Softw Syst Model (2008) 7:49–65
DOI 10.1007/s10270-006-0040-x

REGULAR PAPER

Survey of reliability and availability prediction methods from the viewpoint of software architecture

Anne Immonen · Eila Niemelä

Received: 24 January 2006 /Revised: 18 August 2006 / Accepted: 2 November 2006 / Published online: 12 January 2007

Abstract Many future software systems will be distributed across a network, extensively providing different kinds of services for their users. These systems must be highly reliable and provide services when required. Reliability and availability must be engineered into software from the onset of its development, and potential problems must be detected in the early stages, when it is easier and less expensive to implement modifications. The software architecture design phase is the first stage of software development in which it is possible to evaluate how well the quality requirements are being met. For this reason, a method is needed for analyzing software architecture with respect to reliability and availability. In this paper, we define a framework for comparing reliability and availability analysis methods from the viewpoint of software architecture. Our contribution is the comparison of the existing analysis methods and techniques that can be used for reliability and availability prediction at the architectural level. The objective is to discover which methods are suitable for the reliability and availability prediction of today's complex systems, what are the shortcomings of the methods, and which research activities need to be conducted in order to overcome these identified shortcomings. The comparison reveals that none of the existing methods entirely fulfill the requirements that are defined in the framework. The comparison framework also defines the characteristics required of new reliability and availability analysis methods. Additionally, the framework is a valuable tool for selecting the best suitable method for architecture analysis. Furthermore, the framework can be extended and used for other evaluation methods as well.

Keywords Reliability and availability analysis · Software architecture · Software components

Communicated by Oystein Haugen.

A. Immonen (✉) · E. Niemelä
Software Architectures and Platforms, VTT Technical Research Centre of Finland, P.O. Box 1100, 90571 Oulu, Finland
e-mail: Anne.Immonen@vtt.fi

E. Niemelä
e-mail: Eila.Niemela@vtt.fi

1 Introduction

Software systems are increasingly entering consumers' everyday life. These systems are often highly complicated, and distributed to different platforms over wired or wireless networks. Many of the modern distributed systems are software-intensive, often embodying *service architectures* and providing thereby a variety of services for their users [1]. To satisfy the consumers' needs, these systems must demonstrate high reliability and availability; thus, they must function correctly and without interruption. For example, high reliability is needed for ensuring information exchange over network connections in urgent situations, and high service availability is a prerequisite for meeting customer satisfaction, selling the service, and realizing profit potential.

Reliability is defined here as the probability of the failure-free operation of a software system for a specified period of time in a specified environment [2]. Service reliability extends the traditional reliability definition, requiring in turn that either the system does not fail at all for a given period or it successfully recovers state information after a failure for a system to resume its service as if it was not interrupted. Availability is measured as the

probability of a software service or system being available when needed. Reliability and availability are often defined as attributes of dependability, which is "the ability to deliver service that can justifiably be trusted" [3]. From an architecture point of view, reliability and availability are execution qualities of a software system [4].

Several measures are traditionally used for reliability and availability, such as mean time to failure (MTTF), mean time to repair (MTTR) and failure rate. The traditional views and measures, however, might not scale up to the needs placed on today's complex systems by their multiple stakeholders. Different stakeholders, such as end users, maintainers, and developers might have different requirements for the value of reliability and availability indicators, as well as differing definitions and comprehension of these attributes. Therefore, reliability and availability must be approached from a more global perspective, starting with the collection and documentation of the stakeholders' requirements. Some of these requirements may well conflict with each other, and therefore, a compromise must be sought [5]. As a result, reliability and availability must be accounted for the context of multiple-quality requirements and trade-off analysis [6].

Software architecture is the first asset that describes the system as a whole. An architecture defines the system structure comprising the components, their externally visible properties and their relationships among each other [7]. A service-oriented architecture is a collection of services that communicate with each other with simple data passing or coordinate some activity with two or more services. Architecture modeling and analysis are closely related because the analysis of the architecture is only possible if the architecture is represented in a way that enables the analysis [8]. By analyzing the reliability and availability prior to system implementation, time and resources are significantly saved. This "predictive" analysis enables the reliability and availability problems to be solved more easily, at the architecture level, when modifications are easier and cheaper to implement. In addition, the effects of the design decisions on system reliability and availability can be detected beforehand. Therefore, the analysis approach must enable (a) the derivation of reliability and availability (R&A) requirements to architectural decisions in order to evaluate how the specified R&A requirements are addressed in the architectural models and (b) the tracking of R&A analysis results from the architectural models to the requirements in order to validate whether or not the R&A requirements are met in the architecture. The results of the analysis should enable the evaluation of the architecture against the requirements or evaluation criteria.

Reliability and availability prediction from the architectural descriptions is a challenging task for two main reasons:

- Reliability is strongly dependent on how the system will be used. Since reliability and availability are execution qualities, the impact of faults on reliability differs depending on how the system is used, i.e. how often the faulty part of the system is executed. The analysis of different ways and frequencies to execute the system is a challenge to R&A prediction, especially when the usage profiles of the system are unknown beforehand.
- The reliability of software architecture depends on the reliability of individual components, component interactions, and the execution environment. The reliability of a component depends on its internal capabilities, e.g. implementation technology, size, and complexity, information about which might be unavailable, or not yet exist, while architecting. Furthermore, components rely on other components, interactions between components, and on an execution environment, the reliability of which may be unknown.

Several analysis/prediction methods have been developed during recent decades for different types of purposes and by different communities. Consequently, they have different definitions and measures for reliability, architecture, inputs, outputs, notations, assumptions, users, etc. Our contribution is a state-of-the-art comparison of the architecture-based reliability and availability analysis methods and techniques. The purpose is to find a method or a set of methods that can be applied to today's complex software systems, including service-oriented systems, at the architecture level, as well as to discover the shortcomings of recent methods in terms of identifying development needs. Section 2 introduces our comparison framework for analysis methods. We use this framework to comparing the selected characteristics of the reliability and availability analysis methods collected from the literature. In Sect. 3, we present an overview of reliability and availability prediction. Section 4 represents the comparison of methods and techniques, and Sect. 5 presents the results of our survey. Finally, our conclusions are summarized.

2 A comparison framework for reliabiliy and availability prediction methods

The framework introduced in Table 1 is used as a basis for our method comparison. The framework reveals

Table 1 A comparison framework for reliability and availability analysis methods

Category	Elements	Questions
Context	Goal	What is the goal of the analysis method?
	Scope of applicability	Is the method/technique applicable to the different layers of software: application, middleware, and infrastructure?
	Application domain independency	Is the method limited to any application domain?
	Component-specificity	Can the method be used to predict the reliability and availability of the individual components? How does the method/technique treat black-box components?
	Architecture-specificity	Can the method/technique be applied to software systems that are distributed to several (hardware) platforms? Are the different interactions between components considered?
	Platform/implementation technology independency	Can the method/technique be used before making any implementation-related decisions? Is the method dependent on a certain platform or implementation technology?
User	Target group	Who is the intended user of the method?
	Needed skills	What skills are required for using the method?
	Expected benefits	What are the benefits of using the method?
	Required resources	How much extra work does the method require? How much time does the use of the method require?
Contents	Language	What notation is used in architecture descriptions?
	Architectural viewpoints	What views does the method use for predicting reliability and availability? How is the behavior modeled?
	Analysis model	Does the method provide a special model with which the analysis is performed?
	System usage	Have the different ways and frequencies that were used for executing tasks been taken into account?
	Variability	Is the variation of architecture considered in the analysis? Can the analysis be performed for different product variants by reusing existing knowledge?
	Tool support	Are there any tools that support the method?
	Analysis process	How is the analysis performed (the inputs, outputs and techniques)?
	Limitations	What are the assumptions and limitations of the method?
Validation	Maturity of the method	When was the material of the method first published? Has the method been applied in the lab only or also in the development of large-scale products? How many times and to what size of applications has the method been applied? Is there evidence for the method's benefits and costs?
	Traceability of R&A requirements	Does the method define how to trace requirements into the architecture and to the results of the analysis?
	Precision of prediction	How close are the predicted values to the actual values when the method has been used?

