiments involving more than 40 computer science students to determine the accuracy and effort for applying performance prediction methods in a controlled setting.

This paper is based on our former studies [31,32], but provides more detailed effort estimation, more performance prediction results, and a longer discussion of the benefits and drawbacks of the Q-ImPrESS method.

3 Background: The Q-ImPrESS Method

This section introduces the Q-ImPrESS method. We illustrate the approach by a running example and present the Q-ImPrESS process. Afterwards, we highlight the main concepts of the Q-ImPrESS Service Architecture Meta Model and describe the Q-ImPrESS software tools.

3.1 Running example

In our example system, a client communicates with a server to retrieve user data (e.g., login and password) stored in a database. Fig. 1 shows the components and their execution environment.

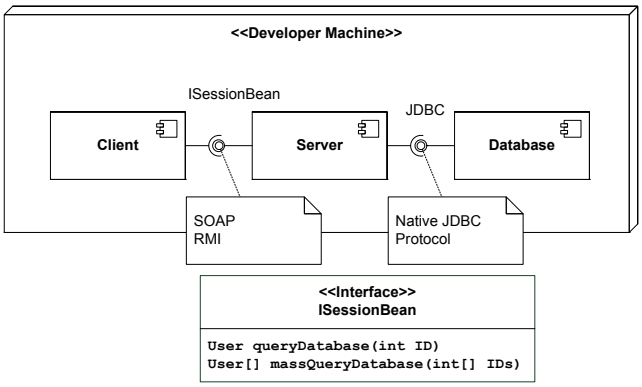

Fig. 1 The Client/Server example system

The `Client` component issues user requests, which are handled by the `Server` component. The `Server` component queries the `Database` component containing the requested data. The `Client` communicates with the `Server` using the `ISessionBean` interface. It has two operations, one single `User` data request operation and one mass `User` data request operation. The communication between the components is denoted by annotations indicating the possible communication styles.

We now introduce an evolution scenario, which represents a change in the system environment. Such a change can for example be a changed user workload, new or changed system requirements, or a changed hardware or middleware environment on which the system is to be deployed. In the Client/Server example, we assume that the evolution scenario is based on a changed user workload, i.e. the amount of users that access the system has increased over time. In order to cope with the increased user workload, the example system has to be adopted in order to meet quality requirements.

For this example, several architectural alternatives can be identified which can be implemented in order to cope with the evolution scenario. Every alternative can have an impact on different quality attributes of the system.

- **SOAP or RMI communication style:** The communication between the `Client` and the `Server` component can be implemented by using the SOAP or the RMI protocol. Both protocols incur different performance overheads, for example due to marshalling/demarshalling overhead, or because the amount of data transferred varies with the selected protocol.
- **Optional inclusion of a cache component** To avoid frequent database accesses, a cache component could speed up user requests. It might allow better average request response times, but also result in different system reliability, as the cache component can be an additional source of failures.
- **Different database access patterns:** The operation `massQueryDatabase` could either issue a single database request for each user id and combine all retrieved `User` values into the result value. An alternative implementation could combine all `User` requests into a single database call, which would lead to fewer, but more time-consuming database calls.

3.2 The Q-ImPrESS Process

Q-ImPrESS defines a process that guides its users in executing its various tasks and using the respective tools [37]. The process assumes the existence of a modeled base system (legacy application or application under development) for which a software architect or quality analyst wants to evaluate different design alternatives. In our example, we could select the architecture as shown in Fig. 1 using RMI for communication, no caching, and single DB requests per user ID as base architecture.

The evaluation goal is to quantitatively predict different quality attributes (e.g., expected response times and probabilities of failure on demand) for each architectural alternative and then to pick the alternative with the best trade-off among the quality attributes. In our example, we are interested in the trade-off among performance and reliability of the specified architectural alternatives.

Fig. 2 illustrates the main steps of the Q-ImPrESS process. Grey steps illustrate activities that require manual work and are not directly supported by the Q-ImPrESS tools. White steps represent tool-supported core activities in Q-ImPrESS. They have fine grained defined sub-processes described briefly in the following (for details refer to [37]). The Q-ImPrESS method's documentation also gives estimates for the durations of each of the process steps involved to allow an estimate of the necessary effort for its application. In this experience report, we will address these estimates and their accuracy (cf. Section 5).

The Q-ImPrESS process starts with a *requirements collection* step (3.1). This step focuses on extra-functional properties, such as performance, reliability, and costs. For our example, functional requirements are the retrieval of user DB records and extra-functional requirements include a maximum response time and a maximum probability of failure on demand.

Next, the software architect defines a set of potential *change scenarios* (3.2) each leading to a new architecture that fulfills the new functional and potentially fulfills the new extra-functional properties. The change scenarios for our example include changing the communication protocols, adding cache components, or picking different server components exhibiting different database query patterns.

For each of these alternatives, the following two steps are repeated: First, the architect *creates a model* of the resulting target architecture by altering the base architecture (3.3). Second, based on the new model of the architectural alternative, Q-ImPrESS tools are used to *predict the values* of the different quality attributes for the alternative (3.4). In the end of the overall step, this leads to a set of architectural alternatives plus the predicted values of their quality attributes.

In the modeling step (3.3), the software architect has to gather a variety of quality annotations, e.g., resource demands of single steps, branching probabilities, loop counts, etc. This step requires collecting measurements from existing systems or estimations based on former experiences. In case of taking measurements this step also includes configuring the existing system in a test environment and instrumenting it with the necessary monitoring probes. Depending on the underlying system, a suitable performance measurement framework, such as Eclipse TPTP, JProfiler, dotTrace, VTune, dynatrace, or Windows Performance Monitor, has to be evaluated and bought beforehand. In case of estimating model annotations, the software architect needs training courses in providing good parameter estimates including the necessary statistics skills.

For example, alternative A_1 in our Client/Server system uses SOAP instead of RMI, adds a cache component, and favors bulk DB queries as DB query pattern. Analyzing this alternative may reveal that the overall response time decreases, i.e., the architecture exhibits a better performance compared to the base architecture.

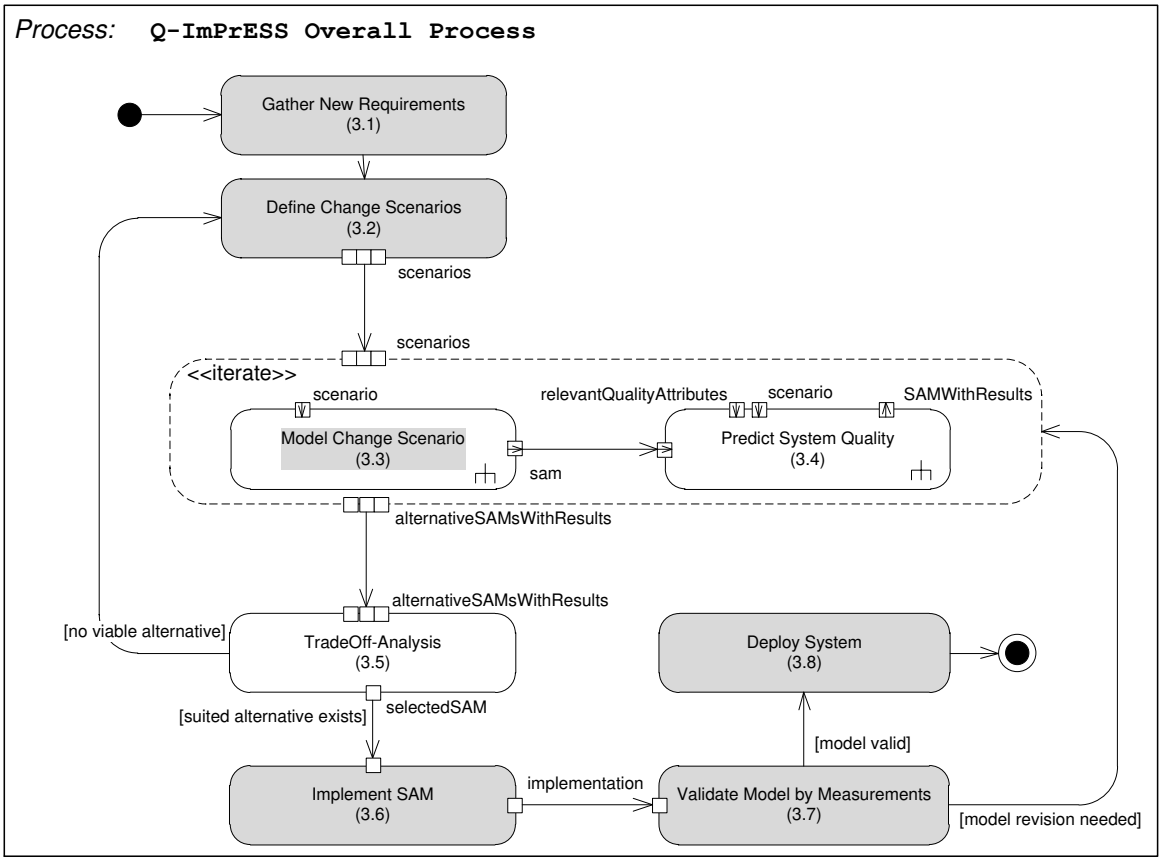

Fig. 2 Overview of the main steps in the Q-ImPrESS process

On the other hand, the additional components reduce the system's reliability, i.e., the probability of failure on demand increases. For another architecture alternative called A_2, imagine that it offers increased reliability but decreased performance.

Based on the set of architecture alternatives and their quantitative evaluation results, the *trade-off analysis step* (3.5) ranks all alternatives based on the preferences of the software architect. Q-ImPrESS additionally features the tool PerOpteryx [35] to search the design space for solutions with optimal trade-offs between quality attributes. Preferences of the software architect are gathered by the trade-off analysis tool using the analytical hierarchy process (AHP) method [21]. The tool then ranks the architecture alternatives based on the software architects preferences for the different extra-functional properties. If we assume in our example a preference of the software architect for performance, the AHP would finally recommend alternative A_1.

In case the trade-off analysis method results in a suitable architecture to implement, the development process continues and the *system is being put into practice* (3.6). The example system to implement would be the one with SOAP, cache, and bulk database accesses. After the implementation phase, the predicted quality attributes should be *validated* in a testing step (3.7). This is important as certain model assumptions or inaccurate model input may lead to inaccurate predictions in some cases. If the system passes the test for the new functional and non-functional properties, it *gets deployed* and hence becomes available for end-users (3.8).

3.3 Service Architecture Meta-Model

To support the Q-ImPrESS user in modeling service-oriented architectures, the Q-ImPrESS Service Architecture Meta-Model (SAMM) was developed. This meta-model was newly created during the Q-ImPrESS project but incorporated concepts from the Palladio Component Model [6], KLAPER [19], as well as UML2 and the UML MARTE profile [44]. We refer to instances of the SAMM as service architecture models (SAM), cf. Fig. 2.

Due to space reasons we cannot formally introduce the more than 100 meta-classes of this model within this paper. Instead we briefly explain the main concepts of the model using our running example in the

following. We refer the reader interested in more details to the comprehensive technical report describing the SAMM [5].

The SAMM consists of six different model packages, which encapsulate different concerns and enable replacement of individual parts with limited overhead:

Static structure package: This package includes modeling constructs for services, components that provide services, provided and required interfaces, interfaces signatures, and datatypes. It is possible to bundle multiple components into a composite component with provided and required interfaces, which then looks like a primitive component from the outside. From the Client/Server example, the three components, its interfaces, and the User datatype are modeled with static package model elements.

Behavior package: For each service implemented by a component, the developer can specify a behavior called service-effect specification (SEFFs), which reflects the internal control flow but abstracts from low-level source code aspects. SEFFs model the behavior of a component as a number of internal actions and external calls connected by sequential, alternative, or parallel control flow operators. A single internal action models a large section of the source code. In the Client/Server example, two SEFFs model the internal behavior of the `Server's` services queryDatabase and massQueryDatabase and include calls to the database component.

Deployment package: This package allows to model the hardware environment of the system as well as component allocation to the hardware resources. This includes a detailed infrastructure model (servers and server resources) as well as the component allocation context, i.e., which component is used for offering a service and on which server the component is deployed and executed. For the Client/Server example, the deployment package includes a single server node with a CPU and harddisk on which all components are allocated.

Usage package: As quality of service heavily depends on the user workloads, SAMM also enables modeling different user scenarios for a service-oriented system. Here, different user classes (i.e., workloads) can be modeled indicating the services that are called. The usage model of the Client/Server example includes a request arrival rate to the `ISessionBean` interface. In this case, the modeled user scenario represents the evolution scenario, i.e. the new user workload of the system.

QoS annotations package: This package allows to annotate the SEFFs with branch transition probabilities, loop counts, resource time demands, and failure probabilities. It can be extended with additional quality properties. In the Client/Server example, the developer would annotate the internal actions in the `Server's` and `Database's` SEFFs with measured CPU execution times, hard disk demands, and failure probabilities.

Results package: The result model serves as central repository for the conducted analyses of different quality predictions and is used as input for the trade-off analysis. In the Client/Server example, the result model would contain predicted response times, throughputs, resource utilization, and system failure probabilities after running the analysis tools on the fully specified model.

3.4 Q-ImPrESS Tools

The Q-ImPrESS IDE combines tools for creating and editing models, performing predictions, and conducting a tradeoff analysis.

Fig. 3 shows a screenshot of the Q-ImPrESS IDE illustrating the Client/Server example. The IDE is based on the Eclipse IDE and is composed of different views and editors. The view in the upper left side of the IDE is used for managing different model alternatives. The Client/Server system has been modeled in three different alternatives. The main alternative contains the initial Client/Server system. The alternative "changed database implementation" contains a Client/Server model with a changed behavior specification for the `Server` component. The alternative "with database cache" contains the Client/Server alternative with an additional cache component.

