An Industrial Case Study on Quality Impact Prediction for Evolving Service-Oriented Software

Heiko Koziolek[1] Bastian Schlich[1] Carlos Bilich[1] Roland Weiss[1]
Steffen Becker[3] Klaus Krogmann[2] Mircea Trifu[2] Raffaela Mirandola[4] Anne Martens[5]

[1]Industrial Software Systems, ABB Corporate Research, Ladenburg, Germany
[2]Research Center for Information Technology (FZI), Karlsruhe, Germany
[3]University of Paderborn, Germany, [4]Politecnico di Milano, Italy
[5]Karlsruhe Institute of Technology (KIT), Germany

heiko.koziolek@de.abb.com

ABSTRACT

Systematic decision support for architectural design decisions is a major concern for software architects of evolving service-oriented systems. In practice, architects often analyse the expected performance and reliability of design alternatives based on prototypes or former experience. Model-driven prediction methods claim to uncover the tradeoffs between different alternatives quantitatively while being more cost-effective and less error-prone. However, they often suffer from weak tool support and focus on single quality attributes. Furthermore, there is limited evidence on their effectiveness based on documented industrial case studies. Thus, we have applied a novel, model-driven prediction method called Q-ImPrESS on a large-scale process control system consisting of several million lines of code from the automation domain to evaluate its evolution scenarios. This paper reports our experiences with the method and lessons learned. Benefits of Q-ImPrESS are the good architectural decision support and comprehensive tool framework, while one drawback is the time-consuming data collection.

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. When a system requires architectural changes, there are often multiple alternatives (e.g., make-or-buy, 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, 11, 13] to quantitatively evaluate evolution alternatives. These methods claim to be more cost-effective and less error-prone than current practice.

ICSE '11 Waikiki, Honolulu, Hawaii

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 [29]. 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 [13]. To reflect reliability characteristics, it must additionally allow modeling service and/or environmental failure probabilities [11]. Creating such models is currently tedious and error-prone because of sparse tool support and missing step-by-step guidelines [29].

While researchers have proposed many low-level modeling methods (e.g., queueing networks, stochastic Petri nets), these were often created in disconnected communities (e.g., performance [1, 13] or reliability [11]) thus treating a single quality attribute in isolation. High-level notations, such as UML MARTE [22] currently lack sufficient tool support [23]. Furthermore, there is only a limited number of documented case studies for model-driven prediction approaches (e.g., [12, 6, 10] for performance, [7, 24] for reliability) often analysing small-scale systems, focussing on particular steps instead of an end-to-end assessment, and providing limited hints on broader applicability.

The contribution of this paper is a large-scale, industrial case study on the applicability of a novel, model-driven prediction method called Q-ImPrESS (Quality Impact Predictions for Evolving Service-oriented Systems)[25]. 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. We applied the method on a large-scale, distributed process control system from ABB and report our experiences and lessons learned in this paper. The paper discusses the advantages of Q-ImPrESS as well as its inherent risks and pitfalls. Additionally, it provides initial evidence about the costs and benefits of this method to enable third party users assessing the usefulness in their own contexts.

The paper is structured as follows. Section 2 introduces the Q-ImPrESS method, models, tools, and prediction capabilities. Section 3 characterizes the system we analyze in our case study. Section 4 reports on experience from reverse engineering and manually building a Q-ImPrESS model. Section 5 then focusses on parameterizing the model for perfor-

mance and reliability properties, conducting predictions for different evolution scenarios, and performing tradeoffs analyses. Section 6 evaluates the Q-ImPrESS from a broader perspective and discusses costs and benefits. Section 7 concludes the paper.

2. THE Q-ImPrESS METHOD

The Q-ImPrESS method enables software architects to predict the impact of architectural design decisions on performance, reliability, and maintainability of a service-oriented software system. Fig. 1 shows a process view of the Q-ImPrESS method including major activities and artifact flow [18].

Service Architecture Model.

The Q-ImPrESS method is based on the novel 'Service Architecture Meta Model' (SAMM) [3] implemented using the Eclipse Modeling Framework (EMF). It enables a uniform handling of multiple quality attributes. The model (Fig. 1 middle part) is divided into multiple parts referencing each other, which in combination enable quality-of-service (QoS) predictions. The *repository* model describes components and interfaces. They are connected and allocated in the *service architecture* model. Component service behaviour is expressed in so-called *service effect specifications* (SEFFs), which abstractly model control flow and resource demands. The *hardware* and *target environment* models contain hardware devices and network interfaces. The *usage* model specifies the workload on the system.

Software architects have to exploit several information sources to construct a SAMM instance (Fig. 1 upper part). Architecture and user documentation serve as valuable source of information for modelling a system.

The *QoS annotation* model contains performance and reliability parameters as well as service transition probabilities. Performance parameters, such as execution times of individual services, have to be measured on a running instance of the system or coarsely estimated based on source code and experience. Reliability parameters, such as failure probabilities of individual services, have to be estimated (e.g., by statistical testing or reliability growth modeling based on bug tracking system data [15]).

The Q-ImPrESS method also includes a reverse engineering step supported by the tools SISSy[1], for parsing Java or C++ code, and SoMoX[2], for deriving component architectures from the parsed representation. These tools can assist the software architect in obtaining the static structure of the system. SoMoX uses heuristics, such as name resemblance, interface adherence, hierarchy mapping etc., to identify architectural structures in the code. Software architects can adapt these heuristics to their source code [4].

Q-ImPrESS Workbench and Evolution Scenarios.

Software architects semi-automatically create a SAMM instance using the Eclipse-based Q-ImPrESS workbench (Fig. 2). It enables running the reverse engineering tools, manually editing the models using graphical and textual editors, and executing the prediction tools. The workbench also manages evolution scenarios and alternatives. Evolution scenarios express planned changes to the system, such as new services, different allocation schemes, higher workloads, faster resources, etc. For such an evolution scenario (e.g., replacing a service), the software architect can model multiple alternatives (e.g., make or buy).