the characteristics required for the analysis methods to be implemented for the evaluation of their suitability for architecture level prediction. The categories of the framework are based on the NIMSAD (Normative Information Model-based Systems Analysis and Design) framework [9]. NIMSAD classifies the method elements into four categories: context, user, method content, and evaluation. In the context category, the method is examined from the angle of the problem situation, whereas in the user category, the method is examined from the viewpoint of the intended method users. In the third category, the focus of the examination is the content of the method itself. The last category, which is evaluation, is turned into the method validation in this study because few or none of the methods focus on the evaluation of the method context, user, and content. Method validation addresses the maturity of the method and the results of the method. The elements of the categories give rise to some specific requirements for the reliability and availability analysis methods that are applicable at the architecture level.

First, the goal of the method must be clearly defined. Different methods have different approaches to R&A analysis, which can affect their suitability for the purposes of the method users. The goal can be, for example, a numerical indicator about the R&A of a system, or qualitative validation in terms of how well the system requirements are met.

An analysis method is the most applicable when it can be applied to different parts of a system architecture. Many software systems that are developed today are based on a three-layer approach. The layer approach

enables the decomposition of software "into subgroups of subtasks in which each group of subtasks is at a particular level of abstraction" [10]. The layers comprise application, middleware, and infrastructure. The application layer is the closest to the end users of the system, whereas the infrastructure layer is closest to the hardware and is the most dependent on the technology that is used in the domain. The middleware layer is in-between, which in turn provides services to the application layer, and is based on the services provided to it by the infrastructure layer. Middleware is designed to manage the complexity and heterogeneity that is inherent in distributed systems, acting as if it were "glue" for connecting distributed applications. Service architecture usually refers to the architecture of the application and middleware layers.

The application domain embodies systems that share a set of common capabilities and data in a specified area. Analysis methods may be targeted at certain application domains, which may limit their applicability to other domains.

Software development has experienced a change in its paradigm; today an increasing number of software programs are being developed by way of integrating existing components. A component is a self-contained, replaceable part of a software system that fulfils a clear function or a group of related functions in the context of software architecture. The components used can be developed in-house or be third party components, such as commercial off-the-shelf (COTS), modified off-the-shelf (MOTS) and open source (OS) components. The use of third party components has a number of potential benefits for component integrators, such as lower costs, better-tested products, and maintenance support. However, the possible black-box nature of these components and insufficient documentation brings several risks along with their use. Without knowing the component's internal structure and dependencies, it is difficult to verify the component's R&A. The specific documentation of a component becomes a key issue in the case of third party components. For example, information about the component testing, such as test environments, test cases, test results, and the coverage of the testing, may give some confidence in the R&A of a component.

The reliability prediction of a component is problematic, as it is affected by several factors, such as the component's implementation technology and configuration. The different usage profiles have a great concern within the frequency of executing each component and each interaction between the components, and therefore, these profiles also affect component's reliability. The reliability depends also on the reliability of an execution environment of a component, as well as the reliability of external services that are called by the component's service. In the case of reusable components there may be a large number of these execution environments. Therefore, the reliability of a component to execute its responsibilities without failures should be predicted independently from the execution environment as well as in the execution environment. Since there are a lot of dependent factors that affect the reliability of components, the analysis method must recognize all of them when analyzing component level reliability.

Component-based software architecture can embody an architectural style or styles [11], [12], [13], which are candidate architectural solutions that support reliability and availability requirements differently. Software systems can be distributed to different hardware platforms, which make reliability and availability gain new dimensions while R&A is being analyzed over a wired or wireless network. Software architecture also reveals the interaction behavior among components, i.e. architectural relations. The set of proposed stereotypes for architectural relations that describe the interaction types between components are control, data, and use [1].

In the ideal situation, architectural level R&A prediction can be performed purely on the conceptual abstraction level prior to making any implementation-related decisions concerning, for example, the selection of platform or implementation technology. Model-driven architecture (MDA) is an approach that enables a separation of the architecture of a software system from its implementation platform and a decoupling of the system architecture from its execution environment. This brings forth several benefits to the development of software systems and the evaluation of the quality properties at the architecture level. The use of MDA is increasingly demonstrating better designs that have a longer life and an architecture that is independent of language, vendor, and platform. MDA is considered a promising approach in service-oriented architecture development and software product family architecture development [14].

Because R&A prediction should be performed prior to software implementation, quality analysis has become the task of software architects. Architects need a prediction method, or a mixture of methods and techniques, to analyze software architecture with respect to reliability and availability. The method has to provide architects with many benefits, including a medium for validating the architecture against the evaluation criteria, predicting the reliability and availability of an architecture, and comparing architectural solutions. The method must be

easy to use, should not require much additional work, and its application should only take a very short time, i.e. some hours or days. If this is not the case, the method is deemed inappropriate for industrial settings.

Due to large-scale requirements, and rather often three-layer characterization and a highly complicated distribution to several platforms, the architecture modeling of software systems is challenging. For architecture analysis, as well as reliability and availability prediction, the architecture models must be represented in a way that enables a comprehensive view of the different aspects and concern levels of the software system. The use of standard and widely accepted modeling languages, such as UML (Unified Modeling Language) [15], improves the understandability of architectural models among different stakeholders. UML 2.0 [16] offers some enhancements in architecture description, e.g. the diagrams required for the structural descriptions of systems. In addition, UML 2.0 enables the description of many elements that are found in today's software technology, such as Model Driven Architecture (MDA) and Service-Oriented Architecture (SOA) [17]. To represent quality aspects, such as reliability and availability in architectural models, modeling languages must be extensible. The UML standard can be extended by specific profiles to support certain quality aspects. A profile is, according to [15], "a stereotyped package that contains model elements that have been customized for a specific domain or purpose by extending the meta model using stereotypes, tagged definitions, and constraints".

The architectural description is usually organized by views that are the representations of a system from the perspective of a related set of concerns [18]. A viewpoint is a specification of the conventions for constructing and using an architectural view. The architectural views are the basis of several design methods, such as in [19], [20] and [1]. Since reliability and availability are execution qualities, the analysis method must exploit the behavioral views of the architecture. The way the views are exploited is entirely method-specific.

The different tasks of systems can be employed in different ways and in different kinds of user groups (e.g. people and other services). Different users can have different frequencies for executing different tasks, which will affect the overall frequency for the tasks. In addition, different users can have different ways of executing tasks. Therefore, the different usages of a system form a complex point of view when estimating the system level reliability and availability.

Software product families are rapidly emerging as a viable and important software development approach. A product family is a set of software systems that have similar or overlapping functionality and properties, as well as having different features that affect the buying behavior of different customer groups [11], [21]. Reliability and availability analysis is specifically important within families because unsuitable architectural choices and the impact of faults can cause extensive and long-term problems that affect all of the products in a given family. Another issue is that within product families, different family members can have different quality requirements. Variability within a family is not necessarily simply functional, but can also play a role in terms of non-functional requirements. Therefore, the analysis method should take into account the variation of architecture, e.g. when selecting different variants in the variation points. In addition, the analysis method should provide reusability for analysis results, so that the results can be reused in analyzing different family members.

The analysis method should provide tool support for different analysis activities. The tools assist the method user in his/her work, thereby not only making the analysis faster, but also preventing human mistakes in the analysis. The analysis tool should be closely related to the architecture modeling tool in order to enable the analysis in connection with the architecture design.