Two models of the main alternative are displayed in the right hand side of Fig. 3. The upper editor shows how the three components are connected with each other. The lower editor shows a SEFF for an interface operation of the `Client` component, which consists of three consecutive actions. The first and last action denote internal actions, which can be annotated with resource demands and failure probabilities. The second action denotes an external call action indicating the access to the required `Server` interface.

To perform a prediction, a SAMM model instance is automatically transformed into a corresponding analysis model. The Q-ImPrESS IDE comprises and integrates several analysis tools:

- The **PCM analytical solver** is an analytical solver for performance prediction. A SAMM model instance is transformed into a PCM model [6], which is transformed into a layered queuing network representation as input for the LQN solver [30].

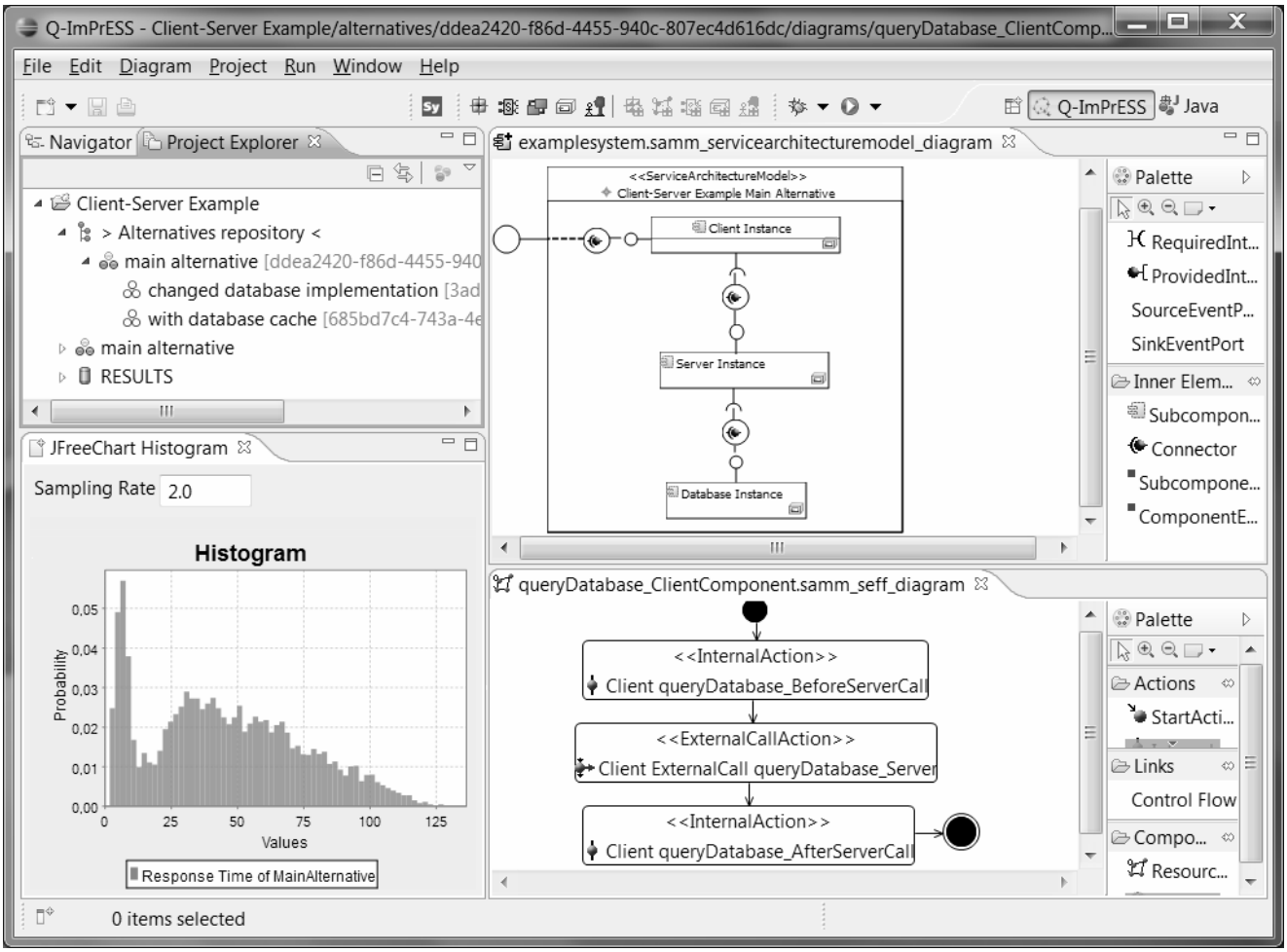

Fig. 3 The Q-ImPrESS IDE

- **PCM SimuCOM** provides a discrete-event simulation framework for performance prediction [4]. This analysis is also based on a PCM model, which is transformed into Java code that integrates into the simulation framework.
- The **Reliability Solver** is based on the KLAPER [19] model. A SAMM instance is transformed into a KLAPER model, which is transformed into a discrete time Markov chain and solved using the probabilistic model checker PRISM.
- The **KAMP Maintainability tool** uses a SAMM model instance in order to estimate cost and change efforts for a change request based on an architecture model. Maintainability prediction is not in the scope of this paper, for more information, please refer to [55].
- Based on the different prediction results, the **AHP tradeoff wizard** [21] guides the user in performing an AHP trade-off analysis for the different quality attributes.

The different analysis tools are integrated into the Q-ImPrESS IDE and support the software architect in performing quality predictions that require little effort. In the lower left side of the IDE screenshot in Fig. 3, a result view for a performance prediction with PCM SimuCOM is shown. For the Client/Server example, a performance prediction of the main alternative has been conducted. The result view shows a histogram of the simulated response time distribution in milliseconds for the `User` data request operation.

The background on Q-ImPrESS provided in this section allows us explain our case study design in the next section.

4 Case Study Design

This section first formulates our research questions 4.1 before briefly describing the system under study 5.

4.1 Research questions

The goal of our case study is stated in Tab. 1 according to the goal-question-metric (GQM) template [2].

Purpose	Evaluate
Object	Applicability
Focus	Model-based quality prediction
Viewpoint	Quality analyst
Environment	Large-scale industrial software system

Table 1 Goal of our case study according to the GQM template

We use the Q-ImPrESS method as a representative, state-of-the-art, model-based quality prediction method. Furthermore, we interpret the term 'applicability' in terms of the prediction accuracy of the method and the effort needed in terms of working hours to execute the method. Thus, from the goal in Tab. 1, we derived the research questions in Tab. 2:

Q1	What is the performance prediction *accuracy* of Q-ImPrESS?
Q2	How much *time* is required for performance prediction with Q-ImPrESS?
Q3	What is the reliability prediction *accuracy* of Q-ImPrESS?
Q4	How much *time* is required for reliability prediction with Q-ImPrESS?

Table 2 Research questions of our case study

To operationalize these questions, we defined several metrics in Tab. 3, which guided our data collection procedures. As performance metrics for Q1, we chose CPU utilization and throughput of the system because these are the metrics that the stakeholders of the system are typically interested in. For Q2, we decided to track the time for selecting a data collection method, conducting the actual measurements, analyzing the data, and carrying out model calibration and initial performance predictions.

For reliability prediction, we used the probability of failure on demand as the metric for Q3. Similar to Q2, we tracked the time for selecting a data collection method, conducting the reliability estimations, analyzing the data, and carrying out initial reliability predictions for Q4.

Tab. 3 also includes a hypothesis for each metric, which we formulated before the case study to persist our expectations and gain a more explicit learning effect after data collection.

4.2 Case: Industrial Control System

We selected a process control system (PCS) from the automation domain to be the system under study. A PCS manages time-dependent industrial processes, e. g., power generation, pulp and paper handling, and oil and gas processing. It periodically collects sensor data like temperature, flow, and pressure from various field devices and visualizes it for human operators. The operators use the system to manipulate actuators in the process, e. g., pumps, valves, and heaters. The system can automatically execute predefined actions and informs operators of irregular conditions using alarms.

Our case study focuses on the server-side part of one ABB PCS and neglects embedded devices. The system under study consists of more than 3 million lines of C++ code and is structured into 8 subsystems with several hundred Microsoft COM components. It is a service-oriented system and consists of several dozen OS processes (i. e., Windows services) during runtime. Clients, such as operator workplace applications, communicate with the services via open standards (e. g., OPC). Our testbed consisted of two regular PCs with quad core CPUs, which are often used in typical smaller customer setups.

ABB continuously evolves the system due to new customer requirements, technology updates, and bug reports. It is desired to predict the impact of these changes in advance, before actually implementing them. Therefore, we a created a model reflecting the current implementation of the system and calibrated it to reflect the current performance and reliability properties. The predictions then concerned different evolution scenarios, which were based on typical evolution steps and had an impact on the architecture. For the predictions, the initial architectural model was adapted to reflect the changes of the evolution scenario, so that predictions for potential change alternatives of the system could be made without changing the implementation.

The following evolution scenarios have been analyzed:

- **Usage profile evolution:** The amount of data items requested by clients from the system continuously increases due to an increasing number of more complex field devices in modern industrial processes. For several former updates of the system, the supported amount of processable field device data was a major selling point of the system and can be a competitive advantage. Thus, it is useful to predict the performance and reliability for higher system workloads.
- **Resource environment evolution:** As an industrial system managed by the ABB PCS can be operating for decades, it is likely that the server hardware will be replaced over the course of the life-cycle of such a system. CPU speeds are constantly increasing (cf. Intel's processor roadmap) and the cur-

Q#	M#	Description	Scale
Q1	M1.1	Mean percentage of deviation between predicted and measured CPU utilizations (Hypothesis: M1.1 < 30)	0...100
	M1.2	Mean percentage of deviation between predicted and measured throughputs (Hypothesis: M1.2 < 30)	0...100
Q2	M2.1	Time for selecting an appropriate data collection method (Hypothesis: M2.1 < 8h)	Time (h)
	M2.2	Time for performance measurement and data collection for a scenario. (Hypothesis: M2.2 < 8h)	Time (h)
	M2.3	Time for data analysis of measurements to derive resource demands. (Hypothesis: M2.3 < 8h)	Time (h)
	M2.4	Time for model calibration and initial performance prediction (Hypothesis: M2.4 < 2h)	Time (h)
Q3	M3.1	Mean percentage of deviation between predicted and measured POFOD (Hypothesis: M3.1 < 10)	0...100
Q4	M4.1	Time for selecting an appropriate data collection method (Hypothesis: M4.1 < 8h)	Time (h)
	M4.2	Time for data collection and reliability estimation for a scenario. (Hypothesis: M4.2 < 8h)	Time (h)
	M4.3	Time for data analysis of the reliability estimations to derive failure probabilities. (Hypothesis: M4.3 < 8h)	Time (h)
	M4.4	Time for model calibration and initial reliability prediction (Hypothesis: M4.4 < 2h)	Time (h)

Table 3 Data collection metrics of our case study

rent trend towards multi-core and many-core processors also evolves the resource environment of an installation of the ABB PCS. Based on predictions for evolved resource environments the customer recommendations for sizing the hardware of the system can be updated.

– **Allocation evolution:** The ABB PCS has a scalable topology and can be installed in different sizes depending on the complexity of the customer's industrial process. For smaller settings, the components of the ABB PCS can be installed on a single server, whereas for larger settings, individual components may be allocated to dedicated server nodes to increase performance and availability. Customers can move from a smaller system size to a larger system size if their process gets more complex.
– **Component evolution:** Over the course of the life-cycle of the ABB PCS, certain components can be replaced with newer implementations because of customer requirements or technology updates. The component-based structure of the Q-ImPrESS model instances allows for easy replacements of components also in the model. The changed performance and reliability of a replaced component needs to be modeled by new resource demands and failure probabilities. Their values can be based on prototype measurements, former experience, or - in case of COTS - third-party input.

The effort for actually implementing these scenarios in the existing system varies. Usage profile evolution (i.e., support for higher workloads) and component evolution can take several person-years based on our experience. Thus, it is useful to conduct model-based predictions in advance. Resource environment evolution and allocation evolution are easier to implement but require purchasing additional hardware, where previous model-based simulation experiments can avoid suboptimal investments.

5 Case Study Execution & Results

This section describes the execution of our case study, which includes initial modeling, data collection for the different quality attributes, predictions, and trade-off analysis. Fig. 4 provides an overview of the following section. First, Section 5.1 explains how we modelled the static structure, behavior, and deployment of the ABB PCS. Section 5.2 elaborates on the data collection and predictions for performance analysis. Section 5.3 elaborates on the data collection and predictions for reliability analysis. Section 5.4 documents our experiences using the Q-ImPrESS tradeoff analysis tools. Finally, Section 5.5 dicusses the internal and external validity of our case study.

