Software architecture plays an increasingly important role in the dependability of today’s large and complex software systems. Formal and informal architectural models embody critical design decisions about systems’ functional and non-functional properties. Traditionally, a wide range of software architecture modeling and analysis approaches focused on validating systems’ functionality. More recently, however, many approaches have been developed to predict and estimate dependability of complex software systems based on the system’s software architecture. In this paper, we present a novel classification of architecture-centric dependability modeling and analysis approaches along with a discussion of the specific underlying methodologies and their suitability to assess different dependability attributes. Our classification and discussion can be used to identify new areas of software architecture and dependability research and to identify compatible formalisms that can be used to perform tradeoff analysis among dependability attributes.

Keywords: software architecture; software dependability; software quality.

1. Introduction and Background

The dependability of today’s complex software systems is as important as the functionality that they provide. Building dependable (secure, reliable, safe, available, performable, etc.) software systems requires methodologies, techniques, and tools that address the functional and non-functional properties of the system throughout the software development life cycle. In other words, dependability must be designed and built into software systems from inception.

Software systems’ architecture and design embody critical decisions that govern systems’ functionality. In turn, these decisions can directly impact the quality of software products (Sommerville, 2006; Stevens et al., 1979; Taylor et al., 2008). Over the last two decades, the discipline of software architecture has emerged as a systematic approach to provide representations, visualizations, and analysis based on high-level abstractions of a system’s structure, behavior, and key properties. Important design decisions—from specific interactions among components and connectors in the system, to policies for exception and error handling, to anticipating extensions of the system in the future—are made at the level of software architecture. Furthermore, architectural styles have emerged to offer solutions within specific contexts that elicit beneficial qualities in each system (Taylor et al., 2008).

Wide ranges of approaches to modeling software architecture have been developed, spanning from highly informal lines and boxes diagrams to formal and specialized languages. These approaches commonly focus on system functionality, and vary in terms of the level of detail and the amount of rigor and formality (Taylor et al., 2008). Given well-defined rules, automated tools accompany the formal approaches to software architecture modeling and can be used to assess functional correctness, conformance, consistency, and compatibility of the models. Less formal approaches offer greater flexibility in representation and are used as a critical communication tool among stakeholders. Often though, they lack the rigor required to build tools for automated analyses.
More recently, architecture-based *dependability modeling and analysis* approaches have been developed that use software architectural knowledge to ensure that dependability requirements are addressed in software system design. Additionally, these approaches are used to evaluate the impact of specific architectural design decisions on system dependability. Some of these approaches are informal and rely on the knowledge of domain experts and software architects; others use formal analysis techniques and rely on availability of different sources of information from domain knowledge to simulation results to data obtained from functionally similar software systems.

While these approaches are an important step in bridging the gap between requirements, architectural design, and implementation, important open research problems are still to be explored in this domain. Our long-term research involves design and development of architecture-centric approaches and tools to help predict and assess the quality of software systems throughout the development lifecycle. This work includes development of architecture-based reliability prediction approaches (Cheung et al., 2008; Roshandel et al., 2007); reliability analysis approaches for software product families (Markides et al., 2010) and domain specific techniques for dependability prediction of mobile, distributed, and situated software systems (Cooray et al., 2010). Our research reveals the need for a systematic study of existing architecture-centric software dependability approaches and is motivated by the following two observations: First, many of the existing modeling approaches can be used with different sources of information (e.g., Markov-based reliability modeling may be used in early design or as part of the final system acceptance testing). A systematic knowledge of these approaches can be helpful in identifying situations in which existing techniques can be adapted to new kinds of systems, new domains, and new analyses. Second, given the inter-dependencies between various dependability attributes, analysis of a single attribute, while helpful, does not ensure dependability of a complex system. A more comprehensive dependability analysis framework—in which the tradeoffs among attributes are effectively modeled and understood—is thus necessary. The dependability discussion presented here takes a first step in identifying compatible analysis approaches which may be combined or coordinated to provide tradeoff analyses.

This paper presents a framework for classifying existing architecture-based software dependability modeling and analysis approaches. We use a representative set of existing approaches in order to describe the classification. Our classification and discussion can be used to identify areas of software architecture and dependability research requiring additional research. Moreover, the classification can be used to identify compatible formalisms that can be used to perform dependability tradeoff analysis across multiple dependability attributes. We expect that our classification framework will be adapted and extended to classify approaches in specific domains (e.g., mobile systems, product families).

This paper is organized as follows: In Section 2 we present a brief background on software dependability analysis. We then present our classification framework in Section 3, in the context of representative approaches for each class. We conclude with a discussion of possible new directions in dependability and tradeoff analysis in Section 4.