The inputs, outputs, and techniques of the method should be clear and easily available. A complicated analysis process requires extra skills and time from the user, thereby diminishing the benefit of the effort that is put forth in promoting the use of the method. In addition, the possible assumptions and limitations of the method should not be excessively restrictive. Specific limitations can even prevent the use of the method in certain cases.

There should be some kind of evidence regarding the maturity of the method, such as evidence of its use, applicability, benefits, and costs. Additionally, the method should provide a way to validate its results. The validation of the analysis results is about comparing the predicted and measured values. This can be performed after the system implementation phase. The "real" values are measured from the implemented system and subsequently compared with the predicted values to see how well these two match. However, the method should also provide a way to validate its results by way of tracking the requirements prior to system implementation. In bi-directional requirements tracking, the requirements are traced to the architecture and to the results of the analysis, and vice versa. The precision of the prediction should also be validated. This can only be performed by using the method for predicting the reliability of an architecture, and by comparing the predicted values with the values that are measured from an implemented system. The validation of the prediction precision helps to find out how close the analysis model is to the architectural model.

3 Overview of reliability and availability prediction approaches

At a high abstraction level, the reliability and availability analysis methods can be roughly classified into quantitative and qualitative methods. Methods employing quantitative techniques have been used since the 1970s [22]. There is a lot of variation in the quantitative methods; some of the methods are applicable before, and some after, system implementation. System measurement based methods, which focus on failures and down times, are used for analyzing systems already in use and for making predictions on implemented systems that are usually run and tested in a lab. The effort in software reliability growth based models is concentrated on statistical testing, and therefore the models are applicable in the late development phase. There are also some other measurement based approaches, such as [23] and [24], which suggest measures for software artifacts. These types of methods are called black-box approaches, since they ignore the internal structure of software systems. Since today's software systems are based on components and their interactions, these methods do not encompass the reliability and availability prediction of component-based software architectures.

The so-called white-box approaches consider the system's internal structure in reliability prediction, computing the system level reliability based on the reliabilities of its components. Goseva-Popstojanova and Trivedi [25] provide a useful survey of architecture-based approaches, categorizing them into state-based, path-based and additive models. The state-based models use the probabilities of the transfer of control between components to estimate the system reliability, whereas the path-based models compute the reliability of composite software based on the possible execution paths of the system. The additive models address the failure intensity of composite software, assuming that the system failure intensity can be calculated from component failure intensities. The additive models, however, model failure intensities with mathematical algorithms, and therefore do not explicitly examine software architectures. The earliest methods of state-based and path-based models [26], [22] were proposed in the 1970s and new methods have evolved since then. White-box and black-box approaches rely on implemented software artifacts at some level at minimum. However, these implementations can also be estimations, such as an estimated code size or number of machine instructions.

Qualitative analysis methods manipulate knowledge rather than numbers. This knowledge is usually specific for the system under study and can be *explicit*, i.e. documented; or *tacit*, undocumented. The tacit knowledge is only in the designers' mind, which makes the analysis process highly human-dependent and therefore prone to errors. Knowledge can also be *abstract/general*, or *domain/application specific*. Recently, there has been a tendency to document general knowledge, for example, by identifying and using architectural styles and patterns. The effect of architectural patterns on the achievement of certain quality attributes has been discussed in several studies, such as [11], [27], and [10]. There still exists a considerable lack of architectural styles and patterns that concentrate on solving the problems of reliability and availability. Currently, only one style, the Simplex ABAS (Attribute Based Architectural Style) [12], focuses on software reliability.

There is a large variety of knowledge-based methods available. Failure mode and fault tree based analyses look at how a system or software can fail, and what the faults are that may cause these failures. Another class of methods is based on heuristics derived from software engineers' experience in developing high quality software. For example, the non-functional requirements (NFR) framework [5] and the design rationale concept, possibly in conjunction with the use of architectural styles and patterns, can be used to map requirements to design decisions, and to document and justify these decisions. This can be used in turn for evaluating how well, and how completely, the architecture addresses the requirements. The existing methods are essentially design methods and should therefore be enhanced with a more elaborated analysis part. Another set of heuristics based methods concentrate specifically on the analysis of software architecture, such as the Scenario-Based Architecture Analysis Method (SAAM) [28], SAAM founded on Complex Scenarios (SAAMCS) [29], SAAM extended by its integration into the domain (ESAAMI) [30] and the Architecture Tradeoff Analysis Method (ATAM) [31]. These methods have different goals, and are focused on multiple software attributes such as modifiability, flexibility, and performance. Today, there are several scenario-based reliability analysis methods. These methods, however, use computations in the analysis and are therefore, regarded as quantitative analysis methods in this study.

4 Comparison of reliability and availability prediction methods

4.1 Scoping of comparison

Since the number of reliability analysis methods presented in the literature is large, we have to restrict the scope of our survey. We therefore define the criteria

for the selection of approaches for this survey and concentrate only on the methods that fulfill these criteria. According to these criteria, an analysis approach has to

(a) concentrate specifically on software reliability and/or availability,
(b) be based on architectural models,
(c) provide a clear, applicable analysis method, and
(d) take the user/stakeholder-centric approach on software and its analysis.

There are several reliability approaches available in the literature that fulfill some or all of these criteria. We introduce a number of different methods and approaches on reliability and availability prediction and compare the most interesting and promising ones based on the framework of Sect. 2.

As described in Sect. 3, several analysis methods exist that have different goals and that focus on one or multiple quality attribute(s). We only consider those methods here that especially focus on reliability and/or availability. A *method* is considered here to consist of one or more techniques, together with rules or guidance criteria for their application. Typically, a method provides a clear set of steps and guidelines on how to apply the method in each step. Furthermore, we only consider the methods that are applicable in the early stages of software development, mainly at the software architecture level.

While examining the broad range of analysis methods, it could be established that architectural models can be used in several ways in reliability prediction. Some methods use the models as such to collect information for analysis, i.e. the analysis models are derived from the architectural models. Other approaches select the models to use and describe the system based on the reliability aspects. Then, there are such methods that are based on the fact that in order to develop a universal method for reliability prediction at the architectural level, the architecture notation must be extended. Accordingly, the architectural models should also be able to represent the reliability domain, in which way they further support reliability analysis. The simplest way to extend UML is to merely represent additional information in UML models using annotations. The UML standard can also be extended by specific profiles to support the required quality aspects.

User or stakeholder centricity is a key issue in several analysis approaches. User-oriented reliability is, according to Cheung [32]: "Reliability of a program (in a certain user environment) defined as the probability that the program will give the correct output with a typical set of input data from that user environment". Accordingly, a user profile summarizes the dynamic characteristics of a typical execution of the program in a particular user environment. An operational profile of a system is an external user-oriented model that specifies the intended usage of a system as a set of input events and their associated probabilities of occurrence expected in actual operation [33], [34]. The faults of software have a different impact on the reliability of the software, depending on how frequently the faulty code is executed [33]. Operational profiles and usage profiles, which are usually abstractions from component execution details, are commonly modeled as Markov chains [32], [35]. Markov chains are generally Finite State Machines (FSMs) that are extended with inter-module transition probabilities as the user profile.

Use cases and scenarios are a means for requirements engineering to capture system requirements [36]. Use cases are used to define usage scenarios for different system users, thereby defining the external requirements on system capabilities. Use cases are described with a Use Case Diagram (UCD) that provides a functional description of a system, its major scenarios (i.e., use cases), external users called actors, and a graphic description of how external entities (actors) interact with the system. A scenario is a brief description of a single interaction of a stakeholder with a system. While use cases focus on runtime behavior with the stakeholder as the user, scenarios also encompass other interactions with the system, such as a maintainer carrying out modification. Scenarios should illustrate the kinds of activities that the system must support, and the kinds of changes that are anticipated to be made to the system [7]. Message Sequence Charts (MSC) and Sequence Diagrams (SD) are widely accepted notations for scenario-based specifications.