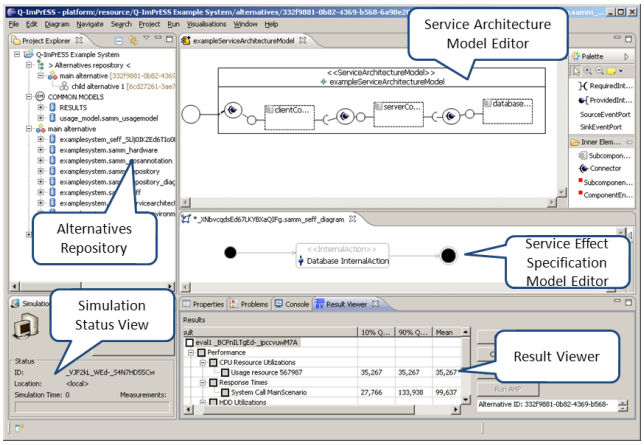


Figure 2: Q-ImPrESS workbench showing editors, simulation status, result viewer

Each change is modeled as a distinct architecture alternative. Note that such a SAMM instance in turn may serve as the original SAMM instance for a subsequent architecture change, thus, after modeling a series of architectural changes, the software architect ends up with a tree-like hierarchy of SAMM instances.

Q-ImPrESS also supports automatically generating architectural alternatives, which each posses optimised QoS attributes, with the tool PerOpteryx [17]. It requires the software architect to specify degrees of freedom in the architectural model (e.g., sizing of the hardware environment or allocation of components to different server nodes). Then it applies a metaheuristic search on the spanned design space, by manipulating the model with an evolutionary algorithm. As a results, the architect gets a set of Pareto-optimal alternatives.

Predictions and Tradeoff Analysis.

For each SAMM instance in the hierarchy of alternatives, model-to-model transformations create instances of existing QoS prediction models that are seamlessly processed by the respective model solvers (Fig. 1, lower part). For performance prediction, the Q-ImPrESS workbench creates instances of the Palladio Component Model (PCM) [5]. As performance metrics, the PCM supports response times, throughput, and resource utilizations. Internally, it can solve a model either using simulation (SimuCom) or using numerical analysis based on an additional transformation into layered queueing networks (LQN) [6].

For reliability prediction, the Q-ImPrESS workbench creates an instance of the KLAPER model [8], which in turn is transformed into a discrete time Markov chain and solved using the PRISM model checker. As a reliability metric, the solver supports calculating the probability of failure on demand for system-level services.

The results of the different solver tools are collected in the Q-ImPrESS workbench and feed the tradeoff analysis tool. The tool is a wizard application based on the Analytic Hierarchy Process (AHP) [26] and lets the user specify preferences for quality attributes and prediction metrics for in-

[1]http://fast.fzi.de/index.php/sissy
[2]http://www.somox.org

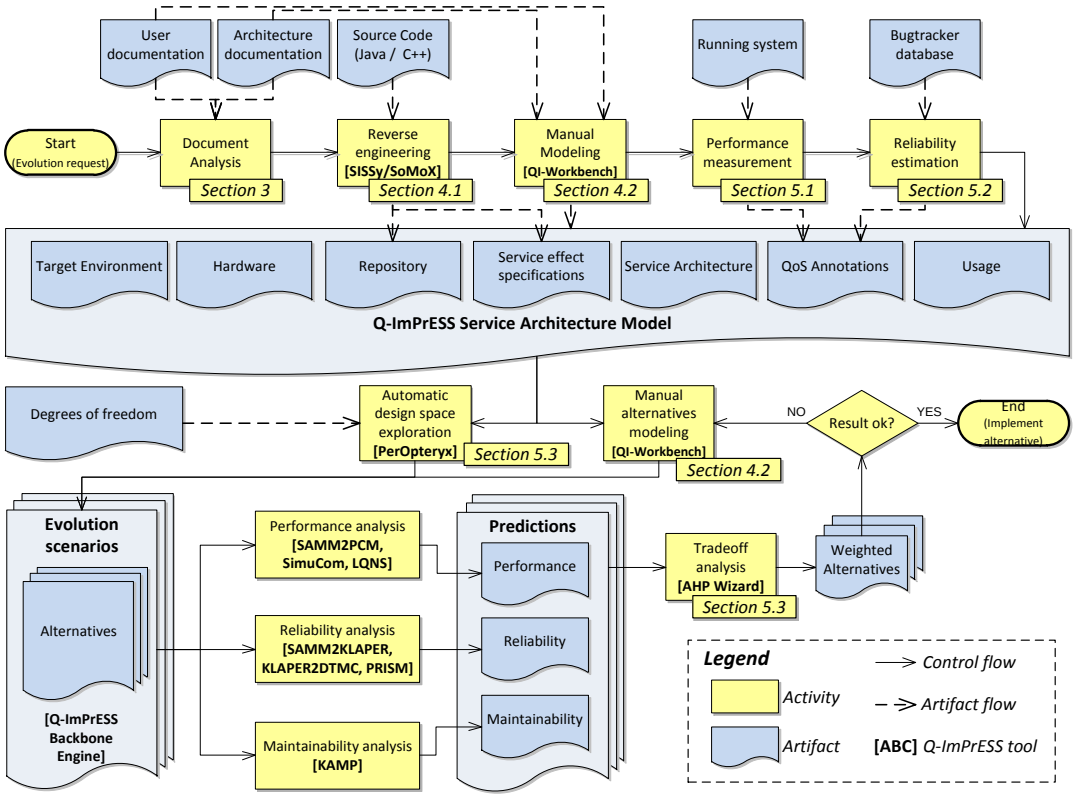


Figure 1: The Q-ImPrESS method: activities, artifacts, and tools

dividual alternatives based on pairwise comparisons. Based on these utility estimates the tool presents a ranking of the design alternatives. The software architect can then select the best suitable design alternative for implementation.