### 2. Dependability Analysis

The term “dependability” informally refers to those properties of software systems that enable us to rely on their functionality. Avizienis et al. (2004) define dependability of computing systems as their ability to deliver services that can justifiably be trusted. Sommerville (2006) identifies reliability, availability, safety, and security as the four principal dimensions of dependability. A multitude of methodologies and approaches have been developed to evaluate various dimensions of software system dependability as presented in recent surveys (Babar et al., 2004; Balsamo et al., 2004; Dobrica and Niemela, 2002; Gohkale, 2007; Goseva et al., 2003b; Immonen and Niemela, 2007; Ionita et al., 2002). These approaches vary in the set of *information sources* and the kind of artifacts they rely on, as well as the underlying formalism they are based upon. For example, one performance analysis approach relies on the requirements specification and uses use case maps (Petriiu and Woodside, 2002), while another performance analysis approach is based on software architecture specification and uses stochastically timed process algebras (Bernardo, 2000).
We have focused on those dependability analysis approaches that in some way rely on or incorporate architectural knowledge into the assessment process. Software architecture-based approaches emphasize designing dependability into complex software systems. Our classification framework, presented in the next section, serves as a taxonomy of architecture-based software dependability modeling and analysis methods. We focus on a set of representative approaches to describe the classification and later summarize our insights based on the specific dependability attribute. Given the broad nature of the classification presented here, the treatment of the related work reflects only a representative set of approaches within each category. A more comprehensive survey is outside the scope of this paper and may be found in specialized surveys within each sub-discipline (e.g., Balsamo et al., 2004; Goseva et al., 2003b; Immonen and Niemela, 2007).

We begin by briefly and informally defining the subset of dependability attributes we considered in our studies. In this paper, we focus on a handful of principal dimensions of software dependability: reliability and availability, performance, security, safety and modifiability.

Reliability is defined as the probability that the software system will perform its intended functionality within specified design limits (Pham, 2006). Availability is a closely related property and is defined as the probability of a software service or system being available when needed. Architecture-based approaches to software reliability and availability analysis strive to provide a prediction or estimation of a system’s reliability/availability and assess the impact of architectural design choices on system reliability/availability.

Security is an aspect of dependability that relates to the ability of the system to function “correctly” under malicious attack or accidental malfunction (McGraw, 2006). Security is often defined and characterized in terms of its sub-characteristics: confidentiality, integrity, and availability. Architectural design choices can directly impact the vulnerability of the system to malicious attacks.

Safety is a system property and is defined as the freedom from accidents or losses (Leveson, 2005). In the context of safety critical systems, safety is assessed in terms of risk (a combination of the likelihood of an accident and the severity of the potential consequences). In safety critical software systems, the goal of safety analysis is to anticipate failures and reduce their risk. Architecture and design-based approaches model and analyze software system safety using a variety of formal approaches.

Performance analysis involves analyzing the system with respect to a set of related attributes such as response, throughput, and execution time (Balsamo, 2004). The goal of analysis is to measure, monitor, record, identify, and understand areas of poor performance. The system’s software architecture can directly impact its performance. Methodologies have been developed to incorporate performance properties into architectural modeling and analysis, in order to identify the most suitable architectural design for a software system.

Finally, modifiability of a software system is the ease with which it can be modified to changes in the environment, requirements or functional specification (Bengtsson, 2004). A related concept, maintainability, is additionally concerned with the correction of errors and bugs. Architecture-based approaches for software modifiability analysis generally focus on assessing the impact of various architectural design choices with respect to the ease with which they can be modified.

The intricate dependencies among these attributes, together with their relationship with other aspects of dependability (e.g., extensibility, usability, portability), as well as cost, constitute a significant challenge in design and development of dependable software systems. For example, a specific architectural configuration for a software system may be considered to be highly reliable, but could negatively impact the system’s efficiency and performance. Similarly, highly secure systems are desirable but expensive. While many approaches have been developed to assess a single aspect of a system’s dependability, tradeoff modeling is a less explored area of research.

---

1 For each of these attributes, a selection of metrics and evaluation techniques exist. While the definitions provided here are among those widely accepted in the literature, individual approaches may rely on a variation or adaptation of these definitions.
3. Classification

As is evident from the large body of research, software architecture forms a suitable foundation for assessing software dependability. Given that architecture-based dependability analysis is a specialized form of architectural analysis, we have chosen to base our classification on the architectural analysis classification by Taylor et al. (2008). Taylor’s classification presents a cross-section of architectural analysis techniques classified under three categories: inspection & review-based, model-based, and simulation-based. The classification is further elaborated in terms of a set of dimensions that include the goals, scope, models, type, automation level, and stakeholders involved in the analysis process. While Taylor’s classification offers a suitable starting point for classifying architecture-based analysis approaches, it is not detailed enough to be used as a tool to classify dependability-specific approaches.

In this paper, we present an adaptation and extension to the above classification (Taylor et al., 2008) aimed at dependability-specific architectural analysis approaches. Our research shows that dependability analysis approaches also vary based on when they are applied during the software development life cycle. For example, the kinds of models applied in early stages of prototyping and shortly after requirements analysis are fundamentally different from those models that are applied on a pre-release system. We also observed that different analysis techniques may be used with a single modeling technique. For example, Markov modeling is widely used in a variety of settings.

![Figure 1. The classification of dependability analysis techniques](image-url)
However, corresponding analysis techniques vary from a simple steady-state behavioral analysis to Monte Carlo simulation and risk analysis. The observation that several different analysis techniques can be used with a single modeling technique led us to separate between modeling and analysis techniques in our classification, although the two are clearly closely related. Finally, our research revealed that some of the existing approaches combine modeling and/or analysis techniques that fall under different classes within our classification. We call these techniques cross-cutting, as they often constrain a more informal technique and combine it with a more formal technique to achieve their goals.

In the rest of this section, we briefly summarize our classification scheme depicted Figure 1. A detailed description of the classification, in the context of representative approaches for each category, is presented in the next section.