5.1 Modeling

As a prerequisite to carry out performance and reliability predictions, we first created a Q-ImPrESS model of the architecture without any QoS annotations. While

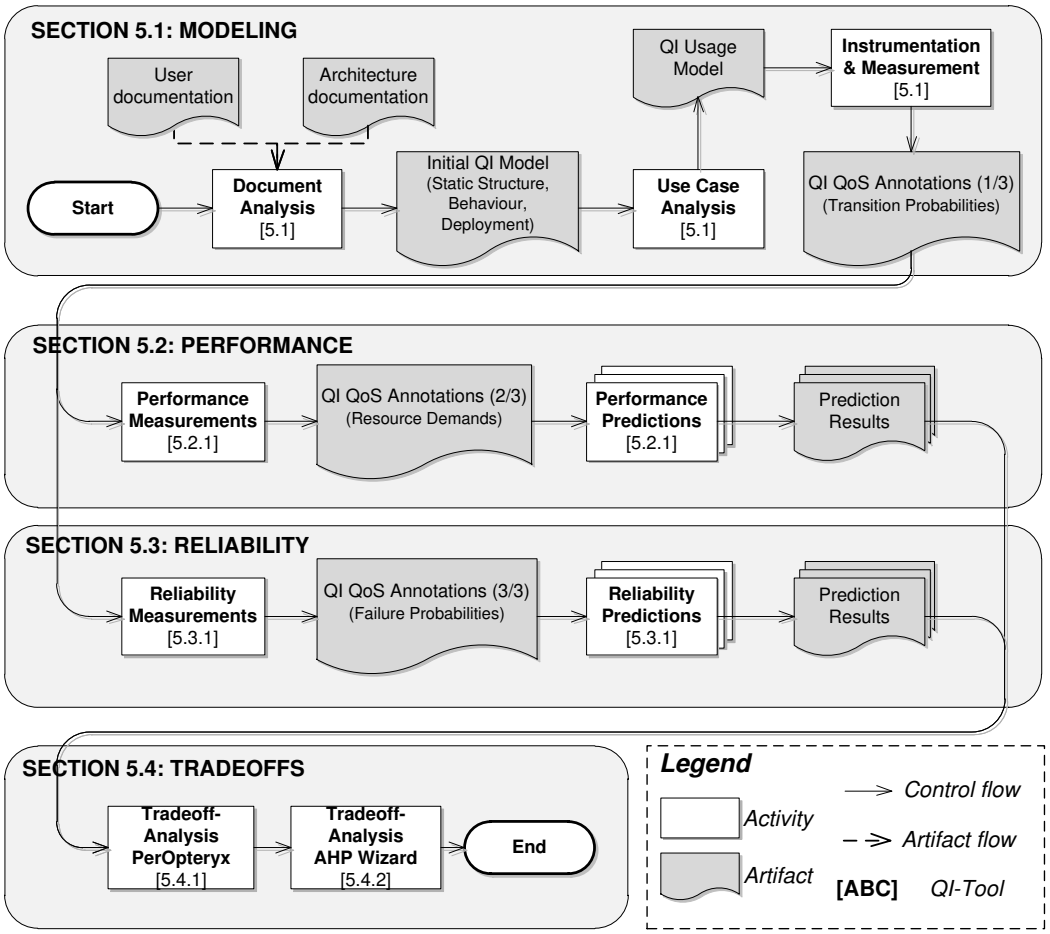

Fig. 4 Case study execution: data collection and prediction steps

Q-ImPrESS offers two reverse engineering tools to extract such an initial architectural model from source code, we were not able to execute these tools successfully on the code of the ABB PCS. Parsing errors due to proprietary code extensions as well as scalability issues prevented us from producing a meaningful model of the PCS using the tools [32]. These limitations might not persist for Java-based systems, which the reverse engineering tools are better suited for.

Instead, we created the Q-ImPrESS models manually using the available documentation and code inspections. The main challenge for modeling is to find a suitable abstraction level that is both manageable and enables meaningful predictions. We therefore searched for an abstraction level sufficient to analyze the architectural evolution scenarios. An important question with respect to Q-ImPrESS is to decide which parts of the system should be considered as *components* as there are no guidelines given by the method.

The PCS architecture documentation included a high-level subsystem description. However, modeling on this level of abstraction would exclude analyzing several evolution scenarios that would not be reflected in the coarse-grained structure. Thus we aimed at a more refined model. We deemed using Microsoft COM components as Q-ImPrESS components as too detailed because the system includes hundreds of these components, whose modeling would not be necessary to evaluate the evolution scenarios.

Instead we mapped runtime processes of the system to primitive components (i. e., services) of the Q-ImPrESS SAMM and subsystems to composite components. The runtime processes implement Windows services and therefore make up the service-oriented structure of the PCS. The process abstraction allowed us to easily measure component resource demands using the Windows performance monitor, which can record service times per process. It also allowed us to analyze scenarios where components are allocated to different servers and to include the available reliability data into the composite components.

Fig. 5 depicts an abstracted overview of the architectural model including 8 composite components (gray), 20 primitive components (white), three servers,

and four use cases. Here, primitive components refer not to OO-classes, but to larger architecture elements. Some of these components comprise several hundreds of thousand lines of code. The model cannot be described with all detail here (details in [29]), but it is available for download[1]. For some of the composite components, their inner details are abstracted in the figure. The three server set up resembles a standard configuration sold to customers. The figure exemplary shows one SEFF at the bottom, which defines the call propagation and hardware usage of component `C13`. In total, the model contains 28 SEFFs, which are not shown here for brevity.

We identified four key usage scenarios of the system relevant for quality predictions based on discussions with experienced developers and consultation of the user documentation. They are visualized at the top of Fig. 5 with use case actors. Each scenario triggers a certain control and data flow through the system's components. One of the scenarios was modeled as a proxy for a number of functional scenarios less critical for performance and reliability, so that some "background resource demands" were present in the model. This emulated a more realistic execution of the system.

The four scenarios relate to steady-state use cases of the system, i.e., they are constantly executed in parallel within a running industrial system. We omitted transient use cases (e.g., maintenance activities or engineering tasks) to limit the scope of our analysis. The selected scenarios have a number of configuration options, which impact performance and reliability. These include, for example, subscription rates for clients, package sizes, or the types and amount of data in the system. To reflect typical customer settings, we aligned these parameters with recommendations provided in the user documentation.

Like any other performance or reliability prediction model, a Q-ImPrESS model requires the specification of transition probabilities between components, which are encoded in the branch probabilities of the SEFFs. These probabilities can only be determined by instrumenting and running the system as they cannot be derived statically from the source code.

Therefore, we set up a testbed for the system, which consisted of two standard PCs with quad-core CPUs and 4 GB RAM each connected via Gigabit Ethernet. One PC ran a virtual machine with the ABB PCS installed. To simulate the embedded controller devices the other PC ran a virtual controller software. The virtual controller received a number of simple control programs via a download from an external engineering tool. The control programs changed the values of several thousand items each second to emulate the data volume in a real industrial process.

Before running the application, we configured an ABB development tool for logging CS system calls to record only interactions between processes. This involved identifying the components responsible for process transitions and adjusting the log detail level for these components. Each process transition represents a DCOM call between two processes. Each log entry contains information about which process called which other process.

We then continuously executed the system for two days to produce representative process transitions logs with limited distorting, that is, transient conditions. Running the system included starting all applications (i. e., the PCS, the additional servers, the logging application, and all clients), performing the initial setup, and operating the system. Operating of the system consisted of observing data and interacting with the system. It resulted in a five-digit number of invocations, and the log file generated in this step had a size of 2 GB.

The created log files were then passed to a self-coded script, which generated the list of processes involved, the transitions between these processes, and the probabilities of these transitions. We validated the resulting 20 branch transition probabilities by examining the different paths through the model and matching these paths to the operations that we actually executed. During this process, we were supported by two PCS experts.

Manually modeling our system structure using Q-ImPrESS required substantial effort (approx. 1.5 person months). For performance analyses, it would have been easier to create a simple queuing model, but this would have complicated analyzing evolution scenarios regarding the topology of the components. It should be mentioned that thinking about an architecture in terms of the Q-ImPrESS model and being forced to formalize certain elements can bring additional incentives besides QoS predictions by improving the architectural documentation.

The resulting Q-ImPrESS model of the PCS included 28 SEFFs but still lacked the resource demands and failure probabilities on internal actions, which are necessary for performance and reliability predictions. The following two section focus on determining these values and conducting subsequent predictions for the evolution scenarios.

[1] http://www.q-impress.eu/wordpress/demonstrators/abb-demonstrator/

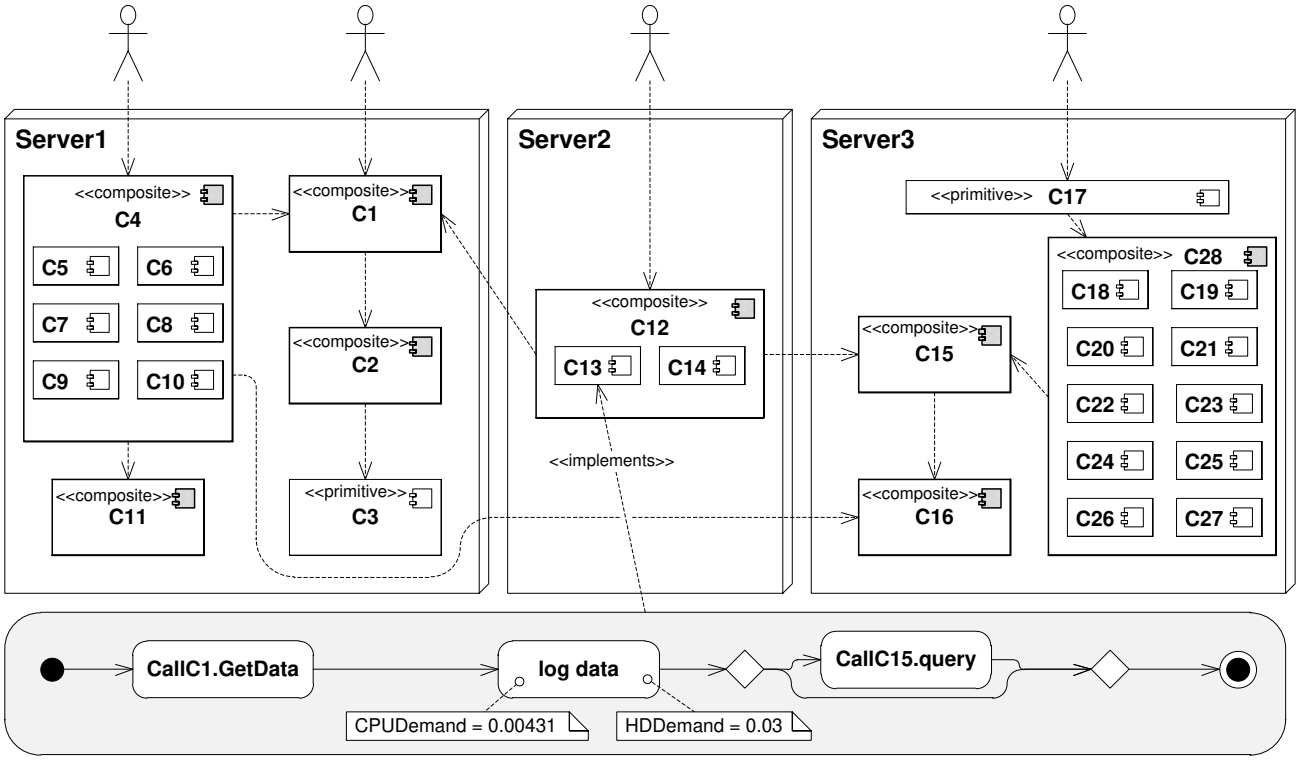

Fig. 5 Q-ImPrESS architectural model of the ABB process control system: components, connectors, deployment, and service effect specification (in UML notation)

5.2 Performance Prediction

As a prerequisite for performance predictions, we first had to annotate the PCS Q-ImPrESS model with resource demands (i.e., execution times of individual services). This section describes the conducted performance measurement and subsequent performance predictions for the different evolution scenarios (Section 5.2.1), as well as the answers to research questions Q1 and Q2 (Section 5.2.2).

5.2.1 Data Collection

Measurement Measurements of the CPU and HDD utilization U_i per process using the Windows Performance Monitor (WPM) determined the resource demands of the ABB PCS's services. This method required no changes to the source code, which meant treating the services as black boxes and abstracting from their internal performance interdependencies.

The granularity of the WPM (i.e., the minimal interval between measurements) is 1 second, which is too long to measure the resource demands for individual requests to the system. However, if the resource utilization U_i for a particular component i and the throughput X_j of the system for a particular use case j is known, the resource demands $S_{i,j}$ for a component i in use case j can be determined using the service demand law [25], which is stated as:

$$S_{i,j} = \frac{U_i}{X_j}$$

The employed load drivers controlled the number of requests issued to the system per time interval and recorded the number of items received per second.

Successively letting more clients request items from the server processes in parallel stressed the system to determine its limits. 1 to 10 load drivers ran in parallel. Each load driver subscribed for several thousand items with a subscription rate of one second and thus emulated an operator workplace. For higher numbers of load drivers, the throughput measurement sunk beneath the number of requested items, as the system was not able to handle the workload. In another measurement experiment, the system wrote up to 10000 log files to store sensor data persistently and to stress the hard disks. Some large customer installations require such an amount of log files.

Fitting a linear function in dependency of the request arrival rates to the measurement data allowed modeling the CPU demands of two of the performance-critical use cases. For the other use cases, we assumed fixed CPU demands as we did not intend to vary the arrival rates in these cases. Using linear interpolation a

HDD resource demand function was derived depending on the use case request arrival rate.

In total, the system-wide performance monitoring resulted in 25 resource demands. The concrete resource demands as incorporated in our Q-ImPrESS qos-annotation model have to remain confidential. Our model abstracted from main memory accesses, network transfers, and virtualization overheads.

Model validation To assure that our model reflected the current performance of the system well, we calibrated it (i.e., iteratively refined it) until the prediction error from the tools was under a given threshold. In order to do so, the SAMM2PCM transformation mapped the annotated ABB PCS Q-ImPrESS model into a PCM instance. The PCM tools then allowed us to conduct simulations using SimuCom and numerical analyses using LQNS. We refined the resource demands until the error between measured and predicted response times was below 40 percent in all cases.