3. SYSTEM UNDER STUDY

To evaluate the various features of the Q-ImPrESS method, we selected a process control system (PCS) from the automation domain. 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 evaluation focuses on the server-side part of one ABB PCS and neglects embedded devices. The system under study consists of millions lines of C++ code and is structured into 8 subsystems with several hundreds 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 constantly evolves the system due to new customer requirements, technology updates, and bug reports. With the architectural models from Q-ImPrESS, we evaluated those evolution scenarios that have an impact on the architecture. These scenarios include replacing components with newer versions, adding new components, increasing the workload, or changing the hardware environment. Many of these scenarios include multiple alternatives (e. g., different implementation technologies) whose trade-offs with respect to quality-attributes need to be analyzed. The current practice of evaluating evolution scenarios is to build prototypes or rely on experience without having quantitative data to support design decisions.

4. MODEL CREATION

This section describes how we created the basic Q-ImPrESS model of the ABB PCS, which includes all Q-ImPrESS models except the *qos annotation* model. We describe both applying the reverse engineering tools (Section 4.1) and manual modeling (Section 4.2). Section 5 will provide more details on how the *qos annotation* model was created using performance measurements and reliability estimations.

4.1 Reverse Engineering

The Q-ImPrESS reverse engineering tools can support program comprehension and reduce the time for modelling by creating an initial component and connector model using static code analysis. As architecture knowledge is not fully encoded in the source code, a 100% reverse engineering of the original architecture cannot be expected. The tools apply heuristics to identify higher level components in source code.

For the PCS system, we executed a special use case of SoMoX, where we evaluated whether the static analysis can produce a model similar to a reference decomposition of the system provided in the architectural documentation. To limit the scope, we focused our analysis on the source code of a single subsystem. It consists of 250 KLOC in C++ and

uses standard Microsoft technologies.

SISSy needed 3.5 hours to parse the code and produced a 80 MB generalised abstract syntax tree (GAST) model. Unfortunately, the Eclipse CDT parser did not support Microsoft extensions to C++ and required manual preprocessing of the code. While all classes appeared in the GAST, component interfaces were not correctly identified. We therefore had to configure SoMoX to create one SAMM interface per recognized class and include all publicly defined functions as services of the interface.

SoMoX can be provided naming schemas (e.g., "EJB_" prefixes) to identify class names which could be misleadingly identifying higher level components. It additionally requires the user to adjust a number of weights for the included heuristics, such as code coupling, name resemblance, the level of composition, and the desired level of abstraction from the class-level decomposition. The user needs to change the weights and run SoMoX iteratively until a satisfying decomposition is found.

In our case, we ran SoMoX aprox. 30 times with different weights on the GAST produced by SISSy. Each run took between 4 and 12 minutes. While the initial class-level decomposition consisted of 348 primitive components and 80 composite components, we subsequently increased the abstraction to get 45 primitive components and 4 composite components (Fig. 3).

The resulting SAMM static structure model was complete and valid for model transformations, but did not resemble our reference decomposition. SoMoX identified two large clusters that could not be mapped to the 6 high-level components in the reference decomposition. Besides the nesting of components, the SoMoX output however also included connectors. We checked some of these connectors and could find respective calls in the code. Nevertheless, additional experience with the complex interplay of the SoMoX weights is needed to more easily arrive at a desired decomposition.

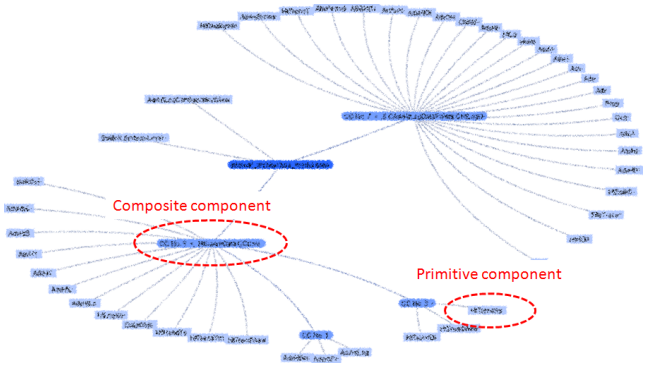


Figure 3: Reverse engineered SAMM repository including composite and primitive components for one subsystem of the ABB PCS (masked view)

Findings: Currently, missing documentation and experience make the user-configured component clustering in SoMoX with heuristic weights an intransparent trial-and-error process that needs further study. Although the Q-ImPrESS reverse engineering can be performed within one person week on a system with 250 KLOC, our achieved results were not yet convincing. As a reference decomposition was used to validate the SoMoX output, it is still questionable if executing SISSy/SoMoX is indeed faster than manually creating a Q-ImPrESS model based on this decomposition. Manual modeling might however miss architectural erosion and violation of the reference decomposition in the code and bears the risks of inconsistent modeling.

Risks of the method are associated with the limits of static analysis and the validation of the results. Static analysis for example cannot determine whether multiple instances of a component are active during runtime, which can be important for QoS predictions. If a system does not have a component-based structure the clustering by SoMoX fails. As another pitfall the results are hard to validate. If SoMoX is configured to a low abstraction level (meaning many components), it is tedious to validate whether the correct components were clustered. If the abstraction level is too high, assessing the validity of a few large components subsuming many classes is again hard.