**Category.** Most generally, we distinguish among formal and informal approaches in dependability analysis.

**Information Sources.** The nuances of analysis approaches, and the accuracy and utility of their results generally depend on the sources of information available at the time of analysis. For example, early in architectural modeling and prototyping, most of the information is obtained from domain experts and requirements documents. At this stage, analysis serves as a dependability prediction. Later in development, formal specifications of systems properties, results from simulation of executable models, or perhaps runtime data obtained during execution may be used for automated approaches which could result in a more accurate dependability estimation.

**Class.** Each class denotes a group of architecture-centric modeling techniques. For example, the class of Scenario-based modeling techniques encompasses use-case maps, component dependency graphs (CDG), and abuse cases.

**Modeling Technique.** This corresponds to the specific modeling techniques used to assess software dependability. For example, Queuing Networks are extensively used to model performance-related aspects of the system such as schedulability and response-time.

**Analysis Technique.** One or more analysis techniques may be used on certain modeling techniques. For example, a failure model used to assess the safety of a system may be analyzed using fault-tree analysis methods or the Failure Mode, Effects, and Criticality Analysis (FMECA) technique. Also note that a specific analysis technique may be applied to more than one modeling technique. For example, risk analysis has been applied to both Markov Models (for reliability analysis) and Threat Models (for security analysis).

3.1. Inspection and reviews

**Inspection and review-based** approaches refer to a group of informal, human intensive approaches based on specification of system goals and scenarios. Software architects and other stakeholders work together to evaluate both functional and non-functional requirements. Goal modeling and scenario notations have been systematically used in requirement engineering (Liu and Yu, 2001) and have been adapted to address dependability issues at the software architecture level, as described by Taylor et al. (2008). Goal oriented modeling treats quality attributes as goals to be achieved during the design process. These goals also serve as guidelines in identifying and selecting specific design alternatives. Scenarios present possible ways to use a system to accomplish desired functional goals or, in the case of dependability analysis, desired quality goals (Liu and Yu, 2001). Below, we provide an overview of existing dependability analysis approaches that rely on inspection and review-based information sources.

3.1.1 Scenarios

In the context of software dependability analysis, scenario-based approaches use scenarios to present possible paths for achieving a desired quality goal. Scenarios may be expressed in a variety of forms, including narrative text, structured text, images, simulations, charts and maps, and formal notations (Liu and Yu, 2001). The content of the

---

2 The term cross-cutting used in this context signifies those approaches that cannot be categorized under a single modeling/analysis technique. This is not related to the term “cross-cutting concerns” in Aspect-Oriented Programming.
scenario could describe either events internal to the system or events between the software system and its environment. The most common modeling techniques for the scenario-based class can be summarized as use case maps, component dependency graphs (CDGs), and abuse or misuse cases. A number of scenario-based software architecture evaluation methods have been developed (e.g., Clements et al., 2002; Dolan et al., 2000; Kazman et al., 1994, 1998, 2001; Olumofin and Misic, 2005). The study of scenario-based software architecture evaluation methods performed by Ionita et al. (2002) concluded that these approaches all share a common characteristic in being restricted to a particular class of systems and addressing a limited set of non-functional requirements.

In the domain of reliability modeling and analysis, SBRA (Scenario-Based Reliability Analysis) (Yacoub et al., 1991) was developed for component based software systems. SBRA also relies on CDGs to construct a probabilistic model, based on scenarios of component interactions. An extension of this approach uses the CDG models to develop a risk aggregation model and combines this information with severity analysis, performed with the aid of simulations to develop a reliability risk assessment (Yacoub and Ammar, 2002). This is an example of a cross-cutting approach that incorporates both inspection and review-based and simulation-based approaches in our classification. CDG-based approaches are limited by not considering failure dependencies among components and requiring information about execution times and component usage patterns.

A prominent approach within this class is the Software Architecture Analysis Method (SAAM) (Kazman et al., 1994). SAAM was created to assess architecture modifiability, but has also proven useful for quickly assessing other quality attributes, including portability, extensibility, integrability, and functional coverage. The Architecture Trade-off Analysis Method (ATAM) (Kazman et al., 1998), an enhanced version of SAAM, provides insight into quality attribute interdependencies. SAAM and ATAM help increase stakeholders’ knowledge of the architecture as it relates to the quality attribute being evaluated, often resulting in an improved architectural design. However, neither of the two methods, nor their extensions (e.g., Holistic Product Line Architecture Assessment (HoPLAA) (Olumofin and Misic, 2005)), Cost Benefit Analysis Method (CBAM) (Clements et al., 2002; Kazman et al., 2001), Architecture Level Modifiability Analysis (ALMA) (Bengtsson et al., 2004), and Family-Architecture Analysis Method (FAAM) (Dolan et al., 2000)) provide a clear quality metric for the attributes being analyzed. Although these tradeoff models are effective in representing relationships among several quality attributes, they all rely on availability of implementation and runtime operational profile to perform dependability analysis.