Additionally, we checked the performance metrics predicted by the model against performance measurement available from regular integration tests of the system. We also confirmed the plausibility of our Q-ImPrESS performance model by several PCS experts.

Predictions We conducted performance predictions for all the evolution scenarios described in Section 4.2. The first prediction targeted the *usage profile evolution* scenario. In the ABB PCS SAMM instance, usage profile changes can predominantly be expressed by shorter arrival rates for the different usage scenarios in the usage model. Thus, we increased the arrival rates of usage scenarios 1 and 2 until one of the underlying resources became saturated (Utilization $> 99\%$). For the other usage scenarios, it is not expected that their workload will increase significantly in the future, thus they remained constant.

Fig. 6 shows the throughputs and CPU utilization predicted by SimuCOM for the usage scenario 1 and an increasing number of calls per second. The CPU is saturated at a workload of 290 calls per second (i.e., an arrival rate of 0.0034) when the utilization reaches 99.929%. The throughput curve shows that the system cannot handle each request in time beyond this point, the CPU is the bottleneck.

The figure also shows the measured CPU utilization and throughputs (black lines) besides the predictions by SimuCOM (grey lines). We did not measure response times in our measurements because these values could not be readily obtained from our load drivers and they were less relevant in our study. We also did not record

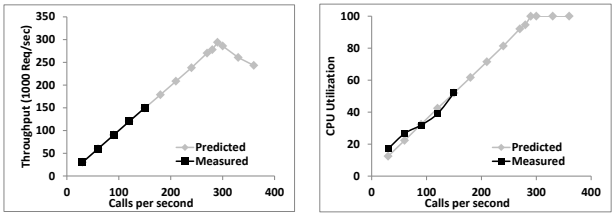

Fig. 6 Performance prediction results for different workload levels (usage profile evolution)

utilizations and throughputs for workloads higher than 150 requests per second because our load drivers disturbed the measurements above this workload level. Predictions above this level remain unvalidated. However, in practice the system is never run above the validated CPU utilization levels to avoid safety critical conditions. Furthermore, the following ranking of evolution alternatives does not change because of this.

The next performance prediction concerned the *resource environment evolution*. The Q-ImPrESS SAMM hardware and target environment allows changing the processors clock frequencies as well as latencies and throughput for hard disks and network devices. As the measured use cases were CPU and hard disk bound, we altered the processor clock frequencies (factor of 0.75, 1.0, 1.25, and 1.5 for the default clock frequency) and hard disk read/write speeds of the ABB PCS SAMM instance and analyzed the impact on throughput and utilization. The impact is not trivial (e.g., 1.5 times the throughput for 1.5 times the clock frequency) because of resource contention and the background resource demands.

Fig. 7 shows the CPU utilizations and throughputs for the different clock frequencies. If the clock frequency of server 1 is increased, the CPU utilization decreases (Fig. 7, upper left) and higher workload levels can be reached, before the CPU is saturated.

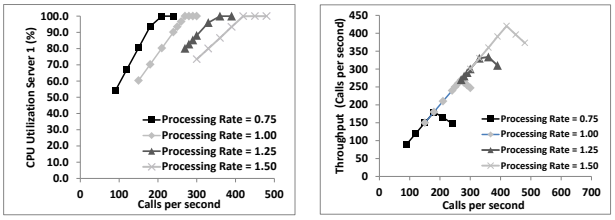

Fig. 7 Performance prediction results for different CPU clock speeds (resource environment evolution)

To analyze the *allocation evolution* scenario we modeled three default system sizes as Q-ImPrESS alternatives:

– Small size: all components are allocated to a single server, there is only one instance per component
– Medium size: the components are allocated to three servers according to a fixed scheme (Fig. 5), there is only one instance per component
– Large size: the component are allocated to six servers according to a fixed scheme (Fig. 8); for five components there are three instances each instead of only one instance; load balancers distribute the user requests evenly among these instances

Fig. 9 compares the performance predictions of the large allocation size with the small and medium size. The large allocation size allows a maximum throughput of approx. 800 calls per second, which is 207.7% higher than the medium size and 280.9% higher than the small size. Notice that due to the triple replication of the component stressing the hard disk to different nodes, the HDD utilization can be reduced from approx. 90% to approx. 30% (Fig. 9, upper left). The HDD utilization increases unexpectedly after a certain threshold, which may be caused by the CPU being overutilized thus confusing the simulation.

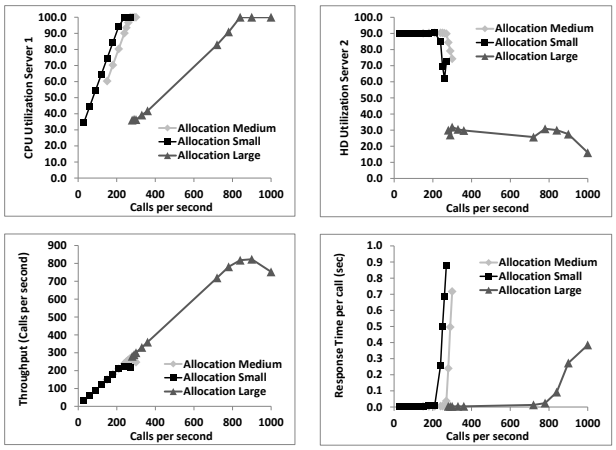

Fig. 9 Performance prediction results for different deployment sizes (allocation evolution)

Finally, we investigated the *component evolution* scenario. We modeled two alternatives for one component and one alternative for another component and estimated the change resource demands and failure probabilities. For example, dual-core exploitation in newer versions of a component can optimally lead to a response time reduction of 50%.

Fig. 10 depicts the performance predictions for the system when replacing the respective components. The lower CPU demands of the two alternatives for component 1 have almost no effect on the CPU utilization or maximum throughput. However, the HD utilization varies significantly for these variants (60% and 37% compared to 45% in the default variant).

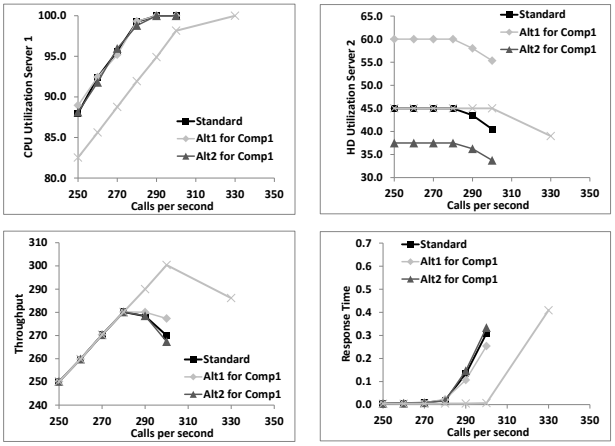

Fig. 10 Performance prediction results for alternative components (component evolution)

The highest impact has the alternative for the second component. It enhances the maximum throughput to approx. 300 calls/sec compared to approx. 280 calls/sec for the default variant.

5.2.2 Results

Our research questions Q1 and Q2 ask for the accuracy of the predictions and the effort required to obtain these predictions.

Question 1 (Performance Prediction Accuracy) We discuss the outcome of the metrics M1.1-M2.4 (Tab. 3) in the following. To assess the accuracy of our performance model we analyzed the mean percentage of deviation between CPU utilizations and throughputs for two different usage scenarios, which allowed us to reduce the distortion caused by single usage scenario.

Tab. 4 - 7 show the measured values, the predictions using SimuCom and LQN, as well as the error of the predictions. The CPU utilizations and throughputs are provided for different workload levels, i.e., for increasing numbers of requests per second. For the first usage scenario, the CPU utilization varies between 17 and 51 percent, while the throughput varies from 30 to 150 requests. The deviation in the simulated CPU utilization is higher than for the throughputs. For the second usage scenario, the CPU utilization varies between 19 and 22 percent, while the throughput varies from 3000 to 10000 requests. As for the first usage scenario, the deviation in the simulated CPU utilizations is higher than for the throughputs.

The mean response time deviation between measured and predicted values was 16.2 percent (=M1.1)

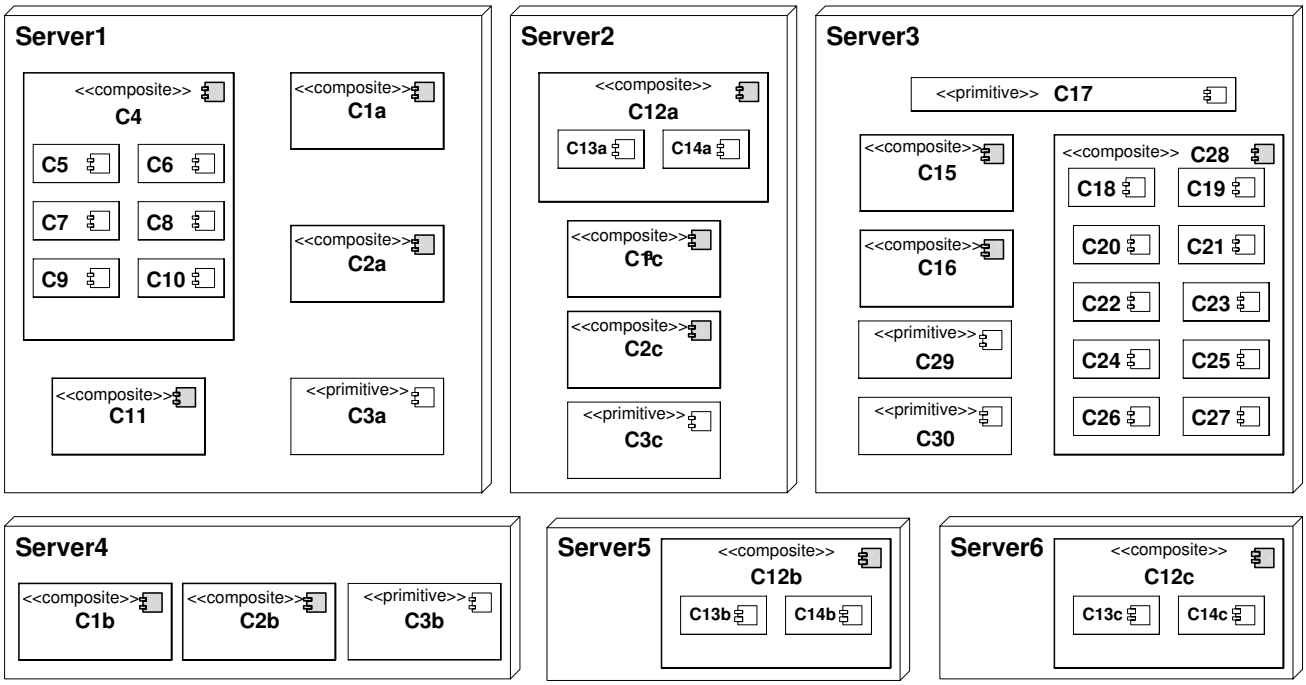

Fig. 8 Large size allocation evolution of the ABB ICS (UML deployment diagram)

	CPU Utilization				
Workload (Req./Time)	Measured (%)	Predicted			
		SimuCom	Error (%)	LQNS	Error (%)
30	17.146	12.467	27.288	12.464	27.305
60	26.681	22.366	16.174	22.343	16.260
90	31.902	32.347	1.395	32.322	1.317
120	39.016	42.432	8.754	42.329	8.490
150	51.929	51.943	0.027	51.760	0.326

Table 4 Prediction accuracy: CPU utilization, predictions vs. measurements, usage scenario 1

	Throughput				
Workload (Req./Time)	Measured (Req./Time)	Predicted			
		SimuCom	Error (%)	LQNS	Error (%)
30	30.000	30.059	0.197	30.030	0.100
60	60.000	59.930	0.117	59.880	0.200
90	90.000	90.154	0.171	90.090	0.100
120	120.000	120.549	0.458	120.482	0.402
150	150.000	149.326	0.449	149.254	0.497

Table 5 Prediction accuracy: throughput, predictions vs. measurements, usage scenario 1

	CPU Utilization				
Workload (Req./Time)	Measured (%)	Predicted			
		SimuCom	Error (%)	LQNS	Error (%)
3000	19.090	14.005	36.312	13.992	36.438
4000	19.315	14.759	30.866	14.754	30.911
5000	17.146	15.521	10.468	15.516	10.503
10000	22.241	19.336	15.023	19.322	15.107

Table 6 Prediction accuracy: CPU utilization, predictions vs. measurements, usage scenario 2

	Throughput				
Workload (Req./Time)	Measured (Req./Time)	Predicted			
		SimuCom	Error (%)	LQNS	Error (%)
3000	3.000	3.010	0.333	3.000	0.000
4000	4.000	4.010	0.250	4.000	0.000
5000	5.000	5.010	0.200	5.000	0.000
10000	10.000	10.009	0.090	10.000	0.000

Table 7 Prediction accuracy: throughput, predictions vs. measurements, usage scenario 2

over both usage scenarios. For smaller workloads there was a deviation up to 36.3 percent, whereas for higher workloads the error decreases down to approx. 1.4 percent. Nevertheless, the average value is below our formerly state hypothesis of 30 percent.

The reasons for the found deviations of the model lie in the simple regression model and the model's assumptions. A more sophisticated multi-variate regression model could have reduced the error. Furthermore, incorporating the virtualization or memory overheads in the model could have led to a more accurate model. Longer running measurements could have reduced the error as well. However, a highly accurate model costs more effort and is not needed to evaluate the evolution scenarios, because it does not affect the ranking of evolution alternatives. The predictions by the calibrated model appeared plausible to us and were confirmed by several PCS experts.