4.2 Manual Modeling

As the Q-ImPrESS reverse engineering tools did not produce a *repository* model close to the reference decomposition of the PCS, we manually built such a model using the Q-ImPrESS model editors. Additionally, we manually created a *target environment*, *hardware*, *service effect specification*, *service architecture*, and *usage* model.

The main challenge for modeling is to find a suitable abstraction level of the system to enable meaningful predictions. We therefore searched for an abstraction level sufficient to analyze the architectural evolution scenarios defined for the ABB PCS. We decided that modeling on the level of components was too detailed, while modeling on the level of subsystems was too coarse-grained and would have excluded analyzing several 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. This allowed us to analyze scenarios where components are allocated to different servers. It also enabled us later to include the available reliability data into the composite components. For modeling the workload of the system in the *usage* model, we focused on four concurrently running, steady-state use cases and neglected transient use cases. We defined an user arrival rate for each scenario based on typical customer behavior. Fig. 4 depicts an abstracted overview of the architectural model including 28 components.

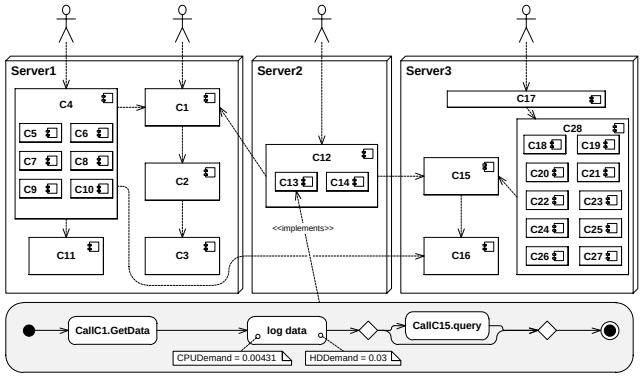


Figure 4: Architecture of the ABB process control system: components, connectors, deployment, and service effect specification

The typical workload of the system as well as the static dependencies between the components are described in the architectural documentation. However, it does not cover the number of calls between components during runtime, which are required for the *service effect specification* model. Therefore, we configured an ABB development tool for logging PCS system calls to log transitions between components. We set up the ABB PCS and executed it according to the use cases mentioned above. The logging tool produced a large list of component transitions, which we processed with a PCS-specific script to derive transition probabilities between components. Two PCS experts validated the transition probabilities by matching different paths through the model with the use cases we executed and referring to the architectural documentation.

Findings: Manually modeling our system using Q-ImPrESS required substantial effort (ca. 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.

A main *risk* of manual modeling is the expensive amount of time needed, which can be caused by imprecisely defined goals or missing input data. Thus modeling should rely on explicitly defined evolution scenarios and concrete non-functional requirements to avoid modeling unimportant details.

Finding a suitable abstraction level for the model is one of the hardest tasks. The Q-ImPrESS method drives the focus on providing a complete architectural picture. It is possible to model both on a too high abstraction level, where important bottlenecks can be missed, or to model on a too low abstraction level, where many modeling elements can be irrelevant for certain performance and reliability scenarios.

5. MODEL ANALYSES

This section describes the performance analysis (Section 5.1) and reliability analysis (Section 5.2) for the ABB PCS as well as applying the Q-ImPrESS tradeoffs analyses tools (Section 5.3).

5.1 Performance Analysis

For performance analysis, we parameterized the manually built Q-ImPrESS model of the ABB PCS with performance annotations obtained from measurements as described in the following. We also show how we used the model to make performance predictions.

QoS Annotations: To enable performance predictions, we had to annotate the approx. 35 service effect specifications of the ABB PCS model with resource demands to hardware devices. Because the resource demands of individual components cannot be directly measured from a running implementation with standard tools, we decided to exploit the service demand law [20] to determine the resource demands indirectly. According to this law, the demand of a component at resource k can be computed as $D_k = U_k / X_0$, where U_k is the utilization and X_0 is the system throughput.

We set up load drivers on a running instance of the ABB PCS and used them to measure the system throughput X_0. For measuring the utilization U_k, we employed the Windows performance monitor to record the CPU utilization per process. The hard disk utilization could be attributed to a single component. We neglected memory overheads and abstracted network overheads because they showed only a small overhead in our measurements.

Using the load drivers, we stressed the system for each use case in isolation and tried each time to reach the maximum possible throughput. Each measurement period lasted 180 seconds and was repeated eight times to avoid transient disturbances. Across different workload levels (e.g., number of of requested items) we applied a linear regression on the measurement data to determine individual resource demands. We calculated the mean value for all resource demands and additionally approximated some important resource demands using normal distributions incorporating standard deviations.

Model Validation: Table 1 shows some measured and predicted CPU utilizations for one use case. We used the SAMM2PCM transformation to make simulations using SimuCom and numeric analyses using the layered queueing network solver LQNS. In this use case, the average error between measured and predicted values was 10.7%. In other use cases the error went up to 40 percent. The model is used for extrapolation from the measurement data to determine the maximum throughput per use case. (Fig. 5). We could not measure the system for very high load levels, because the load drivers disturbed our measurements. Therefore, the predictions with a workload greater than 150 are not validated. However, the observed linear dependencies increase the confidence in model extrapolation. In conclusion, we deemed the model sufficiently accurate to conduct predictions for the evolution scenarios.