Scenario-based approaches may also be applied at the very early stages of design. For example, ARID (Active Reviews for Intermediate Designs) (Clements, 2000) is used at the inception stage to assess the suitability of various components. Furthermore, ALPSM (Architecture Level Prediction of Software Maintenance) (Bengtsson and Bosch, 1999) analyzes the maintainability of a software system architecture by assessing the size of changes as a predictor for the effort needed to adapt the system into a given architecture-based scenario. Finally, a more specialized technique in this category is Scenario-Based Architecture Reengineering (SBAR) (Bengtsson and Bosch, 1998), focusing on architecture-based scenario evaluation of multiple quality attributes for the architecture of an existing system. This approach aids transformation of a software architecture in order to meet its quality requirements. It evaluates each quality attribute by relying on appropriate analysis techniques such as simulation, mathematical models, and experienced reasoning.

Another modeling technique used within this class is Use Case Maps (UCMs), a well-developed scenario language that expands UML Use Cases into scenarios (Buhr, 1998; Buhr and Casselman, 1996). UCMs are more powerful than use cases and UML behavioral diagrams in that they can combine interacting scenarios. The UCM Navigator (UCMNav) was developed as a tool support for UCMs to create large industry-scale scenario specifications (Miga, 1998; Moriconi et al. 1997). Petriu and Woodside (2002) describe an add-on to UCMNav called UCM2LQN that converts UCM scenario models to layered queuing models for performance modeling. Specifically, UCM2LQN converts the UCM specifications to Layered Queuing Networks (a stochastic specification-based approach in our classification, as discussed later in this paper). This is another example of a cross-cutting modeling technique.

In the domain of security analysis, Pauli and Xu (2005) present an approach to architectural design and analysis of secure systems using abuse cases. This approach uses security requirements in the form of abuse cases to guide
architectural design. The approach offers a smooth transition from requirements elicitation to high-level architecture design and improves the traceability of security concerns, but it lacks a formal specification language.

3.1.2. Goal driven

Goal-oriented modeling approaches are based on description of system objectives obtained from different stakeholders. In the context of dependability related goals and objectives, existing modeling techniques can be divided into Non-Functional Requirement Frameworks and Goal Trees.

Non-Functional Requirements (NFRs) refer to quality or dependability needs, such as those related to accuracy, performance, security, and modifiability. To enable software architects to deal with a system’s diverse NFRs, Chung et al. (2000) presents the NFR Framework—a structured graphical facility for describing NFRs, managing them by refining interdependent NFRs, justifying decisions, and determining their impact on various aspects of the system design and development. NFRs are represented as “softgoals” in the NFR Framework, whose interdependencies are captured using graph notation. The impact of design decisions is qualitatively propagated through the graph to determine how well a system’s software architecture satisfies its target NFRs.

Subramanian and Chung (2001) treat software adaptability (the extent to which a software system can be adapted to changes in its environment without external intervention) as a goal to be achieved in development. This NFR Framework allows for consideration of design alternatives, analysis of tradeoffs and rationalization of design decisions all in relation to the stated goals.

Jurjens (2002a) presents a method aimed at secure systems development using a UML extension for security (UMLsec) (2002b) and goal trees. The goal tree approach enables use of object-oriented analysis results for functional requirements, and complements those with the formulation of goals especially suited to security concerns. Additional discussion of UMLsec may be found in section 3.2.1.

Finally, a related approach proposed by Duenas et al.(1998), called Software Architecture Evaluation Model (SAEM), uses the Goal-Question-Metric (GQM) technique to verify whether specific quality goals are satisfied based on the set of related quality metrics. The analysis process is divided into external and internal processes, corresponding to the users’ and developers’ views, respectively. As part of this process, specified quality requirements are mapped into internal quality attributes using the experts’ knowledge and the companies’ accumulated data.

3.2. Specification based

The next set of approaches in our classification scheme relies on more formal information sources, such as formal (or semi-formal) specification techniques accompanied by automated analysis tools. Specifically, we classify approaches under the specification-based information source if they principally rely on design-level knowledge. This is in contrast to simulation-based approaches (Section 3.3) that rely on executable models of the system and can perhaps more closely emulate the actual runtime behavior of the system.

Modeling techniques using this information source vary widely based on the properties of the system they represent and the specification languages. These models may represent static aspects of the system property (such as interfaces, pre- and post-conditions) or dynamic aspects of the system (such as data flow or control flow). The analysis techniques enabled by specification-based modeling techniques may be quantitative, qualitative, or a hybrid of the two.

Specification-based approaches are further grouped into four classes as described below.
Several architecture-based dependability analysis approaches are based on extensions to existing specification and modeling languages and notations, such as Architecture Description Languages (ADLs) (Taylor et al., 2008) or the Unified Modeling Language (UML) (OMG, 1997). Many ADLs as well as UML offer extension mechanisms that allow creation of domain specific languages as well as representation of system’s non-functional properties.

While ADLs vary in their degree of formality, they are often accompanied by automated tools that allow for analysis of functional properties, consistency, and completeness. UML-based approaches, on the other hand, often lack the rigor needed to support automated analyses. Subsequently, many UML-based dependability analysis approaches have been developed that rely on UML’s representational power, but focus on limited use of the UML semantics in order to develop rigorous analyses. Some of these approaches use annotations (Cortellessa et al. 2002; Leangsukun et al., 2003a; Singh et al., 2001) while others build extensions to the UML meta-model (Espinoza et al., 2006; Gu and Petriu, 2002; Jurjens, 2002b; OMG, 2002, 2003; Petriu and Shen, 2002; Rodrigues et al., 2003; Williams and Smith, 1998) to represent required quality aspects.