4.2 Comparison of the selected methods

The state-based and path-based models described in [25] use architecture models to derive special analysis models. Depending on the manner in which the model is solved, the state-based analytical models can be *composite* or *hierarchical*. Composite models characteristically combine architecture and failure behavior into a single model. The major drawback of the state-based composition models is that since a large software system can have hundreds of states, the repeatable use of this kind of analysis is rather time consuming and cumbersome. Cheung [32] provides a *user-oriented* software reliability model, according to which the reliability of a system can be computed as a function of both the deterministic properties of the structure of a system and the

stochastic properties of the utilization and failure of its components. Basically, Cheung's model is a Markov reliability model that models the *composite structure* of a system as a control flow graph of a program. The model forms a basis for several recent studies analyzing systems by using states and state transitions.

The approach of Reussner et al. [2] assumes that components rely on other components of the environment, which furthermore use the properties of underlying hardware. The aim of this method is to predict component reliability through the c *ompositional analysis* of usage profiles and the reliability of environment components. The compositional analysis in this context consists of the reliability analysis of components, reliability analysis of architecture, for example, connections between components, and reliability analysis of the environment (mostly 3rd party components). We subjected this approach to further examination (see Table 3) because this approach has been proposed for service architectures and is one of the few methods that do not take component reliability as given. Rodrigues et al. [37] present a scenario-based approach on reliability prediction, in which the more fine-grained system architecture model is synthesized for computing a reliability prediction. The approach is based on scenario specifications and Cheung's user-oriented software reliability model [32]. The approach utilizes a high-level message sequence chart (HMSC), which is annotated with scenario transition probabilities derived from the operational profile of the system. This approach is also brought into further inspection (see Table 2), since it is largely stakeholder-centric, annotating scenario specification with probabilistic properties.

Hierarchical models divide reliability analysis into two separate steps. Littlewood [26] proposes a *hierarchical model* that represents the software architecture by a semi-Markov process and the failure behavior of each individual component by a Poisson process. Similarly, Thomason and Whittaker [38] represent the software usage probabilities and the likelihood of failure at a state by a time-homogeneous Markov chain. The probability for the counts of failures in long execution intervals is approximated as a Poisson distribution. One of the major drawbacks of hierarchical models is that they only provide approximations, and therefore, they are not as accurate as the composite models. Hence, these methods are not selected for further examination. Gokhale and Trivedi [39] propose an *accurate hierarchical model*, which assumes that architecture is modeled using a DTMC (discrete time Markov chain). The model is analyzed to obtain the mean and variance of the number of visits in a transient state, which are then combined with the reliabilities of individual modules for reliability prediction. If this analysis is to be employed during the design phase, the architecture information, such as the inter-component transition probabilities, can be obtained, for example, from the occurrence probabilities of various scenarios based on the operational profile of the system as is illustrated in [40]. The predictions achieved with the Gokhale and Trivedi method [39] are closer to the accuracy of the composite models.

The *path-based approaches*, such as [22], [41], [40] and [42] focus on running the software for various inputs. For each run, the resulting execution path is specified in terms of sequences involving components and connectors. The reliability of the software is a weighted average of the reliabilities of all the paths [22]. The Krishnamurthy and Mathur model [41] estimates the reliability of a sequence of components executed in each test run and subsequently calculates the average of all the test runs. Gokhale and Trivedi [42] also propose a path-based approach to architecture-based software reliability prediction, removing the assumption of independence among the successive executions of the components by proposing a solution based on the failure intensities of components.

The execution paths can be defined with the help of simulation when the application code is unavailable. The simulation reveals the paths that are likely to be executed when the software is exercised. Gokhale et al. [43] propose a discrete-event simulation that is capable of capturing a more detailed system structure. The simulation is based on a control-flow graph of the application, where the application architecture is specified by inter-component transition probabilities. The simulation procedure returns the number of time units that are necessary for a single execution of the application. The model presented in [40] is a *scenario-based* probabilistic model, which is applicable for high-level designs. This model is specific to component-based software whose analysis is based on execution scenarios. The Scenario-Based Reliability Analysis (SBRA) method provides the Component Dependency Graph (CDG) model, which represents in turn the components, component reliabilities, link and interface reliabilities, transitions, and transition probabilities. The CDG employs a similar representation of architecture to that of the state based models; the difference can be found in the way the solution is achieved. A tree-traversal algorithm is introduced to analyze the reliability of the system. Due to the diverse attributes of the CDG, this approach is also subject to further examination (see Table 2).

We found several (state-based and path-based) approaches that were based on annotations or extensions of the *UML notation*. Leangsuksun et al. [44] propose a modeling framework, a method, and a tool for

Table 2 Comparison summary of reliability analysis methods (1/2)

Elements	Cortellessa et al. [46]	Rodrigues et al. [37]	Yacoub et al. [40]
Goal	To predict system reliability based on component and connector failure rates	To predict software system reliability taking into account the component structure that is exhibited in the scenarios and concurrent nature of systems	To analyze the reliability of component-based applications as a function of their components and interfaces
Scope of applicability	Focused on the application layer but can also be applied to other layers of software	Not designed for any specific software layer	Designed to the application layer; can be applied to other layers as well
Application domain independency	Domain undefined, may be applied to any domain	Domain undefined, may be applied to any domain	Targeted to all component-based applications
Component-specificity	Assumes that reliability estimates of the components are available	Assumes that reliability estimates of the components are available	Assumes that reliability estimates of the components are available
Architecture-specificity	Distribution is considered (deployment diagram), different types of interaction not considered separately	Different interaction types not considered but all of them can easily be described in scenario descriptions. Distribution not discussed	Distribution is considered as link reliabilities. Component interactions are described as dependencies (adapted from control flow graph)
Platform/implementation technology independency	Implementation independent	Implementation independent	Implementation independent
Target group	System architects	Method user not defined, likely targeted to software architects	Software architects
Needed skills	No special skills needed	Familiarity with Markov chains	No special skills needed
Expected benefits	Reliability analysis before implementation. Allows selection of elements with suitable reliability characteristics	Reliability analysis before implementation. Detection of mismatch between behavior and architecture	Detection of the influence of usage scenarios on reliability of components. Application level reliability
Required resources	Fully integrated with UML, the annotation of diagrams requires only slight additional work. Due to the tool support promised some extra time required	First, the scenario annotations must be performed, the rest of the analysis is partly automated. Tool support for synthesis of LTS (Labeled Transition System) models. Several tools for analysis based on Markov chains	Time to estimate the analysis model parameters and construct the CDG depends on the size and complexity of the system. The calculations are automated
Language	UML	MSC as scenario notations, architecture description is not set to any particular notation	UML
Architectural viewpoints	Architecture is modeled with use case, sequence, and deployment diagrams	Behavior is modeled with scenarios	Sequence diagrams are adopted as a means of documenting scenarios
Analysis model	Annotations	Annotations, Cheung's model (i.e. Markov chains)	CDG
System usage	Different user profiles with related occurrence probabilities are detected from annotated use case diagram	Composing multiple scenarios from different stakeholders is possible (i.e. scenario specification). Scenario transition probabilities are derived from an operational profile of the system	Based on scenarios. Component execution probabilities assigned to scenarios are similar to the operational profile
Variability	Not supported	Not supported	Not supported

analyzing the reliability from architecture. The modeling framework specifies the failure rate and the repair rate for each component (in this case; nodes) as a tagged name-value pair, which are then annotated to the deployment diagram. The system reliability is calculated using, for example, a fault tree model or a Markov chain model. A Bayesian framework [45] incorporates the annotations of use case diagram and sequence diagram into a reliability prediction model. It uses the prior information about the failure probabilities of components and the known scenarios for the prediction of software system reliability.