	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 1: Small excerpt of measured vs. predicted CPU utilizations for one use case of the ABB PCS

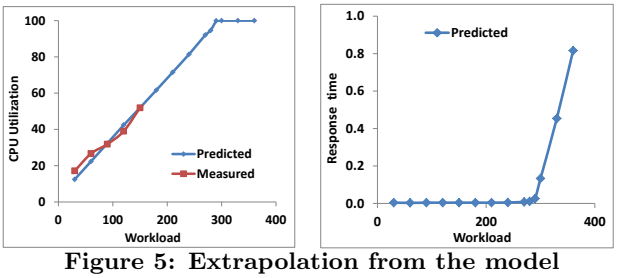
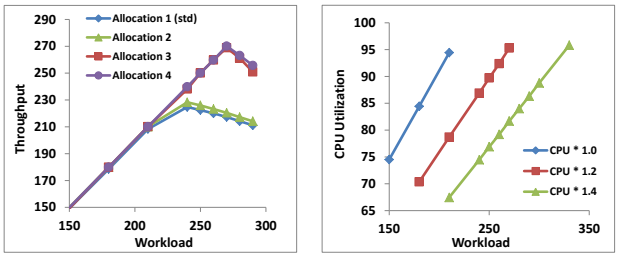


Figure 5: Extrapolation from the model

Analyses: Using the performance annotated model, we analysed various evolution scenarios for their performance impacts. Fig. 6 depicts two exemplary predictions. In Fig. 6(a), the maximum throughput for four possible allocation scenarios mapping components to different nodes was determined. While allocation 2 shows almost no improvement, allocation 3 and 4 enable an approx. 18 percent

higher maximum throughput. Fig. 6(b) shows the CPU utilization of one server in a specific use case. The different curves illustrate the impact of a 20 percent or 40 percent faster CPU. In the latter case, the CPU is not saturated up to a workload of 330.



(a) Throughput for allocation scenarios (b) CPU Utilization for resource evolution scenarios

Figure 6: Predictions for evolution scenarios

Findings: Q-ImPrESS offers two expressive and matured performance solvers (Simucom and LQNS), which allow evaluating many realistic settings. Q-ImPrESS does not introduce any new concepts in performance modelling. It would be desirable if specifics of service-oriented system would be better supported (e.g., dynamic adaptation or the comparison of transmission protocols, such as SOAP or REST). The accuracy of the performance predictions depends on the efforts spent for data collection (estimation for Q-ImPrESS in Section 6). Given detailed input data, the tools can provide accurate predictions.

In the future, prototype measurements could be integrated into the architectural model to analyse the influence of new components to the overall system performance. The models can then also help in scalability analysis.

Risks in applying Q-ImPrESS for performance analysis are mainly associated to data collection and the expressiveness of the SAMM. Q-ImPrESS offers limited support for performance measurement, but instead relies on the user to provide resource demands. If no performance test-beds or former measurements are available, Q-ImPrESS users have to create testbeds or rely on possibly inaccurate estimations. Unsupported constructs in SAMM (e.g., networks latencies, and memory effects) can rule out analyzing practically relevant evolution scenarios. Missing trust into the predictions might be another obstacle for successfully supporting design decisions.

5.2 Reliability Analysis

To conduct a reliability analysis using the Q-ImPrESS tools, we first annotated the general model described in Section 4 first with QoS annotations representing failure probabilities. After validating the model and obtaining the overall reliability, we conducted a sensitivity analysis.

QoS Annotations: There are different ways to obtain the failure probabilities of a system including statistical testing, reliability growth modeling, defect prediction based on code metrics, and fault injection [11]. We decided to use failure reports from a bug tracking database to calculate the failure probabilities of the subsystems of the PCS [15].

In the bug tracking database, each bug report has a timestamp, a criticality, a reference to the involved subsystem, and a list of actions taken. We only selected bug reports that were reported after the system release date, had a certain criticality (i. e., critical and high), and had been fixed.

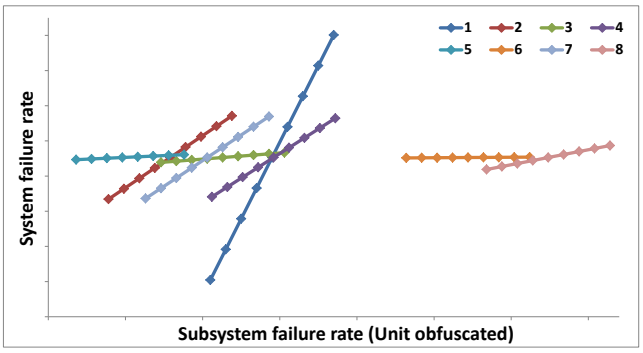


Figure 7: Sensibility analysis: system failure rate varies differently for individually changed subsystem failure probabilities (units obfuscated)

To calculate the failure probabilities of the subsystems, we searched the IEEE Std. 1633-2008 for a fitting software reliability growth model. We chose the Littlewood and Verrall model [16] (LV), which was used in the same domain before and fits to the two assumptions that we made, namely, that the reliability is growing, but that fixing a bug can introduce new bugs.

Using the LV model and the filtered failure reports from the bug tracking database, we calculated the failure probability for each subsystem of the ABB PCS. Then, 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.

Model Validation: We validated the annotated model used for reliability analysis in two ways. First, we compared the failure probabilities of the subsystems to different code metrics. Then, we compared the outcome of the reliability analysis done within Q-ImPrESS with the results of a reliability analysis of a Markov model that we created outside the Q-ImPrESS tools using a script.

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 [21, 27]. We compared the subsystem failure probabilities and the arithmetic average cyclomatic complexity per method [19], 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 within Q-ImPrESS with a reliability analysis of a Markov model done outside of the Q-ImPrESS tools. The deviation between the two results was 7%. Given that the grade of abstraction is slightly different in both models, this deviation is not too high. Therefore, we are confident that the Q-ImPrESS model is valid.

Analyses: 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. The result of the sensitivity analysis is shown in Fig. 7.

Subsystems 8 and 6 have the highest failure probabilities

(on the far right side), while subsystem 5 has the lowest failure probability (on the far left side). The slopes of the curves are a measure for the sensitivity of the system failure probability to the subsystem failure probability.