The Object Management Group (OMG) is an important contributor in this domain by providing several UML profiles for the integration of UML models with quality properties. OMG provides a UML profile for modeling the quality of service and fault tolerance characteristics and mechanisms (OMG, 2003) as well as another profile for Schedulability, Performance and Time (SPT) (OMG, 2002).

Many approaches extend the OMG’s UML profiles to incorporate new dependability prediction methods as summarized below. In the context of performance modeling, Petriu and Woodside (2004) highlight shortcomings of the SPT profile due to its weakness in capturing a system’s behavioral representation and introduce the Core Scenario Model (CSM). While CSM currently only provides support for performance analysis, Woodside et al. (2009) argue that it can be further extended for reliability and schedulability analysis. Moreover, Petriu et al. (Gu and Petriu, 2002; Petriu and Shen, 2002) have proposed approaches to transform UML models into performance models, based on meta-model extensions. Their graph-grammar based method automatically transforms a UML model annotated with performance information into a Layered Queuing Network (LQN) (Petriu and Shen, 2002). Their XSLT-based solution eliminates the need to build the internal data structure of the UML model, but its inability to be integrated with any UML tools (Gu and Petriu, 2002). Another related approach (Williams and Smith, 1998) applies the Software Performance Engineering (SPE) methodology to evaluate the performance of a software architecture specified using UML class, deployment and sequence diagrams, and Message Sequence Charts. Williams and Smith (2002) later generalize their approach in PASA (Performance Assessment of Software Architectures) to determine whether a software architecture meets its required performance objectives. Finally, PRIMA-UML (Cortellessa and Mirandola, 2000) (an extension to (Williams and Smith, 1998)) elicits information from different UML diagrams in order to incrementally generate a performance model.

In reliability analysis, Rodrigues et al. (2003) extend the UML meta-model to develop a platform-independent reliability prediction model to be used in the early stages of software architecture design. Leangsuskn et al. (2003a) offers a UML-based modeling framework for reliability analysis that annotates UML deployment diagrams with failure and repair rate for each component and constructs statistical fault trees and Markov Chain models. In another approach, a Bayesian framework is integrated with annotated UML models (use case and sequence diagrams) (Singh et al., 2001). An extension of this approach further annotates the deployment diagrams in order to include the probabilities of hardware/logical connector failure (Cortellessa et al., 2002).

In the domain of security modeling, Jürjens (2002b) introduces UMLsec, an extension of UML that allows expression of security relevant information within the system specification diagrams. UMLsec uses precisely defined semantics to describe the behavioral aspect of the system, with a formal notion of an adversary. It includes formalizations of basic security requirements, descriptions of threat scenarios, and security concepts such as encrypted communication links. In addition to this, SecureUML (Lodderstedt et al., 2002) extends the UML meta-model to define Role-based Access Control (RBAC) while AuthUML (Alghathbar and Wijesekera, 2003) models
RBAC policies using use cases and Horn clauses to represent security information and to check consistency. The limitation of the latter two approaches is that they each address only one control access scheme. An additional extension by Pavlich and Michel (2007) includes representation for role-based, discretionary, and mandatory access controls.

A more generic approach is taken by Espinonza et al. (2006) by integrating the UML profiles into a framework for annotating various qualitative or quantitative non-functional requirements. This approach adopts structural concepts and qualifiers from the Quality of Service and Fault Tolerance (QoS&FT) profile (OMG, 2003) along with its library style definitions of domain-specific NFPs, as well as key features from the SPT profile (OMG, 2002).

In the context of extensions to Architecture Description Languages (ADLs), the most notable extension is by Ren and Taylor (2005) proposing Secure xADL, an extension to xADL (Taylor et al., 2008) based on a unified access control model incorporating the classic model, the role-based model, and the trust management model. A related approach is OASIS’s (2002) Extensible Access Control Markup Language (XACML), providing a common language for the configuration of security policies within systems. Earlier, Moriconi et al. (1997) described an extension of SADL (Medvidovic and Taylor, 2000) to model security properties using the RAPIDE (Luckham et al., 1995)0 architecture description language.

In the realm of performance analysis, Balsamo et al. (1998) automatically derive a performance model, based on a Queuing Network Model, from a software architecture specification formally described in CHAM (Chemical Abstract Machine) (Inverardi and Wolf, 1995).

### 3.2.2. Aspect-oriented

The aspect-oriented (AO) class of approaches is centered on modularization techniques aimed at improving separation of concerns in the software system design and development (Kiczales et al., 1997). Aspect-Oriented Modeling (AOM) techniques allow software designers to conceptualize, describe, and communicate solutions for “cross-cutting concerns” (such as security, reliability, and new functional features). While many aspect oriented approaches have been developed to address dependability concerns from requirements modeling to design and implementation, our focus is on those approaches that are applicable in architecture and design time dependability modeling and prediction (France et al., 2004; Petriu et al., 2007; Woodside et al., 2009).

France et al. (2004) introduced an AOM approach to aid generation of logical, aspect-oriented architecture models (AAMs) consisting of a primary model (which is the base architecture model) and several aspect models (each of which describes the ways in which a dependability concern is addressed through the system’s architecture). Each AAM also determines the ways in which various aspect models can be woven with the primary model in order to provide a complete integrated view of the logical architecture of the system. A related approach by Petriu et al. (2007) uses France’s modeling framework, with the goal of transforming UML models of a given system to Layered Queuing Networks in order to subsequently analyze their performance. Note that this kind of transformation was previously used for systems designed without AOM (Gu and Petriu, 2002; Petriu and Woodside, 2004). Finally, Woodside et al. (2009) integrate France’s approach with the PUMA project (Woodside et al., 2005) and introduce an AO analysis framework that relates different security features to their impact on performance.