Table 2 continued

Tool support	Working currently on a set of automation tools	Tool support exists for the automation of a synthesis of LTS models.	The calculation algorithm is automated
Analysis process	Input: annotations, Technique: calculation formulas, Output: component and system failure probabilities	Input: Annotated MSCs, LTSs synthesized for each component, Technique: Markov model, Output: System reliability estimate, detected implied scenarios	Input: Parameter (attribute) estimates, Technique: CDG, SBRA algorithm, Output: reliability of application as the function of reliability of components and transitions
Limitations	Failure probabilities for components must be available. Independence of failures among different components	Transfer of control between components has the Markov property. Failures are independent across transitions. There is only one initial and one final scenario for a system. Component reliability must be available	Execution profiles of scenarios and component reliability must be available. Does not consider failure dependencies between components or take into account the overall application reliability growth as a function of time
Maturity of method	Validation is based on experimental evaluation performed by the authors	Validation is based on empirical evaluation performed by the authors	An experimental case study is used to illustrate the applicability of the approach
Traceability of R&A requirements	Not supported	Not supported	Not supported
Precision of prediction	Not compared with actual values	Not compared with actual values	Not compared with actual values

An extension to the Bayesian Approach has been proposed by Cortellessa et al. [46]. While the approach uses the same *UML extensions* as the Bayesian approach, it also extends the approach by annotating the deployment diagram. The annotation of the deployment diagram with the probabilities of failure over the connectors among sites enables the reliability model to embed the communication failures. The purpose of annotations is to incorporate information about the expected system usage (i.e. operational profile) and the failure probability distributions of the components and connectors that are used in the system. The annotation-based approach of Cortellessa et al. seems quite versatile, also fulfilling the requirement for user-centricity, and is therefore selected for further inspection (see Table 2).

The Object Management Group (OMG) provides a *UML profile* for modeling the quality of service, and fault tolerance characteristics and mechanisms [47]. The profile includes reliability and availability as QoS characteristics. Cortellessa and Pompei [48] provide a UML extension that enables reliability modeling for component-based systems. Their domain model serves the goal of having a unique profile for Quality-of-Service and Fault Tolerance, but does not consider how to generate a reliability analysis model from these extensions. The approach of Rodrigues et al. [49] provides reliability support for the Model-Driven Architecture. The approach exploits the standard UML and the profiles that were already created, such as the profile for Schedulability, Performance and Time [50] and the UML Profile for EJB [51], and makes an attempt to achieve reliability in such a way that it can be specified in a platform-independent way in the early stages of software architecture design. The approach extends the metamodel using stereotypes, tagged definitions, and constraints, and defines the steps to achieve a platform-independent reliability model. The approach is more similar to a design approach, providing reliability profiles in MDA; the reliability analysis part is planned for future contemplation by the authors [52].

Zarras and Issarny [53] propose a reliability modeling method that describes the architecture based on the behavior and reliability aspects of the system. The architecture is first described, after which success criteria, i.e. abstract descriptions of the behavior expected by the system, are defined through the definition of use case diagrams. The expected system behavior is mapped with the help of a collaboration diagram into a concrete system supported by a subset of the architectural elements that constitute the system. The failure rate, MTTF, and reliability of each of the architectural elements are approximated, and described with the *signal class* that describes the failures generated by a particular architectural element. Finally, the overall reliability of the system is assessed using the Reliability Block Diagram (RBD), which can be derived directly from the collaboration diagram.

We also found some approaches that concentrated on specific software and service development approaches

[54], [55]. The approach of Grassi [54] directly focuses on service reliability, and is therefore examined further according to our framework (see Table 3). The approach exploits a unified service model that helps to model and analyze different architectural alternatives, where the characteristics of high and low level services are taken into account. The approach is based on the idea that a set of components requires and subsequently provides services. By using different types of connectors to assemble the same set of services, the impact on the overall system Quality-of-Service can be experimented by discovering the different ways of architecting the service assembly. The approach of Wang et al. [55] especially discussed architectural styles. The approach provides a model for computing the reliability of heterogeneous systems consisting of various architectural styles. System reliability is analyzed based on the reliabilities of components and connectors. The operational profile is taken into account as transition probabilities between the components. Since styles have a significant impact on how the requirements are met, this approach is also selected for further inspection (see Table 3).

We could find only a few approaches that address *availability*, such as [56], [57] and [58]. The approach presented in [56] is mainly concerned with architectural design and not analysis. The Laprie and Kanoun model [57] also addresses the problem of modeling reliability and availability with respect to various classes of faults. Their approach is one of the few methods that also consider hardware reliability. An analysis method of Ledoux [58] uses state-based modeling and addresses not only reliability indicators, but also availability. This model takes into account the delay for recovery following an execution break, therefore overcoming one of the limitations of other state-based methods that assume that the software is instantaneously operational after a failure.

Tables 2 and 3 represent the detailed comparison results of the selected six methods.

5 Results of the survey

Although there is a large body of literature on software reliability, reliability and availability, as well as other quality attributes, have just recently begun to be addressed at the architecture level by methods, techniques and notations. Design approaches already exist that use quality attributes as primary requirements when designing software architecture [1], [59]. It has also been recognized that analysis from the architecture is only possible if the architecture is represented in a way that enables the analysis [8]. A standard notation extension is required in order to unify the different analysis methods and to avoid the development of an enormous amount of separate annotation and extension techniques. The UML standard has already been extended by specific profiles to support quality attributes [47], and especially reliability and availability [48], [49]. The common trend seems to support the needs of future distributed systems.

All of the surveyed methods require some additional work, mostly regarding the development of an analysis model or application of mathematical algorithms. It is obvious that approaches closer to UML require less additional work as UML being a widely used standard, and therefore, are more familiar to architects working in industry than the approaches that require a separate analysis model. However, approaches that are far away from UML do not always require more additional work, if the design and analysis methods are compatible. The most important issue is the design method to provide the information required for the analysis method. The used design method should be taken into account when selecting the analysis method, since the analysis is made in connection with the architecture design. Therefore, it could be argued whether or not the use of UML is rational, if the analysis model is hard to built and the required properties for the analysis are difficult to provide. However, the UML standard offers many other benefits that should be carefully considered.

It is also obvious that more tool support is needed in order to make reliability prediction a fluent part of software development. Currently, there are several tools available that support at least the analysis that is based on Markov chains [60]. However, new analysis tools are required that could be used in connection with modeling tools to bridge the gap between architecture modeling and analysis.

We could not find any method that would also consider variability in the analysis; therefore, no method is applicable for software family engineering. None of the available methods recognizes the variability in reliability requirements, or in architecture descriptions. It would be possible for these methods to consider variability if the architecture descriptions considered variability. Some elements of today's software technology already encourage families of products. The UML notation has already been extended to support the development of product family architectures, as presented in [61]. The extension provides a profile that describes a variability modeling technique for architectural elements. Additionally, MDA is looked upon as a promising approach in software product family architecture development [14]. MDA promotes an efficient use of system models and planned reuse of assets in software development, which can be regarded as a move towards software product families.