Subsystem 1 is the most sensitive for the system reliability, which appears plausible because it is responsible for processing most of the data in the PCS and is called most often. Subsystem 6, which is infrequently used by many 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 8, we had estimated the highest failure probability, but it is in fact also only a minor driver for system reliability.

To validate the results of the sensibility 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.

Findings: The reliability analysis in Q-ImPrESS is simple as there is currently no support for concurrency, hardware reliability, replication patterns and fault tolerance mechanisms. Thus, there are only limited options to model reliability evolution scenarios. Changing the hardware, reallocating components, or increasing the workload do not change reliability prediction results in Q-ImPrESS although in practice these changes can have a significant impact. A more refined model would be needed to evaluate the impact of different component topologies on the system reliability.

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 experience. 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 [24]. This can also save future maintenance costs.

Data collection for the reliability annotations is the hardest part, as it is not obvious which data to collect and how to collect this data. The validation of the data and the evaluation of reliability prediction is also difficult. The collection of data needed for evaluating the reliability predictions would take years because the changes would first have to be implemented and due to the low failure probability of the system, the system would have to be monitored for a few years.

5.3 Tradeoff Analysis

The following describes applications of the two alternatives for 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 trading off a handful of candidates.

Design Space Exploration with PerOpteryx.

We applied the Q-ImPrESS PerOpteryx tool [17] 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 QoS prediction tools (e.g., 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. 8).

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 to allocate 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 rules of thumb.

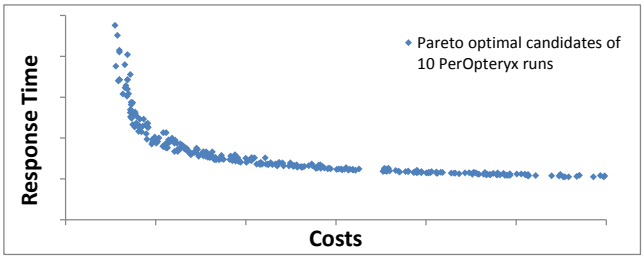


Figure 8: Tradeoff analysis results from PerOpteryx: response time vs. costs for the ABB PCS visualized as a set of unified Pareto front (units obfuscated)

Findings: The application of the PerOpteryx tool is an interesting alternative to the current practice of sizing and configuring systems by rules 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.

The explorable degrees of freedom in the design are limited to architecture-level changes. As another pitfall, 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 modelled and the required validation of the resulting candidates, while the effort for running the tool is negligible.

Weighting Alternatives with AHP Wizard.

As a second alternative for tradeoff analysis, we applied the Q-ImPrESS AHP Wizard [9]. AHP was chosen in Q-ImPrESS because of promising results in former software architecting studies [30]. 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 analysed 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 demand, 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. We neglected specifying costs for each alternative.

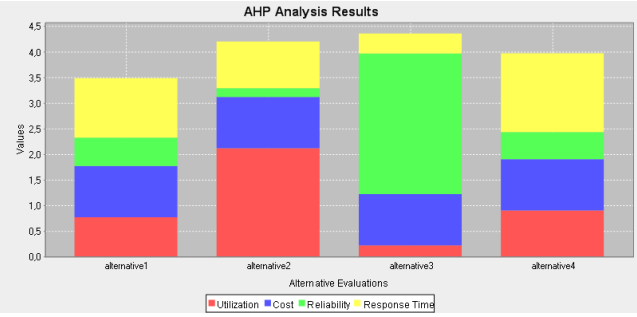


Figure 9: Tradeoff analysis results from AHP Wizard: alternative 3 receives the highest score because of the preference for reliability

Using the Q-ImPrESS AHP wizard, we specified a high preference for reliability and provided 24 weights for the prediction result comparison. Fig. 9 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, 4 quality attributes). For 10 alternatives, 180 comparisons would be required thus becoming impractical.

6. Q-ImPrESS EVALUATION

This section discusses the applicability of Q-ImPrESS, evaluates cost and benefits, and reports on lessons learned.

6.1 Applicability of Q-ImPrESS

Below we discuss some findings about the applicability of the Q-ImPrESS method in an industrial environment:

- **Integrated approach with comprehensive tool support:** The method uniquely integrates multiple mature research approaches in a single environment; even though the tools still need more stabilization. The method is technology-agnostic and has been applied in different domains (e.g., automation, telecommunication [14], or business information systems [17]).
- **Limited integration with other tools and methods:** While the Q-ImPrESS tools are well-integrated with each other, they do not easily incorporate into existing non-Java development environments. Existing UML models need to be recreated for Q-ImPrESS as there are currently no model transformations to SAMM. The tools are aligned with Eclipse but do not seamlessly integrate into other development environments.
- **Cost-effective only for large systems:** The complexity of the SAMM (>100 classes) requires a substantial learning effort (approx. 1 week). Q-ImPrESS models are more difficult to understand than simple queueing networks because they are developed for evaluating evolution scenarios. Therefore, it might not be justified to apply Q-ImPrESS for single evolution scenarios or smaller systems (efforts discussed below). However, the effort might pay off when the models can be reused.
- **Limited expressiveness of SAMM:** Although being complex, SAMM still lacks constructs to model many interesting evolution scenarios. There is no direct support for virtualization, OS changes, application server configurations, transmission protocols, event-based communication, real-time scheduling, or dynamic architectures. There is also only limited support for modeling the middleware.
- **Missing data collection support:** The hardest, most time-consuming, and error-prone activity in modelling is data collection (i.e., determining resource demands, failure probabilities, etc.). For these activities, Q-ImPrESS offers no method or tool support and requires manual work or third party product use from the user. It would be conceivable to create technology-specific versions of Q-ImPrESS, for example to support data collection in .NET or Java environments.