The throughput deviation between measured and predicted values was 0.2 percent (=M1.2) over both us-

age scenarios. The low deviation was to be expected because we measured the system only for workloads not saturating the resources, and thus the system was able to handle all requests without inducing queuing delays. We were not able to validate the throughput predictions for higher load levels because our workload drivers impacted the measurements in those cases.

In summary, our findings supported our hypotheses for M1.1 and M1.2 and we were not able to reject these hypotheses. We deemed the models accurate enough to predict the performance for the evolution scenarios. However, we did not validate the predictions for the evolution scenarios. This would have required implementing the alternatives and was beyond the scope of our study.

Question 2 (Performance Prediction Effort) The outcome for the metrics M2.1-M2.4 can be found in Tab. 8. The table includes the actual efforts (post-mortem estimation) of the activities for the ABB case study in the second column, which sum up to 126 hours. These efforts were inferred from the project accounting software used at ABB. We rejected the hypothesis of M2.1 (effort less than 8 person hours) as well as M2.4 (effort less than 2 person hours) for all activities. Our effort estimations from the beginning of the study can thus be considered much too low.

Performance Analysis	**Effort in person hours**			
	ABB	Estimation		
Activity	Actual	Optimistic	Realistic	Pessimistic
M2.1 Select data collection method	60	4	8	40
M2.2 Data collection / Performance Measurements	37	9	16	56
M2.3 Data Analysis	24	6	32	52
M2.4 Initial Predictions	5	5	9	14
Sum:	126	24	65	162

Table 8 Efforts for Q-ImPrESS performance prediction activities

However, the efforts for the ABB case study were influenced by some factors specific for our project. Some personal with needed information was not readily available. Additional domain knowledge needed to be acquired. Other projects interfered with the performance modeling activities, and the Q-ImPrESS tools were still in a maturation stage leading to time-consuming bug fixes. Therefore, the efforts spent in our study cannot be directly transferred to other cases.

We thus decided to re-estimate the efforts excluding the interfering activities to get a fair estimate for future projects. For these estimations, we made the following assumptions:

- The user of the method is familiar with the system under study.
- Source code and documentation of the system under study is readily available.
- The Q-ImPrESS tools are free of defects.
- Evolution scenarios have been designed and discussed and non-functional requirements are documented.

Even if these assumptions are taken into account, there is still a high potential for variability in the efforts. This depends on whether facilities for data collection (e.g., performance testbeds or bug tracking systems) are readily available or need to be set up. It also depends on how complex the desired evolution scenarios are and on how accurate the prediction results are expected to be. Thus, we provide not a single potential effort value in persons hours, but a distribution expressed in the optimisic case, realistic case, and pessimistic case in Tab. 8 column 3-5.

According to these estimation the likely effort for all activities is approx. 65 hours, while the pessimistic case effort can rise up to approx. 160 hours. Both values are well above the initially expected effort of 26 hours. The largest portion of the efforts is spent in conducting the performance measurements and analyzing the resulting data. Respective performance experiments need to be designed to derive the resource demands needed for the Q-ImPrESS models. Opposed to our expectation this is an iterative process, where the initial experiments are refined and repeated to achieve sufficiently accurate resource demands.

5.3 Reliability Prediction

To conduct a reliability prediction, we annotated the PCS Q-ImPrESS model with component failure probabilities. After validating the model and obtaining the overall reliability, we conducted a sensitivity analysis.

5.3.1 Data Collection

Measurement Determining component failure probabilities is more complicated than measuring component resource demands as there is limited guidance and no standard tool. Methods for reliability prediction are still being researched as discussed in Section 2. We thus first had to conduct a literature search on suggested methods for obtaining component failure probabilities. As the aspect of data collection is critical also for third-party applications of model-based reliability analysis methods, we briefly discuss the benefits and drawbacks of the found methods in the following (also see [17,13, 24,8]):

- **Defect prediction based on code metrics [11, 43]:** compute code metrics, such as lines of code, inheritance depth, or cyclomatic complexity to estimate the number of component defects (e. g., four defects per 1 KLOC [14]). Given source code this method is easy to execute, but its validity is not proven [43] and even debated in literature [11].
- **Reliability growth modeling [42,23]:** assume that software reliability grows over time due to bug fixes and extrapolate curves of field failure report dates to predict future failures using statistical regression. Many software reliability growth models (SRGM) are available from literature [42]. However, this method is reasonably applicable only on already completed or almost completed software. Applying the method on individual components is often not possible because of limited failure reports.
- **Random/statistical testing [39]**: generate random test data for individual components and incorporate the number of successfully executed tests into a statistical failure rate estimation model. This method is applicable to any type of software and does not require source code or historical data. However, the effort for generating and executing a sufficient number of test cases is high and the method might not scale. Because of inter-component relationships, it is difficult to test components in isolation.
- **Fault injection [18,14]:** manually insert faults into the source code or use test cases from fixed bugs on former versions of the software. The failure rates can then be estimated as the number of failed vs. the number of successful test case executions. This method is accurate for former versions of the software and the effort can be low if suitable test cases are available. However, this method does not determine the *current* component failure rate. Additionally, it is often difficult to attribute test case failures to component faults [15].
- **Explicit failure modeling [8]:** construct a state-based behavior model per component explicitly including manually specified transition probabilities for failure states. To create such a model, user requirements, domain knowledge, and/or experience with similar software can be used. While this method is useful for newly developed components, it requires manual estimation of failure rates and its accuracy is not proven.

Users of model-based reliability prediction methods have to select one of these methods based on the available data (e.g., code metrics, test coverage metrics, or failure reports). In our case study, we decided to construct a SRGM based on failure reports from a bug tracking database of the ABB PCS to calculate the failure probabilities for the PCS [31]. We used the IEEE Recommended Practice on Software Reliability [23] to guide our data collection.

ABB systematically records all problems that a software product experiences during its entire life cycle in a bug tracker database. Thus, from our point of view, a failure occurs when the developer and/or the end user reports a failure. The ABB PCS bug tracker database includes issue reports with different types of severity. For each report, it includes title, description, status, action performed (e. g., change applied, duplicate, forwarded), affected subsystem, and further information. Due to the availability of failure data per subsystem, we constructed a SRGM for each subsystem to estimate the failure probabilities.

We only accounted failures that were fixed and assumed that the corresponding defects causing the failures were located in the same subsystem because we lacked further data about which parts of the code have been updated to fix a bug. Note that this simplifying assumption might have introduced an inaccuracy into our model [15].

We filtered the failure reports from the bug tracker database according to the following criteria. We selected only one release, and for this release, we selected only "critical" and "high" severity failures, which are defined similar to Severity #1 and #2 in [23]. Failures in these two categories cause downtime that affects the overall availability of the system. In order to comply with the assumption that "faults are immediately removed when failures are observed" [23], we selected only those failures for which a change was applied.

Another assumption of SRGM models is that the component being modeled is somehow "stable", i. e., it can run for *some* time before failing. This means that compilation and crude execution errors have been already eliminated during testing. Thus, only subsystem failure records where the failure submit date is greater than or equal to the system release date were selected.

To calculate the failure probabilities of the subsystems, we searched the IEEE Std. 1633-2008 for a suitable SRGM. To reduce the complexity, we decided to use a single type of SRGM for all subsystems, which we selected according to industry affinity of former applications as suggested in IEEE Std. 1633-2008. The Littlewood/Verrall model [33] was developed from a large SCADA/DCS system (Supervisory Control and Data Acquisition / Distributed Control System), which controlled power stations in the UK and provided similar services as the ABB PCS. It had an a development effort of approx. 17 person years and could therefore also be considered a large-scale system.

Additionally, the LV-model and is the only calendar time between failure (TBF) SRGM that accounts for both operational and imperfect fault removal uncertainty, that is, fixing a bug can introduce new bugs. The model also exhibited good fit to TBF data during the preliminary exploratory analysis and was therefore selected for further modeling.

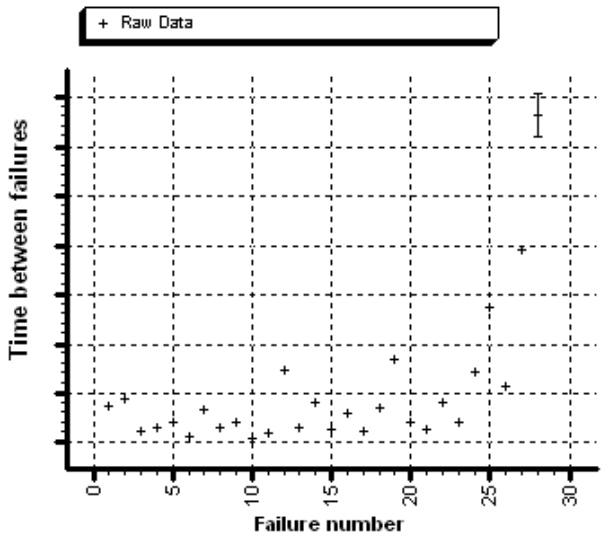

Fig. 11 Time between failures for one subsystem of the ABB PCS

We chose the 'Computer Aided Software Reliability Estimation' tool (CASRE) recommended in [23] to perform our failure rate estimations. Fig. 11 shows a CASRE plot of the critical and high failure history of the PCS. Notice that the unit of the time between failures has been intentionally obfuscated for confidentiality reasons.

We were able to fit the whole dataset without filtering data at 5% significance level with the quadratic Littlewood/Verrall model (LV-Q). Fig. 12 shows a plot of the failure intensity. Finally, the current failure intensity estimated by the model is the value used to annotate the failure probability of the respective component. We applied the same procedure to all processes of the ABB PCS.

Finally, we annotated the composite components in the manually built Q-ImPrESS model, which represent subsystems, with the failure probabilities of the corresponding subsystems using the QoS annotation editor. The primitive components did not get annotated with failure probabilities and are therefore assumed to be perfectly reliable, which is reasonable for the scope of our study.

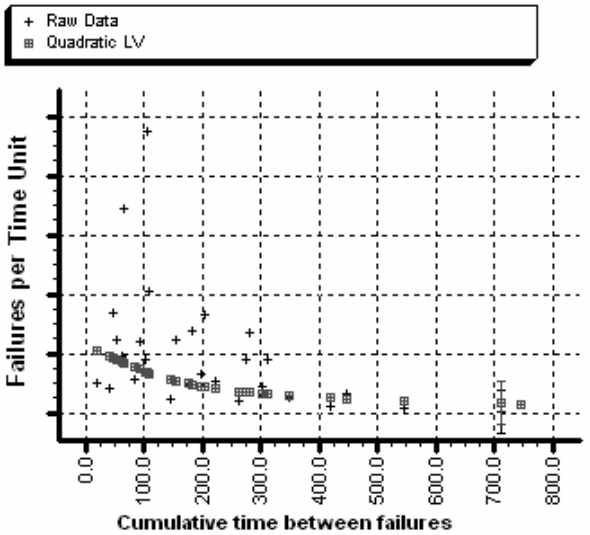

Fig. 12 Failure intensity of a subsystem of the ABB PCS showing the original points and fitting curve

Model Validation To get a first hint of the plausibility of the subsystem failure probabilities predicted by the LV model, we searched for a correlation between code metrics and failure probabilities [43,54]. We compared the subsystem failure probabilities and the arithmetic average cyclomatic complexity per method [38], which had been used in former studies.

Spearman's rank correlation coefficient ρ is moderately high for average cyclomatic complexity vs. the failure rate ($\rho = 0.6428$, $p = 0.1389$). The slight correlation between complexities and failure probabilities gives us some confidence that the failure probabilities predicted by the LV model are indeed representative for the current failure probabilities of the system.

To further confirm the validity of the model, we compared the results of the reliability analysis done with KLAPER/PRISM from Q-ImPrESS with a reliability analysis of a plain Markov model done without the Q-ImPrESS tools. The deviation between the two results was 7%. Given that the grade of abstraction was slightly different in both models, this deviation is not too high. Therefore, we are confident that the Q-ImPrESS model is valid.

Predictions Opposed to our expectations in the beginning of our case study, the Q-ImPrESS tools were not capable of making reliability predictions for most of the evolution scenarios described in Section 4.2.

The usage profile evolution scenario cannot be assessed for reliability because higher workload levels are not taken into account by the Q-ImPrESS reliability solver. The models only include a probability distribution for the system-level usage scenarios. Higher workloads (e.g., calling functions with the same probabilities

more often in a given time interval) generally do not lead to more user perceived failures. The situation that the system rejects a request because of overload is currently not reflected in the reliability models. However, if we assume that the performance prediction states for a higher workload that the system will not be saturated and if the probabilities of calling functions at the system level do not change, we can assume that the probability of failure on demand does not change. Notice that this assumption neglects failures for example induced by undersized data structures.

The resource environment evolution scenario (e.g., using faster processors) can also not be assessed because the processing rates by Q-ImPrESS resources are ignored by the reliability solver. In general, faster resources should not introduce failures per se. Increasing the number of cores could have an influence on reliability, because parallel threads could run into synchronization deadlocks. However, this is also not reflected by the current models. This scenario could also include replacing hardware resources with high availability resources (e.g., RAID disks). However, Q-ImPrESS currently only analyzes software performance in dependency to hardware and cannot take the reliability of hardware resources into account. To account for hardware reliability, the failure probabilities of the software components would have to be adjusted manually and the prediction would have to be run again.

Because the Q-ImPrESS tools do not take hardware resources, such as processors, hard disks, and networks, into account during reliability prediction, the allocation evolution scenario can also not be evaluated. Changing the deployment has no impact on the probability of failure on demand. In practice, the network communication could be a source for failures and thus should be reflected in the reliability model. Furthermore, typical redundancy schemes (e.g., stand-by nodes or replicated networks) are used in practice to address reliability issues and should be incorporated into the models in the future.