### 3.2.3. Hazard analysis

A specific class of software dependability analysis approaches focuses on identifying risks and offering solutions for risk control or mitigation (Leveson, 2005). The core of this class of techniques has been adapted from hardware systems. Some notable analysis techniques include Software Fault Tree Analysis (SFTA) (Leveson and Harvey, 1983) and Failure Modes and Effects Analysis (FMEA) (Stamatis, 1995). While SFTA techniques are used to investigate contributing causes to potential hazards in safety-critical applications, FMEA techniques offer analysis of potential failure modes within a software system for further classification by severity and likelihood of failure. Two classic safety analysis techniques are Hazard and Operability analysis (HAZOP) (Kletz, 1992) and Functional
Failure Analysis (FFA) (SAE, 1996). HAZOP provides a systematic way to identify all deviations from a design’s expected operation, as well as all the hazards related to these deviations. FFA creates an abstract functional model of the system, to be used for identifying possible combinations of different functional failures and assessing their effects and criticality.

Several techniques have been developed to address software systems’ safety analysis based on the software systems’ architecture and design. Fenelon et al. (1994) discuss adaptation of Fault Tree Analysis (FTA) and zonal hazard analysis to software and introduce Software Hazard Analysis and Resolution in Design (SHARD). SHARD is an analysis approach for high-level design based on HAZOP. In the context of SHARD, Fenelon et al. also introduce an integrated notation to overcome some of the limitations of FTA and FMEA. The Failure Transformation and Propagation Notation (FTPN) is a graphical notation that allows working both in top-down (FTA) and bottom-up (FMEA) mode. FTPN can be used as part of the safety analysis of a software artifact, as well as a way to specify failure behavior requirements (which output failure modes are acceptable, given an input failure mode).

In the context of modern aerospace systems, Mauri et al. (1998) propose a safety assessment approach called Failure Logic Analysis for System Hierarchies (FLASH). FLASH is a hybrid technique that combines SFTA, FMEA, HAZOP, and FFA to enable assessment of a hierarchical system. Papadopoulos and McDermid (1999) introduce Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) that is based both on FTPN and FLASH. Similar to FLASH, HiP-HOPS deals with the integration of well-established techniques such as FFA, FMEA and FTA. In order to simplify the analysis, HiP-HOPS offers a new algorithm for the development and synthesis of fault trees. Ultimately, it assumes that a consistent hierarchical model of the system exists and is able to guarantee the consistency of the results.

Hansen et al. (2004) introduce a systematic way to perform HAZOP on UML models and offer interpretations for UML-based software architecture descriptions in order to be able to apply HAZOP at the preliminary stages of a safety critical system design.

The Enhanced Safety Assessment for Complex Systems (ESACS) methodology (Bozzano et al., 2003) suggests two alternatives in failure modeling: (a) system model prototyping for safety, in which the system’s design is provided as an input and is enriched with failures and (b) failure mode injection, in which the system’s model is extended by injecting failure modes. The formal safety models obtained from either of these approaches can then be analyzed using either SFTA or FMEA.

Several approaches use hazard-based methodology to perform software architecture and design-based reliability analysis. Salvatore and Bechta (1993) suggest a reliability analysis method for critical applications by transforming a fault tree representation of the system failure modes to an equivalent Markov model. The recovery models of the system are separated from the fault tree before the transformation and are added automatically to the Markov chain after the transformation. Wu and Kelly (2005) propose an approach derived from the classical approaches (SFTA, FMEA) and use Communicating Sequential Processes (CSP) in order to model and analyze the failures of a system. The overall failure behavior of the system is a composite of the failure behavior of its components, each of which is modeled in terms of failure propagation and generation. This approach also incorporates architectural views into failure modeling and introduces failure model views as different perspectives on the system’s failure behavior.

3.2.4. Stochastic

In the stochastic-based class the operational and usage profiles are modeled as probabilistic processes such as Dynamic Bayesian Networks or Markov Chains. Markov chains are finite state machines which are integrated with transition probabilities that reflect the user’s operational profile. Cheung’s method (1980) is the basis for many stochastic approaches in early software dependability prediction, especially in reliability and availability. In his approach, Cheung provides a parameterized formula for the calculation of the system’s reliability, taking into consideration both the deterministic properties of the structure of the system as well as probabilistic properties for
the use and the failure of its components. Wang et al. (1999) extend Cheung’s reliability model by incorporating different architectural styles.

A milestone for stochastic dependability prediction was set by the theoretical framework introduced by Hamlet et al. (2001). Their framework introduces the separation of the component’s reliability from the user’s operational profile, along with later incorporation of the overall system-level operational profile into the system reliability calculations. It was the basis for many related component-based methods (Gohkale et al., 2004; Reussner et al., 2003). Gohkale et al. (2004) take an analytical approach that relies on data obtained from the actual execution of the application and extract parameters of architectural models. The performance and reliability models are based on DTMC, and the failure behavior of the components is characterized by time-dependent failure intensity.