The following findings are not specific to Q-ImPrESS, but also apply to other model-driven prediction methods:

- **Complexity of evolution scenarios determines accuracy:** The accuracy of model-driven predictions mainly depends on the accuracy of the input data. If a user wants to analyse large-scale redesigns, input data must almost certainly be based on manual estimations (i.e., guessing), as new parts of the system cannot be measured. Then predictions might be inaccurate and lead to suboptimal design decisions. Therefore model-driven predictions are best applicable to analyse smaller evolutionary changes or to relatively rank design alternatives.

- **Risks and pitfalls:** Missing goal-orientation of the Q-ImPrESS user can lead to unfocused modeling and wasted time. Inaccurate input data can lead to inaccurate predictions. Collecting precisely the needed data is difficult, and finding a suitable abstraction level requires multiple iterations. Decision makers have to trust the prediction results so that they become meaningful.

6.2 Cost/Benefit

Former studies have pointed out that model-driven prediction methods bear the potential for a significant return on investment (e.g,. >400% for a 15 person, 18 months project [28]). Improving architectural design decisions can have a profound impact on a system [2]. However, quantifying the *benefits* of model-driven predictions requires long-term studies and is therefore out of the scope for this paper. Instead, the following comparision of *cost* estimations for Q-ImPrESS with conventional methods (e.g., prototyping or relying on experience) helps the reader to further evaluate the applicability of Q-ImPrESS.

Tab. 2 shows estimated efforts for the different Q-ImPrESS activities based on our experience when applying the method. The best case reflects the situation when information for modeling is readily available and the evolution scenarios are rather simple. The worst case reflects the situation when test beds have to be set-up, new methods for data collection have to be applied, or complex scenarios need to be analysed.

Note that the external validity of these estimations is debatable. In general, the efforts for Q-ImPrESS depend on the desired accuracy of the prediction and the complexity of the analysed evolution scenarios and can therefore vary heavily. Furthermore, the efforts depend on the familiarity of the users with the system under study and the existence of data collection facilities (e.g., existing performance testbeds, customer failure data).

#	Activity Name	Best	Likely	Worst
1	Document Analysis	40	80	120
2	Reverse Engineering	20	60	200
3	Manual Modeling	50	150	300
4	Performance Measurement	20	70	190
5	Reliability Estimation	16	48	140
6	Autom. Design Exploration	8	16	40
7	Modeling Evol. Scenarios	10	16	24
8	Performance Analysis	3	4	8
9	Reliability Analysis	3	8	12
10	Tradeoff Analysis (AHP)	1	1	1

Table 2: Effort estimations (person hours) for the Q-ImPrESS activities

Typical prototyping studies at ABB last for 3-12 person months, which is longer than our modeling activities. The efforts for modeling and prototyping are however not mutually exclusive. To assess new technologies or components, some prototype measurements would also be required when modeling in order to parameterize the Q-ImPrESS models. The table currently indicates higher upfront efforts for modeling the initial system, whereas the effort for the analysis of evolution scenarios is moderate. This might indicate that the models pay off best if used multiple times for assessing evolution scenarios over the whole life-cycle of a system.

6.3 Lessons Learned

In the following we report on our lessons learned during the project:

- **Data collection is most time-consuming:** In the beginning of our case study, we vastly underestimated the efforts required for determing QoS annotations (i.e., transition probabilities, resource demands, and failure probabilities). While there are standard tools for performance measurements, it is still time-consuming to set up a distributed, service-oriented system, derive performance requirements, create load drivers, monitor performance, statistically analyze the data, and instantiate the models. For reliability estimation, data collection is even harder as there are no established step-by-step processes but rather many approaches (e.g., reliability growth modeling, statistical testing, code metrics) with different pitfalls (e.g., missing statistical validity, limited scalability [15]).
- **Model creation is an iterative process:** In our experience, finding a suitable abstraction level for modeling is difficult and requires several iterations. This is especially pronounced if there is limited experience with the system under study. Although model creation is currently depicted as a single step in the Q-ImPrESS process model, it is actually an iterative process. Multiple information sources (e.g., architectural documents, developer interviews, measurements, estimations) have to be combined to create a meaningful model.
- **Static code analysis clashes with QoS predictions:** We found that the results of the static code analysis currently are not meaningful for subsequent steps in Q-ImPrESS. This issue is probably more evident in our case study because the size of the system under analysis is much larger than previously analysed Java systems. A QoS model for such a complex system must necessarily heavily abstract from the source code. The structuring of the source code may only bear a limited resemblance of the system at runtime, therefore the resulting decomposition is of limited value. Future work could extend Q-ImPrESS with dynamic analysis methods, e.g., performance monitoring or bug tracking system analyses tools.
- **How to design an integrated meta-model:** Finding a fitting abstraction level for a QoS modeling language is hard. The Q-ImPrESS SAMM is complex but it still does not support many practically-relevant evolution scenarios. Some elements in SAMM are currently not used by the prediction tools (e.g., processor caches). Future meta-modelling should carefully consider the eventual analysis of any meta model element. To reduce the complexity of the model and better support certain evolution scenarios, specialised model support for extended evolution modelling would be beneficial.

7. CONCLUSIONS

We conducted a large-scale industrial case study of the novel Q-ImPrESS method for service-oriented systems. The method combines several formerly disconnected approaches for performance and reliability prediction and provides decision support for software architects. In our case study, the performance prediction part of Q-ImPrESS proved to be most mature, while the reliability prediction part still required substantial manual effort. The Q-ImPrESS reverse engineering tools could not yet be applied successfully.

The benefits of Q-ImPrESS over similar approaches lie in its integrated tool environment, its treatment of multiple quality attributes, its built-in support for evolution scenario

modeling, and its tools for tradeoff analysis. Drawbacks include the tedious, manual data collection, the still unclear cost-efficiency, and the support of only a subset of practical evolution scenarios. Q-ImPrESS should be applied if input data for the models is easily obtainable and multiple evolution scenarios need to be analysed. Our case study provides interested researchers and practitioners evidence for the applicability of model-driven quality prediction methods.