The only evolution scenario that we could analyze with the Q-ImPrESS tools for its reliability impact was the component evolution scenario. As in former reliability analysis studies [49,18], we decided to conduct a sensitivity analysis with our reliability model to simulate replacing the components with more reliable ones.

The sensitivity analysis in the Q-ImPrESS tools is conducted by manually varying the failure probabilities of a single component and executing the reliability analysis. With this procedure it is possible to determine the components mostly contributing to the overall reliability, thus predicting impact of more testing or component replacement.

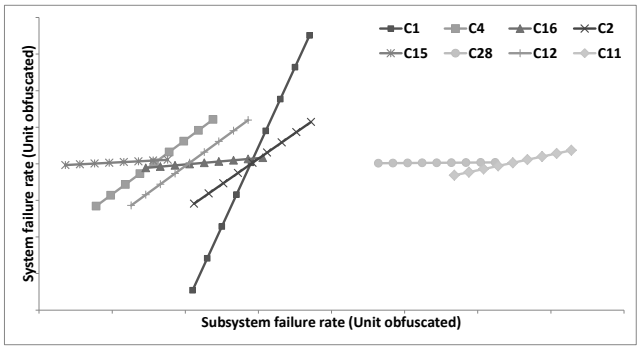

Fig. 13 Sensitivity analysis: system failure rate varies differently for individually changed subsystem failure probabilities (units obfuscated)

The result of the sensitivity analysis of our model is shown in Fig. 13. The actual failure probabilities have been obfuscated in the figure for confidentiality reasons. The curves show the failure probabilities for different subsystems in the ABB PCS, which are expressed as composite components in Fig 5. The central point of each curve is the value determined from the SRGM, while the other points have been created by predictions with the Q-ImPrESS model.

Subsystems C28 and C11 have the highest failure probabilities (on the far right side), while subsystem C15 has the lowest failure probability (on the far left side). This was already a result of our SRGM models for the respective subsystems. The slopes of the curves are a measure for the sensitivity of the system failure probability to the subsystem failure probability.

Subsystem C1 is the most sensitive for the system reliability, because its slope in Fig. 13 is the highest. That means replacing subsystem C1 would have the highest impact on reliability, which appears plausible because it is responsible for processing most of the data in the PCS and is called most often. Subsystem C28, which is used by other subsystems, does not contribute much to the overall system reliability. Compared to other subsystems, this subsystem is called only a limited number of times and therefore has a limited impact on system reliability. For subsystem C11, we had estimated the highest failure probability, but it is in fact also only a minor driver for system reliability because of the usage profile.

To validate the results of the sensitivity analysis, we compared them to results of a sensitivity analysis done on the Markov model. The average deviation of the slopes of the two models is 1.3%. It is to be noted that the deviation of some slopes is quite high (approximately 85%). This stems from the fact that the abstraction level of the two models is slightly different.

The order of the different subsystems, however, stays the same.

5.3.2 Results

In the following, we discuss the outcome of the metrics for research question Q3 (reliability prediction accuracy) and Q4 (reliability prediction effort) from Tab. 3.

Question 3 (Reliability Prediction Accuracy) M3.1 ask for the mean percentage of deviation between predicted and measured probability of failure on demand. As we were not able to evaluate multiple evolution scenarios as discussed before this accuracy would have to rely on our initial Q-ImPrESS reliability model. However, opposed to performance analysis, it is much harder too measure the actual reliability (i.e., the probability of failure on demand) of a running system, which is necessary for the comparison with the predictions.

Different approaches to measure reliability have been proposed (Section 5.3.1). None of these approaches however can convincingly provide the respective measures in our case. Code metrics provide an indicator, but are no reliable evidence for potential failures [43]. Statistical testing is not applicable in our case because the number of test case to execute would be prohibitively large.

Another option is to perform explicit failure modeling and to construct a simulation model (e.g., as done by Brosch et al. [7]), which emulates the exceptions thrown by a system and thus allows for a much quicker estimation of the probability of failures on demand. The simulation results can then be compared to the prediction results of the analytical model. However, the simulation model is again based on certain assumptions and creates an abstraction from reality, and thus it also does not reflect the actual reliability accurately.

Based on these observations, we could not determine a value for M3.1 within the timeframe of our project. This is an inherent problem for architectural reliability models, whose predictions are typically not compared to actual reliability measurements [17,13,24]. Further empirical research for example in longitudinal studies is required. Failure reports need to be tracked for several years to get meaningful data to determine the actual realiability of a system.

To gain some trust into our model we performed the model validation activities described before. Furthermore, the sensitivity analysis of our model helps to assure that failure probabilities estimated with uncertainty do not impact the predicted POFOD in an unexpected manner.

Question 4 (Reliability Prediction Effort) The outcome for metrics M4.1-M4.4 can be found in Tab. 9. We apply the same approach as in Section 5.2 and provide both post-mortem estimations for our study and optimistic/realistic/pessimistic estimations for third-party applications of Q-ImPrESS. In total, we needed approx. 120 person hours for the reliability predictions, which is near our pessimistic case estimation of 126 person hours.

Reliability Analysis	**Effort in person hours**			
	ABB	Estimation		
Activity	Actual	Optimistic	Realistic	Pessimistic
M4.1 Select data collection method	56	4	16	40
M4.2 Data Collection / Reliability estimations	48	10	14	52
M4.3 Data Analysis	8	3	4	16
M4.4 Initial Predictions	12	5	11	18
Sum:	124	22	45	126

Table 9 Efforts for Q-ImPrESS reliability prediction activities

As for the performance predictions, the values for M4.1-M4.4 are well above our initially expected effort of 26 hours. Other than for performance analysis, there are only limited empirical studies in literature dealing with reliability estimation of realistic systems. Thus, it is difficult to select an appropriate method for data collection and even to get aware of the different possibilities for data collection. We first had to conduct a literature survey to get an overview of methods and tools for reliability data collection as no step-by-step guide was available. Furthermore, we spend a considerable amount of time to select an appropriate SRGM.

The data collection itself can be highly variable depending on the chosen method (e.g., reliability growth modeling or fault injection). In our case, we needed to get access to the bug tracking system and analyze the bug reports to construct a model. Data analysis involves validating the data, which we attempted via complexity metrics. The initial prediction can be carried out quickly, but the sensitivity analyses require further tool automation.

5.4 Tradeoff Analysis

The following section describes applications of the two alternatives for the tradeoff analysis in Q-ImPrESS, i.e., design space exploration with PerOpteryx and alternative weighting with AHPWizard. The former alternative is suited for generating a large number of architectural candidates while the latter supports manually trading off a handful of candidates. We did not include the tradeoff analyses accuracy and efforts into our GQM schema because these studies had a more exploratory

character and were not planned at the start of our investigations. Nevertheless, the trade-off analyses are an integral part of Q-ImPrESS, and we report our findings in a descriptive manner, which might be helpful for third party application.

5.4.1 Design Space Exploration with PerOpteryx

Application: We applied the Q-ImPrESS PerOpteryx tool [35] to search for optimal performance and costs tradeoffs in the ABB PCS. It requires the architect to specify degrees of freedom (i.e., variation points with multiple alternatives, examples see below) in the architectural model using an EMF model. From this model PerOpteryx generates an initial set of architectural candidates. Afterwards, it executes the Q-ImPrESS prediction tools (i. e., LQNS and DTMC solver) on each candidate. Based on the results, the tool selects promising candidates (e.g., with low costs and short response times). It then reproduces new candidates from this selection by performing crossover and mutation (e.g., taking component allocations from one candidate and CPU speeds from another candidate). This process is repeated for a pre-specified number of times. The result is a Pareto curve of architectural candidates with optimal QoS tradeoffs.

In our case study, we specified the following degrees of freedom: reallocating the services to different servers, varying the number of servers, varying the processing rates of CPUs, and integrating several alternative services, which we had manually added to the model (e.g., a faster component with higher costs). The costs of the CPUs depended on their processing rate and had been determined by fitting a power function to Intel's CPU price list.

We generated 10 starting populations for the ABB PCS model (i.e., initial selection of components, allocation, and CPU speeds) and performed 10 independent runs of PerOpteryx each lasting 5-6 hours on a standard PC. The replication helps to reduce distortions from the evolutionary algorithm. The tool evaluated around 2000 candidates per run and found 330 Pareto-optimal candidates in total (Fig. 14).

One example candidate exhibited a cost reduction by 23 percent while the response time was increased by 19 percent, which is tolerable within customer requirements. In this case, PerOpteryx suggested allocating all services to a powerful, single-server node thus saving costs for other nodes. This candidate is similar to an actual configuration sold to smaller customers based on rule of thumb.

Findings: The application of the PerOpteryx tool is an interesting alternative to the current practice of sizing and configuring systems by rule of thumb, which can lead to expensive overprovisioning. Future customer recommendations could be substantiated by model-driven prediction results. However, the validity of the generated candidates (i.e., whether the predicted performance can be perceived in a running system) still needs to be proven, which was not attempted here for time constraints.

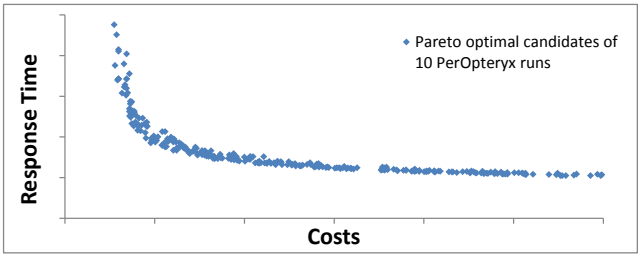

Fig. 14 Tradeoff analysis results from PerOpteryx: response time vs. costs for the ABB PCS visualized as a set of unified Pareto front (units obfuscated)

The explorable degrees of freedom in the design are limited to architecture-level changes. The tool might generate architectural candidates undesirable for reasons not reflected in the model (e.g., security / safety constraints). The effort for applying PerOpteryx depends on the possible degrees of freedom to be modeled and the required validation of the resulting candidates, while the effort for running the tool is negligible.

5.4.2 Weighting Alternatives with AHP Wizard

Application: As a second alternative for tradeoff analysis, we applied the Q-ImPrESS AHP Wizard [21]. AHP was chosen in Q-ImPrESS because of promising results in former software architecture studies [61]. The tool operates on the Q-ImPrESS prediction result repository and requires selecting a set of alternative predictions for tradeoff analysis. Then it asks the user for weights (from -4 to +4) in pairwise quality attribute comparisons (e.g., response time vs. reliability). Afterwards, the tool lets the user specify weights (from -4 to +4) to pairwise comparisons of predictions results (e.g., response time alt1 vs. alt2). Finally, it calculates a score for each alternative based on the weights.

In our case study, we analyzed four (yet artificial) alternatives for one of the components in the ABB PCS system. The first alternative represents the current implementation of the component. In the second alternative, the component has a lower resource demand because it exploits multi-core CPUs but also a higher failure probability because of expected concurrency bugs. In the third alternative, the component comes from a third party and has a slightly higher resource de-

mand but a lower failure probability because it has been proven in long-term use. The fourth variant uses the same component as the first variant but deploys it to a dedicated server.

Using the Q-ImPrESS AHP wizard, we specified a high preference for reliability and provided 24 weights for the prediction result comparison. Fig. 15 depicts the resulting score. Alternative 3 has received the highest score because of its low failure probability. Alternative 2 is best for CPU utilization while alternative 4 exhibits the lowest response times.

Findings: The AHP method is established in business decision support and can help to quantify tradeoffs between multiple architectural alternatives. Benefits of the method in the Q-ImPrESS context are the good tool integration, visualization, and export functionality. Drawbacks are the limited traceability of the results and the inherently limited scalability of AHP. If n is the number of alternatives and q is the number of quality attributes under analysis, the method requires $\binom{n}{2} \cdot q$ pair-wise comparisons from the user, which yielded 24 comparisons in our study (4 alternatives and 4 quality attributes). For 10 alternatives, 180 comparisons would be required thus becoming impractical.

5.5 Evaluation of Validity

We describe threads to the validity of a our study in the following. We focus on construct, internal, and external validity.

5.5.1 Construct Validity

The construct validity states whether the studied persons, settings, and tools represent the intended constructs well [58]. The goal of our study was to evaluate the applicability of model-based quality prediction on a large-scale industrial software system from the viewpoint of the quality analyst.

First, we argue that Q-ImPrESS is a representative, state-of-the-art, model-based, quality prediction method. Q-ImPrESS is based on a number of former performance and reliability prediction approaches (e.g., Palladio [6], KLAPER [19]) and emerged as a recent joint effort by a combined academic and industrial consortium in a EU project. The used performance solvers SimuCOM and LQNS had been formerly applied in a number of other case studies. The reliability solver is based on a standard technique for solving DTMCs. Thus, Q-ImPrESS is based on methods that have been formerly peer reviewed and published in renowned conferences (e.g., WOSP) and journals (e.g., JSS).

Second, we argue that the ABB PCS is representative of a large-scale industrial software system. The analyzed part of the PCS consists of more than three million lines of code and are usually deployed in distributed server environments. The system can therefore be considered as large-scale. Additionally, the system has been sold to customers for several years and manages numerous industrial processes, such as power plants or oil refineries. It can therefore be considered as a real-world software system in industrial use.

Third, we argue that the investigators in this study are indeed representative for the targeted perspective of a QoS analyst. We had former experience with QoS analysis particularly in the area of performance modeling. A threat to the construct validity might be that we only had limited experience with reliability modeling of large scale software systems. To mitigate this threat, we conducted a comprehensive literature search on data collection techniques for reliability modeling [31] and had our study methodology reviewed by our academic partners. We also had limited experience with the system under study, which we tried to mitigate by involving PCS experts where necessary. This might not be a severe threat to construct validity because typical quality analysts are often external consultants [53].