Reussner et al. (2003) extend both (Cheung et al., 2008) and (Hamlet et al., 2001) by proposing a compositional approach to predict the reliability of the system using not only the users’ operational profiles but also the reliability of the interacting components. Consequently, the reliability of each component is a function of its usage profile, as well as the reliability of the services required for its operation. A major advantage of Reussner’s approach is that it can also be used for black-box components—unlike Hamlet’s framework, which requires the component’s source code. Furthermore, our past research (Cheung et al., 2008; Roshandel et al., 2007) extends the aforementioned models by introducing a reliability prediction model used in the very early stages of development, when knowledge related to the user’s operational profile is not available. Our approach covers both component-level reliability prediction (using Hidden Markov Models) and system-level reliability prediction (using Dynamic Bayesian Networks).

Goseva et al. (2003a) provide a risk assessment methodology that aids in the identification of potentially troublesome components that require more careful development and intensive testing. Each component and connector is assigned a dynamic risk factor for each usage scenario and a Markov model is constructed for each scenario to estimate the overall scenario risk factor.

Several approaches consider Stochastic Process Algebras (SPA) to provide reference models for software specification and performance, integrating the functional and non-functional aspects of a system. The most well-known SPAs are Time Processes and Performability evaluation (TIPP) (Hillston and Thomas, 1999), Extended Markovian Process Algebra (EMPA) (Bernardo, 2000), and Performance Evaluation Process Algebra (PEPA) (Hillston, 1993). The use of SPAs has two serious drawbacks. The first is that the solution of the Markov chain for the system’s performance prediction may not be possible due to state space explosion. The second is associated with the burden of requiring the software designer to provide specification using process algebras and to associate the various performance parameters with the appropriate actions. In response to the second problem, Pooley (1999) proposes a theoretical framework for the derivation of SPA models from UML diagrams. Also, Balsamo et al. (1998) introduce an SPA-based ADL called Emilia that enables the designer to create an SPA specification from the software architecture of a system.

The most commonly used modeling technique for early performance analysis of software systems is the derivation of Queuing Network Models (QNMs) from the system’s architecture specification and the subsequent analysis of the QNMs. A QNM is represented as a network of queues which is quantitatively evaluated through analytical or simulation methods in order to obtain a set of specific measures, called performance indices. For the efficient computation of the performance indices, various analysis (solution) methods for the QNMs have been defined over time (Kleinrock, 1976; Lazowska, 1994). However, in order to be able to perform early performance analysis based on QNMs, the research focus changes into the creation of relevant integrated models to represent the necessary performance related information at design time, rather than runtime. Therefore, as discussed in a previous section, most of the approaches (Cortellessa and Mirandola, 2000; Petriu and Woodside, 2004; Williams and Smith, 1998, 2002; Woodside et al. 2009) integrate UML diagrams with performance related attributes and subsequently use these models to predict the performance of the suggested architecture. Nevertheless, not all of the approaches in this category are based on UML extensions. For example, the approach proposed by Balsamo et al. (2002) automatically derives a performance model, based on a QNM, from a software architecture specification formally described in
Chemical Abstract Machine (CHAM) (Inverardi and Wolf, 1995). Their approach aids the performance comparison of two competing software architectures, even at a high level of abstraction.

### 3.3. Simulation-based

A group of dependability analysis approaches relies on simulation of architecture and design models to provide an estimation of software systems dependability attributes. We group approaches with simulation-based information sources into two classes: Language extensions are essentially UML-based approaches that extend or constrain the use of the UML semantics in order to generate more formal models capable of performing rigorous analysis. The Stochastic models are those approaches that take a probabilistic approach in assessing the dependability of the software systems based on their design and architectural models. Approaches in this section differ from those using specification-based information sources in that they rely on data obtained from runtime execution of the system.

#### 3.3.1. Language extensions

The methods included in this class are based on simulation models derived from information within extended UML models. Early simulation of the system can help predict whether specific system design will deliver the expected quality properties. Arief and Speirs (2000) introduce a simulation framework called Simulation Modeling Language (SimML) that aids the design of UML class and sequence diagrams, which are then used to automatically generate Java code mimicking the execution of the proposed system.

Miguel et al. (2000) propose an extension to UML using stereotypes, tagged values and stereotyped constrains, which enable expression of temporal requirements and resource usage of real-time systems. The approach uses the resulting UML diagrams to generate scheduling and simulation models, using the Analysis Model Generator (AMG) and the Simulation Model Generator (SMG) respectively. After the models have been analyzed and simulated, some of the results are presented in the original UML models, in order to help the interpretation of the simulation results.

#### 3.3.2. Stochastic

A group of probabilistic approaches rely on data about the system execution obtained at runtime to perform dependability analysis. Stochastic Reward Nets are used by Muppala et al. (1994) to generate and analyze Markov reward models, in which a reward rate is attached on each state of the Markov chain. This approach has been widely used for performance, reliability and availability prediction. Leangsuksun et al. (2003b) use stochastic reward nets to model the failure-repair behavior of the Open Source Cluster Application Resources (OSCAR), in order to study the availability rate of various configurations.

Another stochastic approach, presented by Sharma and Trivedi (2005), hierarchically models the software architecture using a Discrete Time Markov Chain with assigned suitable rewards for performance, reliability and security. Parameterized formulas are then used in order to predict the reliability, performance, and security of a given system using the aforementioned reward model.