Table 3 Comparison summary of reliability analysis methods (2/2)

Elements	Reussner et al. [2]	Grassi [54]	Wang et al. [55]
Goal	To predict system reliability through compositional analysis of usage profiles and the reliability of environment components	To predict the dependability (inc. reliability) of an assembly of pre-existing independently developed services	To predict the reliability of heterogeneous systems according to reliability of each component, operational profile and the architecture of software
Scope of applicability	Proposed for service architecture but may also be used for other layers	Intended for service-oriented computing (SOC) systems	Designed mainly for the application layer but is applicable to the other layers
Application domain independency	Domain undefined, may be applied to any domain	Domain undefined, may be applied to any domain	Domain undefined, may be applied to any domain.
Component-specificity	Reliability of a component is computed as a function of the usage profile and the reliability of external services. Can also be used for black-box components	Assumes that reliability of basic resources (i.e. services that do not require other services) is known. Predicts reliability of complex resources (i.e. services that require other services to carry out their own services)	Assumes that reliability estimates of the components are available
Architecture-specificity	Use and control interactions are supported. Applicable to open, distributed systems (hierarchical kens define distribution boundaries)	Interactions are described as flows of requests between services. Distribution is supported as flows associated with connectors	Interactions are described as transitions between components. Distribution is not considered
Platform/ implementation technology independency	Implementation independent	Implementation independent. However, based on pre-existing services. Resources are not limited to software resources	Implementation independent
Target group	Software integrators	Service assemblers	Software architects
Needed skills	Familiarity with Markov chains	Familiarity with Markov chains	Familiarity with Markov chains
Expected benefits	Reliability analysis of components, architecture, and environment	Enables to select reliable services when assembling services.	Enables to analyze the reliability of a system that combines heterogeneous architectural styles
Required resources	Development of Markov chains for kens/composite kens. Not easily applicable (time-consuming) if the calculations are not automated	Development of three different models. Not easily applicable (time-consuming) if the calculations are not automated	Transformation of the architectural views into state views, computation of reliability and transition probability of each state, integration of the views. Time-consuming method if the state views do not already exist, or tool support is not provided
Language	Uses RADL, can be applied to UML as well	Does not require any specific architecture notation	Does not require any specific architecture notation
Architectural viewpoints	Describes architecture as a composition of kens	Describes architecture as an assembly of services	Architecture described as components and connectors
Analysis model	Markov chains	Flows of request are modeled by a discrete time Markov chain	Markov chains
System usage	Usage profiles are modeled as probabilities of calls to a provided service in a certain state	Services of complex resources are characterized by a flow modeling the usage profile of other services	Operational profile is taken into account as transition probabilities between components
Variability	Not supported	Not supported	Not supported

Although many of the methods have been around since the 1970s, no accurate conclusions could be made about their maturity. In many cases, the validation of methods is based only on the authors' experiments and evaluations in laboratory circumstances. Except for the approach of [2], the comparison of the predicted values against the actual, measured values was not done. There is a great lack of publications covering large scale, industrial applications of the methods. Due to this, the applicability of the methods and the precise cost estimation

Table 3 continued

Tool support	Not provided	Not provided	Not provided
Analysis process	Input: the reliability of basic kens, service FSMs, and usage profiles of provided services, Technique: Markov chains, Output: Service reliability and overall reliability	Input: failure information of the service flows, Technique: Markov chains, flow model, Output: reliability of a service as the reliability of the services it requires	Input: integrated global state view of the system, Technique: transition Matrix, Output: reliability of the system
Limitations	Requires certain data for the architectural kens. Failures of services are independent	The failure rate of basic resources is known. Each request in a state must be fulfilled according to some completion models before a transition to the next state can take place	Assumes that the reliabilities of components and connectors are independent of the transition probabilities
Maturity of method	Validation is based on empirical evaluation performed by the authors	Validation is based on a laboratory example used for illustrating the approach	Validation is based on experiments performed by the authors
Traceability of R&A requirements	Not supported	Not supported	Not supported
Precision of prediction	In the example system, the deviation of the prediction from the measured value is below 1%	Not compared with actual values	Not compared with actual values

of using the methods could not be defined. We assume, however, the missing industrial applications indicate that the cost of using the methods is higher than the advantage they provide. One reason for not using the methods in industrial settings is the missing traceability observed while comparing the methods. None of the existing methods provides traceability of R&A requirements to predicted R&A, against which the measured R&A should be compared. Application of the methods in industry requires precision in all development phases; in specifying R&A requirements and transforming them to the architectural models, in R&A prediction at the architecture level and while measuring actual values from the running systems. If all of these phases are well-defined and supported by appropriate tools, predictions are strict and useful, the cost is likely to be low, i.e. a few working days or weeks, and the methods would be extensively used in software industry.

In several approaches, component reliability was assumed available. There are a number of approaches that especially analyze the reliability of components, such as [62], [63] and [64]. Except the approach of [2], the analysis approaches studied above do not analyze component reliability, or do not consider the effect of a component's internal behavior on its reliability. This could even mean that a separate component reliability analysis method is required to complement the system analysis method. This also means that the analysis methods used for component and system level analyses have to produce meaningful output for each analysis phase. Roshandal and Medvidovic [65] have introduced a stochastic reliability estimation model that takes into account for the uncertainties associated with early development stages, such as the lack of knowledge about the operational profile. The reliability of a component depends on the sequence of executions of its service and the reliability of the individual states contained therein. The estimation of overall system reliability in a compositional manner using component reliability values is the focus of the ongoing work of the authors of [65].

The literature that addresses the availability analysis methods is very scarce; except two methods [57,58]. The availability analysis has not been studied, or at least, we could not find any evidence. One reason is the confusing definitions of the ISO/IEC 9126-1 quality model [66] that defines reliability as the capability of a software system to maintain a specified level of performance when used under the specified conditions. According to the quality model, reliability is mixed with performance, and availability is a sub-characteristic of reliability. This partly explains why only a few availability analysis methods exist. However, future software systems are service oriented and the availability of services is a critical quality factor for systems end-users. Therefore, availability is an issue that requires specific consideration, extensive research, and development of the appropriate analysis methods, techniques, and tools.

One of the major deficiencies of the surveyed approaches was that they did not commit themselves to R&A requirements at any level. Therefore, they failed to define how R&A requirements could be transformed into different architectural decisions and how architec-

tural decisions could be traced back to requirements. This makes it difficult to validate the results of the analysis against any requirements or evaluation criteria. Since R&A prediction is not only about analyzing, but also deriving the R&A requirements into the architecture and describing the architecture in a way that enables analysis, the recent methods are insufficient for R&A prediction as such.

Operational profiles are the research theme in several studies that concentrate on profile development or reliability modeling issues using profiles, such as [33], [67], [68] and [69]. The different analysis approaches, however, seem to apply the profiles arbitrarily. There are, in fact, no common practices or guidelines available regarding how the profiles should be developed or used in the architecture development. Runeson and Regnell [34] propose the integration of an operational profile and a use case model. The use case model can be transformed into an operational profile model by taking the information from the use case model and building the operational profile model based on the information and additional information from other sources, or the use case model can be extended to include the information necessary for the operational profile. This would decrease the modeling and maintenance effort as well as the risk of inconsistency.

The comparison showed that none of the studied methods would perfectly suit the task of predicting reliability at the architecture level. Some of the methods, however, partially matched the requirements defined in our framework. Most of the methods seemed to share the same weaknesses, such as the lack of tool and variability support, and a weak validation of the method and its results. Furthermore, the methods failed to entirely bridge the gap between reliability requirements and analysis, which is, in fact, one of the major demands of "prediction". Therefore, it is impossible to say which of the six compared methods best suits architecture level prediction; none of them is applicable as such.