5.5.2 Internal Validity

The internal validity states whether changes of a case's independent variables are in fact the cause for changes of the dependent variables [58]. In our case, the independent variables are the Q-ImPrESS method and the ABB PCS system while the dependent variables are the prediction accuracy and the required effort. The question is: was the achieved prediction accuracy and the required effort indeed caused by Q-ImPrESS and the ABB PCS or was another variable responsible?

There are several potentially distorting variables in our case study. The competence and the experience of the investigators might have influenced the achieved accuracy and required effort. While this variable could not be controlled with statistical significance (e.g., through replicated investigations) given the timeframe of our project, at least a second company applied the Q-ImPrESS method on their systems over the course of the Q-ImPrESS project [51]. As that investigation achieved comparable accuracy and required comparable efforts [29], we have at least an additional hint that the competence and experience of the investigators was a minor distorting factor.

As an additional threat to the internal validity, the fact that the Q-ImPrESS method and tools were still in development when the case study was conducted might

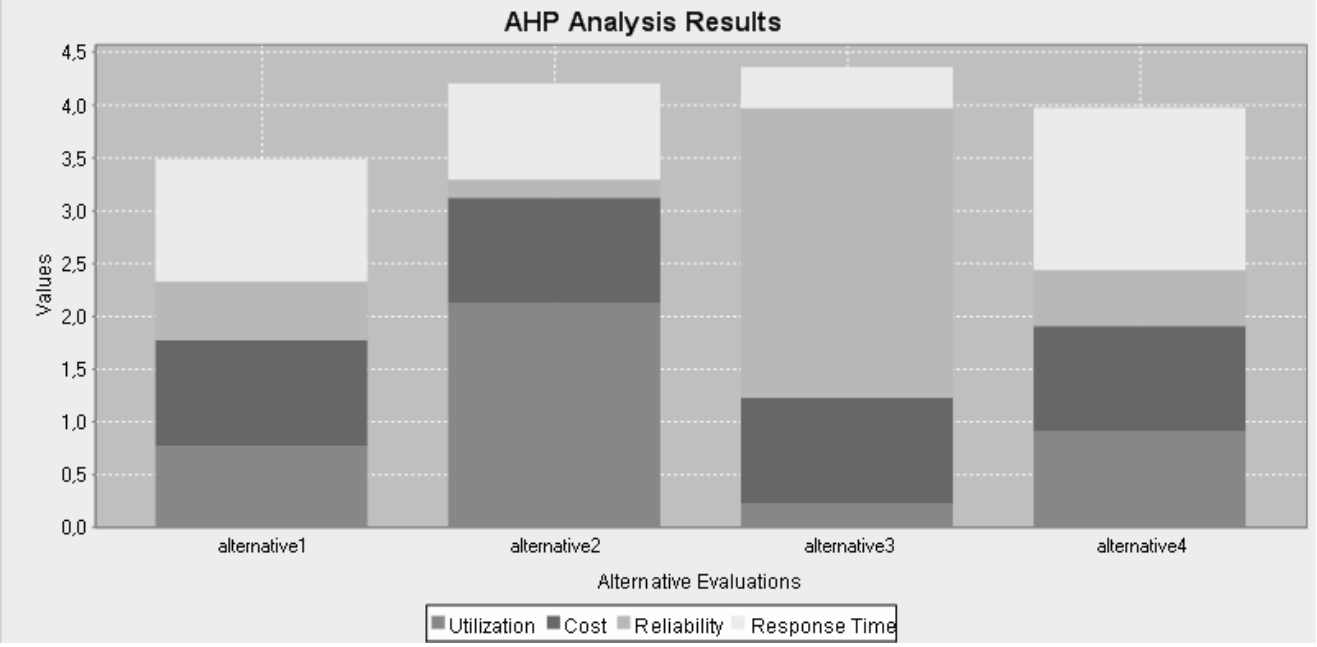

Fig. 15 Tradeoff analysis results from AHP Wizard: alternative 3 receives the highest score because of the preference for reliability

have a distorting influence. For example, delays in the case study occurred due to defects in the tools, which required time-consuming bug fixing and testing. The documentation of both methods and tools was incomplete thus unnecessarily extending the effort for applying the method. We have tried to mitigate this threat by providing effort predictions where these influences were removed (Section 5.2.2 and 5.3.2).

The accuracy of our prediction might be limited by the given timeframe of the project. More detailed modelling might have led to higher prediction accuracy. However, we argue that typical quality analysis projects in practice also operate under strict time and budget constraints, thus our setting was realistic.

The validity of the build model can be debated, i.e., whether it actually reflects the performance and reliability properties of the system under study. To mitigate this threat we have already discussed the validation of the transition probabilities, resource demands, and failure probabilities in Section 5.

5.5.3 External Validity

The external validity states whether the results of a study are transferable to other settings than the specific experimental setting [58]. The question to answer is: are the achieved prediction accuracy and required effort transferable to other methods as Q-ImPrESS and other systems as the ABB PCS?

We already discussed in Section 2 that a number of related studies reported on a performance prediction error of less than 30 percent for various performance measures. This demonstrates that comparable performance prediction accuracy as in our case study is possible for other systems. A similar indicator for reliability predictions is missing.

The results in our case study might be limited to the domain of industrial controls systems. However, it can be argued that the server-side part of the system is based on standard Microsoft technologies that are also used in other domains (e.g., enterprise information systems). It should be noted that other systems might have more intricate performance or reliability influences which would complicate modeling, lower prediction accuracy, and increase effort. For smaller systems an even higher prediction accuracy than in our case study could be potentially obtained with lower effort.

As discussed in Section 5.2.2 and 5.3.2, the effort for applying a model-based, quality analysis method depends on the availability of suitable data collection facilities (e.g., performance testbeds or failure report statistics). In other settings, the effort for data collection might be substantially different.

Whether other methods, which are similar to Q-ImPrESS, can be applied with similar effort and accuracy remains unclear based on our study. Influencing factors are the usability of the available tools and the expressiveness of the modeling notation.

6 Discussion

Reflecting on the experiences while applying Q-ImPrESS on the ABB PCS, we discuss modeling issues, process issues, and implementation issues in the following, which helps to better differentiate the findings of our case study and provides pointers for future work. Modeling and process issues need more research, while implementation issues can be overcome with better tools.

6.1 Modeling Issues

We first discuss issues related to the SAMM meta model of Q-ImPrESS, which codifies the required performance and reliability constructs necessary for the prediction. Q-ImPrESS offers a comprehensive meta model with more than 100 classes, which allows to model a large class of systems to analyze numerous performance and reliability issues. Opposed to many other performance models, the SAMM allows specifying resource demands with arbitrary distribution functions, which can lead to more accurate predictions.

The complexity of the SAMM however also requires a substantial learning effort (approx. 1 week). Q-ImPrESS models are more difficult to understand than simple queuing networks. This stems from the fact that Q-ImPrESS is specifically designed to analyze different evolution scenarios and therefore allows to quickly replace parts of the model (e.g., the QoS annotations or the resource environment) and to parameterize many parts of the model (e.g., user interactions and resource demands). It might therefore not be justified to apply Q-ImPrESS for single evolution scenarios or smaller systems. The higher effort for applying Q-ImPrESS pays off when the models can be reused for multiple evolution scenarios.

However, although comprehensive, the SAMM still lacks constructs to model many interesting evolution scenarios of modern SOA systems, e.g., direct support for virtualization, OS changes, application server configurations, transmission protocols changes, event-based communication, real-time scheduling, or dynamic architectures which bind services at runtime. There is also only limited support for modeling the middleware. It would be desirable if specifics of service-oriented systems would be better supported (e.g., dynamic adaptation by identifying and binding services at runtime or the comparison of transmission protocols, such as SOAP or REST [20]). Even asynchronous communication is currently cumbersome to model with the SAMM. The foreseen modeling constructs for multi-core processors, memory accesses, or cache configurations are currently unsupported by most of the analysis and simulation tools including those implemented in the Q-ImPrESS tools. However, notice there is a trade-off involved. Including more modeling concepts into the meta-model makes the model even more complex to learn for end-users. Also, developers of analysis methods built on top of the SAMM have to support a larger variety of concepts making it more difficult to ensure the correctness and validity of such methods.

The SAMM also has severe limitations in the area of reliability modeling. Specifying the reliability of hardware devices is not supported in the SAMM, which means that not only the hardware is irrelevant for reliability predictions, but also that different allocation configuration have no influence on the predicted reliability. Other approaches [7] support analyzing architectural models with hardware reliabilities, which could be integrated into the SAMM. The SAMM also provides no higher-level constructs to model common fault-tolerance mechanism, such as recovery blocks or design diversity (again, implying a more complicated meta-model as discussed before). More refined models and tools would be needed to evaluate the impact of different component topologies on the system reliability.

6.2 Process Issues

Q-ImPrESS comes with a comprehensive process documentation which describes the different steps to follow to execute the method (cf. Section 3.2 and [37]). While the process description offers guidance, it is usually not followed exactly during a particular application of the method as certain steps are irrelevant or require much more detail.

During our case study, the data collection for the Q-ImPrESS models (especially transition probabilities, resource demands, and failure probabilities) proved to be the hardest and most time-consuming activity. Another difficult activity is determining an adequate abstraction level for modeling. However, for both data collection and choosing an abstraction level the Q-ImPrESS process description offers no hints. Manual work and/or third party product use is required from the users of the method. Such manual work is common in today's performance and reliability engineering approaches due to a lack of standard tools for taking measurements and the high diversity of (component) frameworks and implementation platforms. Nevertheless, for popular frameworks (e.g., JEE or COM/.NET) future improvements could lead to such standards for these platforms.

While the Q-ImPrESS tools are well-integrated with each other, they do not easily incorporate into other de-

velopment environments than Eclipse. However, in industrial software development often commercial tools are used (e.g., Microsoft Developer Studio). Introducing additional tools in a predefined tool chain requires additional process considerations and may increase the needed effort. Additionally, Java as programming language is better supported by Q-ImPrESS than other languages, thus, reducing the effort to integrate the Q-ImPrESS method in a Java-based development process.

On the modeling side, existing UML models need to be recreated for Q-ImPrESS as there is currently no support for model transformations to SAMM. In the future, it may be possible to reuse (parts of) UML2 models based on the Eclipse UML2 project via a QVT-based model transformation.

6.3 Implementation Issues

The implementation of the modeling and analysis concepts in easy-to-use software tools is a major prerequisite to conduct industrial case studies and gain experience with a method. Q-ImPrESS offers a homogeneous tooling environment, which uniquely integrates multiple mature research tools, such as the Palladio Component Model, KLAPER, LQNS, PerOpteryx, and the AHP Wizard. The Q-ImPrESS workbench offers a number of graphical and textual model editors for the different SAMM parts to increase the usability and lower the effort for constructing performance and reliability models.

However, the tooling environment did not reach industrial maturity during the project and needs further stabilization. There are still several usability issues which unnecessarily complicate the creation of models. On the other hand, we have to keep in mind that the tooling has been developed as a joint effort in a publicly funded, joint research project. It is to be expected that a company dedicated to quality prediction modeling tools will be able to create a stable and usable tooling environment. Whenever such a stable tooling becomes available it would be interesting to renew the experience report presented here.

For performance prediction, Q-ImPrESS offers two expressive and matured performance solvers (SimuCom and LQNS), which can solve models in the class of extended queuing networks. These solvers offer a state of the art feature set in performance analysis, e.g., support for passive resources (e.g., semaphores) as well as arbitrarily distributed service times. By reusing these solvers, the Q-ImPrESS project did not contribute new analysis concepts in performance modeling, which would have required specialized solvers. However, some of the proposed extensions (as discussed in sections 6.1 and 6.3) to the Q-ImPrESS SAMM may require such specialized solvers in the future.

For reliability prediction, Q-ImPrESS provides a solver for DTMCs based on the PRISM probabilistic model checker. The solver supports only a limited number of model constructs in the SAMM. For example, concurrently executed threads modeling with fork actions or the acquisition and release of passive resources are not supported by the solver.

However, even simple quantitative reliability predictions are an improvement over the current prevalent practice of relying simply on a number of executed test cases and experiences. The results of the sensitivity analysis are useful to make future testing procedure more efficient by allocating more testing resource on the most sensitive components. This can also save future maintenance costs, which could however not be quantified within the timeframe of this project.

7 Conclusions and Future Work

This paper presented an industrial case study applying the state-of-the-art method Q-ImPrESS for model-driven performance and reliability prediction on a process control system from ABB, which consists of several million lines of code. We found that a reasonable performance prediction accuracy (error < 30 percent) can be achieved with Q-ImPrESS within 2-4 person weeks. The accuracy for reliability prediction remains unknown, because a long-term study collecting field failure reports after system evolution would be necessary. Nevertheless, the sensitivity analysis provided by the reliability prediction provided hints on how to improve test effort distribution. This paper has also described two methods for trade-off analysis between different quality attributes and discussed the modeling, implementation, and process issues of Q-ImPrESS.

The results of our case study are relevant both for practitioners and researchers. Practitioners are provided an estimation of the effort for applying model-driven quality prediction methods, which is useful for cost/benefit calculations and to justify the application of the methods towards management. Our experience documented in this paper should also help practitioners in collecting the data necessary for performance and reliability prediction. Researchers are provided a demonstration on how model-driven quality prediction methods perform when being applied on a large-scale industrial system. We have identified many directions for improving the methods, tools, and processes.