Future methods could work with restricted domain-specific languages targeting specific evolution scenarios. Q-ImPrESS itself could be extended to support more quality attributes (e.g., security, safety), so that even more complex tradeoff analyses could be conducted. To reduce the effort for applying Q-ImPrESS, there could be some automated support for data collection. To better integrate into existing development environments, transformations from UML models to Q-ImPrESS models could be implemented.

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.

8. REFERENCES

[1] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Trans. Softw. Eng.*, 30(5):295–310, May 2004.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley Professional, 2nd edition, 2003.

[3] S. Becker, L. Bulej, T. Bures, P. Hnetynka, L. Kapova, J. Kofron, H. Koziolek, J. Kraft, R. Mirandola, J. Stammel, G. Tamburelli, and M. Trifu. Q-ImPrESS Project Deliverable D2.1: Service Architecture Meta Model (SAMM). Technical Report 1.0, Q-ImPrESS consortium, September 2008. `http://bit.ly/bfSGrh`.

[4] S. Becker, M. Hauck, M. Trifu, K. Krogmann, and J. Kofron. Reverse Engineering Component Models for Quality Predictions. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering, European Projects Track*, pages 199–202. IEEE, 2010.

[5] S. Becker, H. Koziolek, and R. Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[6] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced Modeling and Solution of Layered Queueing Networks. *IEEE Trans. Softw. Eng.*, 35(2):148–161, 2009.

[7] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *Proc. 16th IEEE Int. Symp. on Software Reliability Engineering (ISSRE'05)*, pages 43–52. IEEE Computer Society, 2005.

[8] V. Grassi, R. Mirandola, and A. Sabetta. 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.

[9] L. Hatvani, A. Jansen, C. Seceleanu, and P. Pettersson. 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, October 2010.

[10] N. Huber, S. Becker, C. Rathfelder, J. Schweflinghaus, and R. H. Reussner. Performance modeling in industry: a case study on storage virtualization. In *Proc. 32nd Int. Conf. on Software Engineering (ICSE'10)*, pages 1–10. ACM, 2010.

[11] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Springer Software and System Modeling*, 7(1):49–65, 2008.

[12] S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Trans. Softw. Eng.*, 32(7):486–502, 2006.

[13] H. Koziolek. Performance Evaluation of Component-based Software Systems: A Survey. *Performance Evaluation*, 67(8):634–658, 2010.

[14] H. Koziolek, J. Doppelhamer, R. Weiss, C. Bilich, B. Schlich, I. Skuliber, M. Zemljic, S. Desic, and D. Huljenic. Q-ImPrESS Project Deliverable D7.3: Demonstrator Results Documentation. Technical report, Q-ImPrESS consortium, October 2010.

[15] H. Koziolek, B. Schlich, and C. Bilich. A Large-Scale Industrial Case Study on Architecture-based Software Reliability Analysis. In *Proc. ISSRE'10*. IEEE Computer Society, November 2010. To appear.

[16] B. Littlewood and J. L. Verrall. A Bayesian Reliability Growth Model for Computer Software. *Applied Statistics*, 22(3):332, 1973.

[17] A. Martens, H. Koziolek, S. Becker, and R. H. Reussner. Automatically improve software models for performance, reliability and cost using genetic algorithms. In *Proc. 1st Int. Conf. on Performance Engineering (ICPE'10)*, pages 105–116. ACM, February 2010.

[18] M. Masetti, S. Becker, I. Skuliber, M. Hauck, J. Kofron, K. Krogmann, J. Stammel, C. Seceleanu, J. Tysiak, R. Mirandola, and D. Ardagna. Q-ImPrESS Project Deliverable D6.1: Method and abstract workflow. Technical report, Q-ImPrESS consortium, 2009.

[19] S. McConnell. *Code Complete: A practical handbook of software construction*. Microsoft Press, Buffalo, NY, 1993.

[20] D. A. Menasce, L. W. Dowdy, and V. A. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall, 2004.

[21] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proc. 28th Int. Conf. on Softw. Eng. (ICSE'06)*, pages 452–461. ACM, 2006.

[22] OMG. UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, November 2009. `http://www.omg.org/spec/MARTE/`.

[23] OMG. Tools related to MARTE, October 2010. `http://www.omgwiki.org/marte/node/31`.

[24] R. Pietrantuono, S. Russo, and K. S. Trivedi. Software Reliability and Testing Time Allocation: An Architecture-Based Approach. *IEEE Trans. Softw. Eng.*, 36(3):323–337, 2010.

[25] Q-ImPrESS Consortium. Project website. `http://www.q-impress.eu`.

[26] T. L. Saaty. *Multicriteria Decision Making - The Analytic Hierarchy Process*. RWS Publishing, 2nd edition, 1990.

[27] W. Snipes, B. Robinson, and P. Brooks. Approximating deployment metrics to predict field defects and plan corrective maintenance activities. In *20th Int. Symp. on Software Reliability Engineering (ISSRE'09)*, pages 90–98. IEEE, 2009.

[28] L. G. Williams and C. U. Smith. Making the business case for software performance engineering. In *Proc. Int. CMG Conference*, pages 349–358. Computer Measurement Group, 2003.

[29] M. Woodside, G. Franks, and D. C. Petriu. The future of software performance engineering. In *Future of Software Engineering (FOSE '07)*, pages 171–187. IEEE Computer Society, 2007.

[30] L. Zhu, A. Aurum, I. Gorton, and R. Jeffery. Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process. *Software Quality Journal*, 13(4):357–375, 2005.