Based on the comparison, all six methods can be applied to R&A evaluation at the architecture level, because of their independence on application domain, platforms and implementation technologies. However, the most promising approaches are the methods presented in [2], [40], and [46]. They are the most beneficial ones at the architecture level (cf. Table 2, e.g. columns 'needed skills', 'language' and 'required resources'), each with a different approach to reliability prediction. The methods proposed in [2] and [40] address component and system reliabilities, link/interface reliabilities, and transition probabilities and reliabilities, considering also the distribution of systems. In addition, the method proposed by Reussner et al. [2] exploits usage profiles and environment components, and thus predicts the system reliability in the deployment environment of the system. The method put forth by Yacoub et al. [40] describes the dependencies between components, and analyzes the reliability of the system as a function of reliabilities of its components and component interactions. The algorithm of the method for analyzing the CDG model is automated; the parameter estimation for the model, however, is not. The method introduced in [46] is more user-friendly, basing on the annotations of the well-known UML models. The definition of annotations does not require extra skills from the architect; the computations, however, need to be automated to enable a rational use of the method. The developers of the method have also promised to provide tool support.

Based on our literature survey, we conclude that the current analysis methods have several shortcomings limiting their use in industrial settings. Furthermore, quantitative methods alone cannot provide a comprehensive prediction of the reliability and availability of a system; the architect's knowledge and tacit knowledge should be exploited. Consequently, quantitative methods should be used together with qualitative ones, which would allow them to complement each other.

The comparison process using the framework was straightforward and simple. The framework is a valuable tool for anyone searching for an applicable analysis method. Based on the comparison using the framework the best suitable analysis method can be selected. The framework assists to pay attention to important issues of the analysis methods from the viewpoint of software architecture. Although the framework was not applied to availability analysis methods, we believe that the framework is suitable for the evaluation methods of any quality attribute because its elements have been defined according to the needs of architectural evaluation, not from the viewpoint of any specific quality attribute. The framework also takes into account variability, the specific characteristic of product family architectures that are increasingly applied to software intensive systems in industry.

6 Conclusions

The prediction of system reliability and availability requires that the R&A requirements are derived in a specified way into the architectural decision and the needs of the analysis to be taken into account already in the architecture modeling phase so that the analysis can be performed directly by exploiting the architectural models. In this paper, a framework was defined for comparing existing reliability and availability analysis methods from the software architecture point of view.

The comparison of the methods revealed that none of the studied methods alone could provide adequate support for predicting reliability and availability from software architecture. We discovered that there were several methods available with different approaches to R&A, which, however, still had serious shortcomings restricting or preventing their application in the industry. The most common shortcomings were a lack of support for tools and variability, weak reliability analysis of software components, and weak validation of the methods and their results. In addition, there was no proof of the maturity of the methods as they were not validated or used in the industry. The methods seemed to focus on analyzing systems by computing, for example, the probability of failure, or some other measures. While R&A prediction should fill the gap from requirements to analysis, the methods could not be used to track requirements to architecture or to validate whether or not the architecture would meet the defined requirements.

In summary, future research activities are needed for developing availability analysis methods applicable for service oriented architectures, a standard notation describing reliability, availability and their variations in architectural descriptions, and for improving architecture modeling and analysis tools which are needed for providing architects with an integrated working environment. The main benefit of an integrated environment is that it enables the achievement of a better traceability of reliability and availability requirements, and therefore, a better applicability of the methods for large software products in the industry.

There exist some methods that can be applied in the industry as soon as their shortcomings have been removed. In all, it seems that the tendency is moving from traditional R&A analysis towards architecture level prediction. The prediction of reliability and availability provides benefits that are visible in both product quality and production efficiency, as long as the prediction is fluently integrated with software architecture design.

Acknowledgments This work was carried out at the VTT, Technical Research Centre of Finland, within the ITEA project ip02009, FAMILIES, as part of the Eureka Σ! 2023 Programme.

References

1. Purhonen, A., Niemelä, E., Matinlassi, M.: Viewpoints of DSP software and service architectures. J. Systems Softw. **69**(1–2), 57–73 (2004)
2. Reussner, R.H., Schmidt, H.W., Poernomo, I.H.: Reliability prediction for component-based software architectures. J. Systems Softw. **66**(3), 241–252 (2003)
3. Avizienis, A., Laprie, J.C., Randell, B.: Fundamental Concepts of Dependability. LAAS-CNRS. p. 21 (2001)
4. Matinlassi, M., Niemelä, E.: The impact of maintainability on component-based software systems. In: Proceedings of the 29th Euromicro Conference. Antalya, Turkey (2003)
5. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional Requirements in Software Engineering. Kluwer, Boston, Dordrecht 439 p (2000)
6. In, H., Boehm, B., Rodgers, T., Deutsch, M.: Applying WinWin to quality requirements: a case study. In: Proceedings of the 23rd International Conference on Software Engineering. Toronto, Canada (2001)
7. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Reading, 452 p (1998)
8. Jazayeri, M., Ran, A., van der Linden, F.: Software Architecture for Product Families. Addison-Wesley, Boston, 257 p (2000)
9. Jayaratna, N.: Understanding and Evaluating Methodologies: NIMSAD: a Systematic Framework. McGraw-Hill, London, 259 p (1994)
10. Buschmann, F., et al.: Pattern Oriented Software Architecture. A System of Patterns. John Wiley, New York, 467 p (1996)
11. Bosch, J.: Design and use of Software Aarchitectures: Adopting and Evolving a Product-line Approach. Addison-Wesley, Harlow, 354 p (2000)
12. Klein, M., et al.: Attribute-Based Architecture Styles. in WICSA1 First Working IFIP Conference on Software Architecture. San Antonio, USA (1999)
13. Garlan, D., Shaw, M.: An introduction to software architecture. In: Advances in Software Engineering and Knowledge Engineering. World Scientific, Singapore, (1993)
14. Brown, A.: An Introduction to Model Driven Architecture, Part I: MDA and Today's Systems (2004)
15. OMG, Unified Modeling Language (UML), version 1.5. Object Management Group (2002)
16. OMG, Unified Modeling Language (UML) 2.0 Specification. Object Management Group (2003)
17. Barry, D.K.: Web Services and Service-Oriented Architectures: The Savvy Manager's Guide. Morgan Kaufmann, 200 p (2003)
18. IEEE, IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems. Institute of Electrical and Electronics Engineers Inc (2000)
19. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley, Reading, 397 p (1999)
20. Kruchten, P.: The 4+1 view model of architecture. IEEE Softw. **12**(6), 42–50 (1995)
21. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, 608 p (2002)
22. Shooman, M.: Structural models for software reliability prediction. In: Proceedings of the 2nd International Conference on Software Engineering (1976)
23. McCall, J., et al.: Methodology for Software Reliability Prediction. Rome Labs Technical Report, RADC-TR-87-171, Vols I & II (1987)
24. Smidts, C., Li, M.: Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems. Nuclear Engineering Department, Technical Report, University of Maryland (2000)
25. Goseva-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. Perform. Evaluat. **45**(2–3), 179–204 (2001)
26. Littlewood, B.: Software reliability model for modular program structure. IEEE Trans. Reliability **28**(3), 241–246 (1979)