Besides more robust modeling and prediction tools, better support for data collection, and better process

integration, our study provides several other open research questions:

- Can our achieved accuracy and effort numbers be reproduced in replicated studies?
- How can performance measurement frameworks and performance modeling tools be better integrated?
- How to refine reliability prediction methods so that they support for example hardware or middleware reliability?
- How can reliability prediction models be validated without waiting years for sufficient failure data?
- How can service-oriented systems be better supported in quality prediction?
- How can reverse engineering and performance measurement tools be integrated to speed up data collection?
- Is it possible to construct model libraries, so that models can be reused in different contexts?
- Can other quality attributes (e.g., security, safety, usability) be tackled in a quantitative manner and integrated into this approach?

We are currently working on the first two questions. These and the other questions are pointers for future work and should be addressed in further empirical studies on model-based quality prediction.

Acknowledgements This work was funded within the Q-ImPrESS research project (FP7-215013) by the European Union under the Information and Communication Technologies priority of FP7.

References

1. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. IEEE Trans. Softw. Eng. **30**(5), 295–310 (2004)
2. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In: J.J. Marciniak (ed.) Encyclopedia of Software Engineering, 2 edn., pp. 578–583. John Wiley & Sons (2002)
3. Bause, F.: Queueing Petri Nets - A formalism for the combined qualitative and quantitative analysis of systems. In: Proceedings of the 5th International Workshop on Petri Nets and Performance Models, pp. 14–23 (1993)
4. Becker, S.: Coupled Model Transformations for QoS Enabled Component-Based Software Design. Ph.D. thesis, University of Oldenburg, Germany (2008)
5. Becker, S., Bulej, L., Bures, T., Hnetynka, P., Kapova, L., Kofron, J., Koziolek, H., Kraft, J., Mirandola, R., Stammel, J., Tamburelli, G., Trifu, M.: Q-ImPrESS Project Deliverable D2.1: Service Architecture Meta Model (SAMM). Tech. Rep. 1.0, Q-ImPrESS consortium (2008). URL http://bit.ly/bfSGrh. http://bit.ly/bfSGrh
6. Becker, S., Koziolek, H., Reussner, R.: The Palladio Component Model for Model-Driven Performance Prediction. Journal of Systems and Software **82**(1), 3–22 (2009)
7. Brosch, F., Koziolek, H., Buhnova, B., Reussner, R.: Parameterized Reliability Prediction for Component-based Software Architectures. In: Proc. 6th Int. Conf. on the Quality of Software Architectures (QoSA'10), *LNCS*, vol. 6093, pp. 36–51. Springer (2010)
8. Cheung, L., Roshandel, R., Medvidovic, N., Golubchik, L.: Early prediction of software component reliability. In: ICSE '08: Proceedings of the 30th international conference on Software engineering, pp. 111–120. ACM (2008). DOI http://doi.acm.org/10.1145/1368088.1368104
9. Cheung, R.C.: A user-oriented software reliability model. IEEE Trans. Softw. Eng. **SE-6**(2), 118–125 (1980)
10. Cortellessa, V., Marco, A.D., Inverardi, P.: Model-Based Software Performance Analysis. Springer (2011)
11. Fenton, N.E., Neil, M.: A critique of software defect prediction models. IEEE Trans. Softw. Eng **25**(5), 675–689 (1999)
12. Franks, G., Al-Omari, T., Woodside, M., Das, O., Derisavi, S.: Enhanced Modeling and Solution of Layered Queueing Networks. IEEE Trans. Softw. Eng. **35**(2), 148–161 (2009). DOI http://dx.doi.org/10.1109/TSE.2008.74
13. Gokhale, S.S.: Architecture-based software reliability analysis: Overview and limitations. IEEE Trans. Dependable Secure Comput. **4**(1), 32–40 (2007). DOI http://dx.doi.org/10.1109/TDSC.2007.4
14. Gokhale, S.S., Wong, W.E., Horgan, J.R., Trivedi, K.S.: An analytical approach to architecture-based software performance and reliability prediction. Perf. Eval. **58**(4), 391–412 (2004)
15. Goseva-Popstojanova, K., Hamill, M., Perugupalli, R.: Large empirical case study of architecture-based software reliability. In: Proc. 16th IEEE Int. Symp. on Software Reliability Engineering (ISSRE'05), pp. 43–52. IEEE Computer Society (2005)
16. Goseva-Popstojanova, K., Hamill, M., Wang, X.: Adequacy, accuracy, scalability, and uncertainty of architecture-based software reliability: Lessons learned from large empirical case studies. In: Proc. 17th Int. Symp. on Software Reliability Engineering (ISSRE'06), pp. 197–203. IEEE (2006)
17. Goseva-Popstojanova, K., Mathur, K., Trivedi, K.: Comparison of architecture-based software reliability models. In: Proceedings. 12th International Symposium on Software Reliability Engineering, 2001. ISSRE 2001, pp. 22–31 (2001)
18. Goseva-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. Perform. Eval. **45**(2-3), 179–204 (2001)
19. Grassi, V., Mirandola, R., Sabetta, A.: Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. J. Syst. Softw. **80**(4), 528–558 (2007). DOI http://dx.doi.org/10.1016/j.jss.2006.07.023
20. Happe, J., Becker, S., Rathfelder, C., Friedrich, H., Reussner, R.H.: Parametric performance completions for model-driven performance prediction. Perform. Eval. **67**(8), 694–716 (2010)
21. Hatvani, L., Jansen, A., Seceleanu, C., Pettersson, P.: An Integrated Tool for Trade-off Analysis of Quality-of-Service Attributes. In: Proc. 2nd Int. Workshop on the Quality of Service-oriented Software Systems (QUASOSS'10). ACM Digital Library (2010)
22. Huber, N., Becker, S., Rathfelder, C., Schweflinghaus, J., Reussner, R.H.: Performance modeling in industry: a case study on storage virtualization. In: Proc. 32nd

Int. Conf. on Software Engineering (ICSE'10), pp. 1–10. ACM (2010)
23. IEEE: Recommended practice on software reliability, ieee std 1633-2008. Tech. rep., The Institute of Electrical and Electronics Engineers, Inc. (2008)
24. Immonen, A., Niemelä, E.: Survey of reliability and availability prediction methods from the viewpoint of software architecture. Springer Software and System Modeling **7**(1), 49–65 (2008)
25. Jain, R.: The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley (1991)
26. Jin, Y., Tang, A., Han, J., Liu, Y.: Performance evaluation and prediction for legacy information systems. In: Proceedings of the 29th international conference on Software Engineering, ICSE '07, pp. 540–549. IEEE Computer Society, Washington, DC, USA (2007). DOI http://dx.doi.org/10.1109/ICSE.2007.64. URL http://dx.doi.org/10.1109/ICSE.2007.64
27. Kounev, S.: Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. IEEE Trans. Softw. Eng. **32**(7), 486–502 (2006)
28. Koziolek, H.: Performance Evaluation of Component-based Software Systems: A Survey. Performance Evaluation **67**(8), 634–658 (2010)
29. Koziolek, H., Doppelhamer, J., Weiss, R., Bilich, C., Schlich, B., Skuliber, I., Zemljic, M., Desic, S., Huljenic, D.: Q-ImPrESS Project Deliverable D7.3: Demonstrator Results Documentation. Tech. rep., Q-ImPrESS consortium (2010)
30. Koziolek, H., Reussner, R.: A model-transformation from the palladio component model to layered queueing networks. In: Proc. of the SPEC International Workshop on Performance Engineering (SIPEW'08), *LNCS*, vol. 5119, pp. 58–78. Springer (2008)
31. Koziolek, H., Schlich, B., Bilich, C.: A Large-Scale Industrial Case Study on Architecture-based Software Reliability Analysis. In: Proc. 21st IEEE International Symposium on Software Reliability Engineering (ISSRE'10), pp. 279–288. IEEE Computer Society (2010). DOI http://dx.doi.org/10.1109/ISSRE.2010.15
32. Koziolek, H., Schlich, B., Bilich, C., Weiss, R., Becker, S., Krogmann, K., Trifu, M., Mirandola, R., Martens, A.: An industrial case study on quality impact prediction for evolving service-oriented software. In: Proc. 33rd ACM/IEEE Int. Conf. on Software Engineering (ICSE'11) Software Engineering in Practice Track. ACM (2011)
33. Littlewood, B., Verrall, J.L.: A Bayesian Reliability Growth Model for Computer Software. Applied Statistics **22**(3), 332 (1973). DOI 10.2307/2346781
34. Liu, Y., Fekete, A., Gorton, I.: Design-level performance prediction of component-based applications. IEEE Trans. Softw. Eng. **31**(11), 928–941 (2005). DOI http://dx.doi.org/10.1109/TSE.2005.127
35. Martens, A., Koziolek, H., Becker, S., Reussner, R.H.: Automatically improve software models for performance, reliability and cost using genetic algorithms. In: Proc. 1st Int. Conf. on Performance Engineering (ICPE'10), pp. 105–116. ACM (2010). DOI http://doi.acm.org/10.1145/1712605.1712624
36. Martens, A., Koziolek, H., Prechelt, L., Reussner, R.: From monolithic to component-based performance evaluation of software architectures: A series of experiments analysing accuracy and effort. Empirical Software Engineering pp. 1–36 (2010). DOI 10.1007/s10664-010-9142-8. URL http://dx.doi.org/10.1007/s10664-010-9142-8
37. Masetti, M., Becker, S., Skuliber, I., Hauck, M., Kofron, J., Krogmann, K., Stammel, J., Seceleanu, C., Tysiak, J., Mirandola, R., Ardagna, D.: Q-ImPrESS Project Deliverable D6.1: Method and abstract workflow. Tech. rep., Q-ImPrESS consortium (2009)
38. McConnell, S.: Code Complete: A practical handbook of software construction. Microsoft Press, Buffalo, NY (1993)
39. Miller, K.W., Morell, L.J., Noonan, R.E., Park, S.K., Nicol, D.M., Murrill, B.W., Voas, J.M.: Estimating the probability of failure when testing reveals no failures. IEEE Trans. Softw. Eng **18**(1), 33–43 (1992). DOI http://dx.doi.org/10.1109/32.120314
40. Mizan, A., Franks, G.: An automatic trace based performance evaluation model building for parallel distributed systems. In: Proc. 2nd Int. Conference on Performance Engineering (ICPE'11), pp. 61–72 (2011)
41. Musa, J.D.: Software Reliability Engineering: More Reliable Software Faster and Cheaper, 2nd edn. AuthorHouse (2004)
42. Musa, J.D., Iannino, A., Okumoto, K.: Software reliability: measurement, prediction, application. McGraw-Hill, Inc., New York, NY, USA (1987)
43. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proc. 28th Int. Conf. on Softw. Eng. (ICSE'06), pp. 452–461. ACM (2006). DOI http://doi.acm.org/10.1145/1134285.1134349
44. Object Management Group (OMG): UML Profile for MARTE, Beta 1. http://www.omg.org/cgi-bin/doc?ptc/2007-08-04 (2007). URL http://www.omg.org/cgi-bin/doc?ptc/2007-08-04
45. Pietrantuono, R., Russo, S., Trivedi, K.S.: Software Reliability and Testing Time Allocation: An Architecture-Based Approach. IEEE Trans. Softw. Eng. **36**(3), 323–337 (2010)
46. Q-ImPrESS Consortium: Project website. http://www.q-impress.eu
47. Rolia, J.A., Sevcik, K.C.: The method of layers. IEEE Trans. Softw. Eng. **21**(8), 689–700 (1995). DOI http://dx.doi.org/10.1109/32.403785
48. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009)
49. Sharma, V.S., Trivedi, K.S.: Quantifying software performance, reliability and security: An architecture-based approach. Journal of Systems and Software **80**(4), 493–509 (2007)
50. Singh, H., Cortellessa, V., Cukic, B., Gunel, E., Bharadwaj, V.: A bayesian approach to reliability prediction and assessment of component based systems. In: Proc. 12th International Symposium on Software Reliability Engineering (ISSRE'01), pp. 12–21 (2001)
51. Skuliber, I., Huljenic, D., Desic, S.: Black-box and gray-box components as elements for performance prediction in telecommunications system. In: Telecommunications, 2009. ConTEL 2009. 10th International Conference on, pp. 131 –134 (2009)
52. Smith, C.: Performance Engineering of Software Systems. Addision-Wesley (1990)
53. Smith, C.U.: Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software. Addison-Wesley (2002)
54. Snipes, W., Robinson, B., Brooks, P.: Approximating deployment metrics to predict field defects and plan corrective maintenance activities. In: 20th Int. Symp. on

Software Reliability Engineering (ISSRE'09), pp. 90–98. IEEE (2009)
55. Stammel, J., Reussner, R.: KAMP: Karlsruhe Architectural Maintainability Prediction. In: Proc. 1st Workshop GI-Arbeitskreis Langlebige Softwaresysteme (L2S2'09), pp. 87–98 (2009). URL http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-537/
56. Trivedi, K.: Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 2nd edn. Wiley & Sons (2001)
57. Wang, W.L., Pan, D., Chen, M.H.: Architecture-based software reliability modeling. Journal of Systems and Software **79**(1), 132–146 (2006)
58. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Norwell, MA, USA (2000)
59. Woodside, M., Franks, G., Petriu, D.C.: The future of software performance engineering. In: Future of Software Engineering (FOSE '07), pp. 171–187. IEEE Computer Society (2007). DOI http://dx.doi.org/10.1109/FOSE.2007.32
60. Xu, J., Oufimtsev, A., Woodside, M., Murphy, L.: Performance modeling and prediction of enterprise javabeans with layered queuing network templates. SIGSOFT Softw. Eng. Notes **31**(2), 5 (2006)
61. Zhu, L., Aurum, A., Gorton, I., Jeffery, R.: Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process. Software Quality Journal **13**(4), 357–375 (2005)