Another group of simulation-based analysis techniques are based on the Monte Carlo simulation. Goseva and Kamavaram (2003) present an approach to perform uncertainty analysis on software reliability models in order to study the extent to which uncertainty associated with various parameters impacts the reliability estimation results. Uncertainty analysis is performed using the Method of Moments or the Monte Carlo simulation. The Method of Moments is used to quantify the uncertainty of the overall system reliability due to the uncertainty of its components’ reliability. The Monte Carlo simulation allows consideration of both the uncertainty of the various components’ reliabilities and the uncertainty of the operational profile.
4. Dependability Discussion

Our primary goal in developing the classification presented in this paper is capturing the state of the art in architecture-based approaches for dependability analysis with an emphasis on underlying formalisms. Existing surveys commonly focus on a single dependability attribute (e.g., Immonen and Niemela (2007) for reliability and Balsamo et al. (2004) for performance) or concentrate only on a specific class of modeling techniques (e.g., Ionita et al. (2002) for scenario-based approaches). While performing detailed analysis of a single dependability attribute is indispensable to the design and development of dependable software systems; understanding, modeling, and analyzing the tradeoffs among various dependability attributes is critical for an architect to objectively choose a specific architectural solution. Little research has been done in comprehensive architecture- and design-time dependability and tradeoff analysis. Among existing approaches (Bengtsson and Bosch, 1998; Clements et al., 2002; Duenas et al., 1998; Kazman et al., 1994, 1998; Sharma and Trivedi, 2005; Woodside et al., 2009), the majority are informal (with the exception of Sharma and Trivedi (2005) and Woodside et al. (2009)). Development of rigorous and comprehensive dependability analysis approaches that enable objective tradeoff analysis requires (1) studying quality analysis approaches based on their underlying formalisms, and (2) identifying those formalisms that may be compatible or complementary in nature. The classification framework presented in this paper takes a step in that direction as discussed below.

Table 1 highlights a view of the classification scheme that differs from the presentation given in the previous section. This perspective clearly depicts the modeling/analysis formalisms (rows) used to evaluate specific dependability attributes (columns). Furthermore, Table 1 can be used to identify underdeveloped research areas (blank spaces in the table), but also may be used to identify compatible dependability modeling/analysis methodologies that could form the basis for development of new tradeoff models.

For example, failure modeling and fault-tree analysis have previously been used to assess system reliability and safety (Leveson and Harvey, 1983). At the same time, Markov-based modeling and Reward Net analysis have been used to assess system reliability, security, and performance (Muppala et al., 1994). It is logical to envision a framework where reliability assessment is used as the basis for integrating these dependability analysis methodologies into a coherent tradeoff modeling and analysis framework that addresses reliability, safety, security, and performance analysis.

Similarly, Markov models have been analyzed using Process Algebra based approaches to assess system performance (Balsamo et al., 1998; Pooley, 1999). At the same time, uncertainty analysis of Markov models has been used to evaluate system reliability (Goseva and Kamavaram, 2003). It would be plausible to develop a Markov-based reliability-performance tradeoff analysis framework by relying on the common underlying modeling formalism (in this case Markov Models). Note that these tradeoff analysis approaches are in contrast to existing approaches which rely on the conceptual relationship between various dependability attributes (e.g., an architectural change that enhances system reliability could negatively impact its performance).

Finally, the table also illustrates that most existing dependability tradeoff analysis techniques rely on informal modeling techniques and the corresponding manual analyses (top portion of the table). While these approaches are helpful in providing a mapping between functional and non-functional requirements to high-level architectural design choices, they are limiting in their ability to go beyond qualitative tradeoff analysis. Pairing these informal techniques with more formal ones could help achieve both the breadth and depth necessary for comprehensive dependability analysis.

The novel classification of software architecture-based dependability analysis approaches, along with the discussion on dependability tradeoff analysis presented here identify possible new areas of software architecture and dependability research and a new vision for development of formal tradeoff analysis approaches. It is clear that much work remains in the area of formal modeling and analysis of software dependability. As part of our future work, we plan to extend this classification for the domain of software product families. Additionally, we plan to
develop a reliability-security-performance tradeoff analysis framework based on the insights obtained from this study. This framework will build upon our ongoing research in software reliability and software security.

<table>
<thead>
<tr>
<th>Modeling</th>
<th>Analysis</th>
<th>Reliability/ Availability</th>
<th>Safety</th>
<th>Security</th>
<th>Performance</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Maps</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDG</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abuse Cases</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFR-Frameworks</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal Trees</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML extensions</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADL extensions</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aspect-oriented Modeling</td>
<td>AO Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Joint Points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure Modeling</td>
<td>Fault-tree Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FMECA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threat Modeling</td>
<td>Semi-Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bayesian Models</td>
<td>Parameterized Formulas</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Markov Models</td>
<td>Risk analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parameterized formulas</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uncertainty analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process Algebras</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reward Nets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queuing Networks</td>
<td>Various</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Mapping between dependability attributes and modeling/analysis techniques

Acknowledgments

This work is supported by a grant from the National Science Foundation (award #0937472). We also wish to thank Zhen Yang for her constructive participation in the early discussions leading to this work.
References


Liu, L. and Yu, E., 2001. From requirements to architectural design using goals and scenarios, In *Proceedings of the First International Workshop from Software Requirements to Architectures* (STRAW'01), Toronto, Canada.