27. Douglass, B.P.: Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. Addison Wesley Professional, 800 p (1999)
28. Kazman, R., Abowd, G., Bass, L., Clements, P.: Scenario-based analysis of software architecture. IEEE Softw. **13**(6), 47–55 (1996)
29. Lassing, N., Rijsenbrij, D., van Vilet, H.: On software architecture analysis of flexibility, complexity of changes: size isn't everything. In: Proceedings of the 2nd Nordic Software Architecture Workshop (1999)
30. Molter, G.: Integrating SAAM in Domain-centric and Reuse-based development process. In: Proceedings of the 2nd Nordic Workshop on Software Architecture (1999)
31. Kazman, R., et al.: The architecture tradeoff analysis method. In: The 4th IEEE International Conference on Engineering of Complex Computer Systems (1998)
32. Cheung, R.C.: A user-oriented software reliability model. IEEE Trans. Softw. Eng. **6**(2), 118–125 (1980)
33. Musa, J.D.: Operational profiles in software-reliability engineering. IEEE Softw. **10**(2), 14–32 (1993)
34. Runeson, P., Regnell, B.: Derivation of an integrated operational profile and use case model. In: Proceedings. The Ninth International Symposium on Software Reliability Engineering (1998)
35. Whittaker, J.A., Thomason, M.G.: A Markov chain model for statistical software testing. IEEE Trans. Softw. Eng. **20**(10), 812–824 (1994)
36. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, ACM Press, 400 p (1992)
37. Rodrigues, G.N., Rosenblum, D.S., Uchitel, S.: Using scenarios to predict the reliability of concurrent component-based software systems. In: 8th International Conference on Fundamental Approaches to Software Engineering, FASE 2005. Springer Lecture Notes in Computer Science, Edinburgh, (2005)
38. Thomason, M.G., Whittaker, J.A.: Rare failure-state in a Markov chain model for software reliability. In: Proceedings of the 10th International Symposium on Software Reliability Engineering. IEEE, Boca Raton, (1999)
39. Gokhale, S., Trivedi, K.S.: Reliability prediction and sensitivity analysis based on software architecture. In: Proceedings of the 3rd International Symposium on Software Reliability Engineering (ISSRE 02). IEEE Computer Society, Annapolis, (2002)
40. Yacoub, S., Cukic, B., Ammar, H.: Scenario-based reliability analysis of component-based software. In: Proceedings of 10th International Symposium on Software Reliability Engineering (ISSRE'99) (1999)
41. Krishnamurthy, S., Mathur, A.P.: On the estimation of reliability of a software system using reliabilities of its components. In: Proceedings of the 8th International Symposium on Sotfware Reliability Engineering (ISSRE97) (1997)
42. Gokhale, S.S., Trivedi, K.S.: Dependency characterization in path-based approaches to architecture-based software reliability prediction. In: Proceedings of the IEEE Workshop on Application-Specific Software Engineering Technology, ASSET-98 (1998)
43. Gokhale, S.S., Lyu, M.R., Trivedi, K.S.: Reliability simulation of component-based software systems. In: Proceeding of the 9th International Software Reliability Engineering (1998)
44. Leangsuksun, C., Song, H., Shen, L.: Reliability modeling using UML. In: Proceeding of the 2003 International Conference on Software Engineering Research and Practice. Las Vegas (2003)
45. Singh, H., et al.: A Bayesian approach to reliability prediction and assessment of component based systems. In: Proceedings of 12th International Symposium on Software Reliability Engineering (ISSRE'01). Hong Kong (2001)
46. Cortellessa, V., Singh, H., Cukic, B.: Early reliability assessment of UML based software models. In: Third International Workshop on Software and Performance. Rome (2002)
47. OMG, UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. Revised submission, Object Management Group (2003)
48. Cortellessa, V., Pompei, A.: Towards a UML Profile for QoS: A Contribution in the Reliability Domain. In: Proceedings of the Fourth International Workshop on Software and Performance. ACM Press (2004)
49. Rodrigues, G.N., Roberts, G., Emmerich, W., Skene, J.: Reliability support for the model driven architecture. In: Proceedings of the ICSE Workshop on Software Architecture for Dependable Systems. Portland (2003)
50. OMG, UML Profile for Schedulability, Performance, and Time Specification. Object Management Group (2003)
51. Greenfield, J.: UML Profile for EJB, in Technical report. Rational Software Corp (2001)
52. Rodrigues, G.N.: A model driven approach for software systems reliability. In: 26th International Conference on Software Engineering (ICSE'04). Edinburgh (2004)
53. Zarras, A., Issarny, V.: Assessing software reliability at the architectural level. In: Proceedings of the 4th ACM SIGSOFT International Software Architecture Workshop. ACM, Ireland (2000)
54. Grassi, V.: Architecture-based dependability prediction for service-oriented computing. In: Proceedings of the Twin Workshops on Architecting Dependable Systems, International Conference on Software Engineering (ICSE 2004). Springer, Edinburgh, (2004)
55. Wang, W.-L., Wu, Y., Chen, M.-H.: An architecture-based software reliability model. In: Pacific Rim International Symposium on Dependable Computing. IEEE, Hong Kong (1999)
56. Leangsuksun, C., et al.: Availability prediction and modeling of high availability OSCAR cluster. In: IEEE International Conference on Cluster Computing. Hong Kong (2003)
57. Laprie, J.C., Kanoun, K.: X-ware reliability and availability modeling. IEEE Trans. Softw. Eng. **18**(2), 130–147 (1992)
58. Ledoux, J.: Availability modeling of modular software. IEEE Trans. Reliability **48**(2), 159–168 (1999)
59. Bachmann, F., Bass, L., Klein, M.: Moving from quality attribute requirements to architectural decisions. In: Second International Software Requirements to Architectures, STRAW'03. Portland, USA (2003)
60. Fugua, N.B.: The applicability of Markov analysis methods to reliability, maintainability, and safety. Reliability Anal. Center START Sheet **10**(2), 8 (2003)
61. Dobrica, L., Niemelä, E.: Using UML notation extensions to model variability in product line architectures. In: ICSE, International Workshop on Software Variability Management. Portland (2003)
62. Everett, W.: Software component reliability analysis. In: IEEE Symposium on Application—Specific Systems and Software Engineering and Technology. Richardson (1999)
63. Voas, J.M.: Certifying off-the-shelf software components. Computer **31**(6), 53–59 (1998)
64. McGregor, J.D., Stafford, J.A., Cho, I.-H.: Measuring component reliability. In: Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering. IEEE, Portland (2003)

65. Roshandel, R., Medvidovic, N.: Toward architecture-based reliability estimation. In: Proceedings of the Twin Workshops on Architecting Dependable Systems, International Conference on Software Engineering (ICSE 2004). Edinburgh (2004)
66. ISO/IEC, Software Engineering - Product Quality. Part 1: Quality Model (2001)
67. Ozekici, S., Soyer, R.: Reliability of software with an operational profile. Eur. J. Oper. Res. **149**(Issue 2), 459–474 (2003)
68. Elbaum, S., Narla, S.: A methodology for operational profile refinement. In: Proceedings of the Annual Reliability and Maintainability Symposium (2001)
69. Gittens, M., Lutfiyya, H., Bauer, M.: An extended operational profile model. In: 15th International Symposium on Software Reliability Engineering, ISSRE 2004 (2004)

Author's biography

Anne Immonen is a research scientist at VTT Technical Research Centre of Finland. She received the M.Sc. degree in information processing science in 2002 from the University of Oulu, Finland. Her research interests include design and analysis of software architectures, product families and software quality. The main topics in her current research projects are the quality analysis of software architecture and the methods to verify the quality of architecture. She is also currently performing her Ph.D. studies at the University of Oulu, Finland.

Eila Niemelä obtained the M.Sc. degree in 1995 and the Ph.D. degree in 2000 in information processing science from the University of Oulu, Finland. Before graduation she worked fifteen years as a software engineer of embedded systems and from 1995 to 1998 as a senior research scientist in the Embedded Software research area at VTT Technical Research Centre of Finland. After that she led the Software Architectures Group at VTT until September 2002. Since 2001 she has been working as a research professor at VTT and since 2002 also as a docent of software architectures and components at the University of Oulu. She has acted as a reviewer for several scientific journals and as a member of many conference program committees. Quality-driven architecture design, quality analysis at the architecture level and service architectures of pervasive computing environments are her main research topics. She is a member of the IEEE and the IEEE Computer Society.