***Modeling Software Architectures through Model Based Engineering***

Jeff Palmerino

***Abstract***

As the foundation to the development of software systems, modeling software architectures is critical to the quality of the final product. Existing research in this area has focused mainly around modeling approaches, static software architectures, and modeling analytics. However, as the complexity of software systems increase, so do the architectural models. Therefore, this paper provides an overview of existing research conducted on modeling software architectures, as well as a synthesis of the state of this research area, and possible future work studies.

***Introduction***

Software architecture is defined as the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution **[3].**In layman’s terms, software architecture is an abstraction of the highest-level structures of a software system, and the relationships among these structures. Therefore, modeling the architecture of any software design is an extremely important aspect of the any software’s development life-cycle.

Architecture models are an abstraction of the highest-level design that is used to test and assess proposed concepts prior to their implementation **[3]**. These models help bridge the gap between stakeholder requirements and detailed system design, meaning they help represent a common abstraction of a system that most stakeholders can use as a basis for mutual understanding, consensus, and communication between them and the development team. Formally, experienced architects, usually associated with teams, create their models from knowledge, experience, and usage of general software principles such as abstraction, refinement, coupling, and cohesion **[1].** From there, the model undergoes iterations of inspectionuntil the stakeholders are satisfied that the proposed architecture meets their requirements*.*

The final stakeholder-validated model is considered a formal specification of requirements and becomes the starting point for detailed design. However, just because a model might meet the stakeholder’s requirements, does not mean the software’s architecture has been modeled well. Roughly 80% of software system errors are found during implementation, resulting in exponentially growing rework and qualification costs **[4]**. This is usually due to mismatched assumptions between the system software components, the distributed hardware platform, the physical system, or misunderstandings between development team members **[4]**. On the other hand, these errors might not be found until implementation because insight into critical components of the system like performance, reliability, availability, security, and fault tolerance occurs late in the software’s development lifecycle **[4]**. However, advances in architecture the last 20 years have led to a process to address these problems through an approach called model-based engineering (MBE).

One of the main reasons MBE has grown in popularity is because of its ability to precisely capture a predictable system through repeated analysis early and throughout the software’s development life cycle **[3]**. This allows architects to see system-wide impacts of architectural choices, as well as reducing architectural risks and costs associated with integration problems, and a more simplified lifecycle of support is created.

The remainder of this paper is outlined in the following manner. First, an overview of existing research is provided. This is followed by an analysis of the current state of this research and lastly, possible areas for future work are discussed.

***Existing Research***

Much of the existing research concluded in support of modeling software architecture has focused around developing approaches that can enrich the activities of software architecture design. For example, Garcia, Astudillo, and Correal **[2]** addressed these issues associated with modeling software architecture by using Business Process Management Notation (BPMN) as a structured, consistent manner to keep the design processes as decision-making activities. By doing this, a mechanism is created to systematically formalize and manage architectural decisions, unlike the industrial approaches Krutchen’s “4+1”, Architecture Centric Design Method (ACDM), or Rozanski and Wood’s Working with Stakeholders Using Viewpoints and Perspectives (V&P) that identify and recognize the importance of architecture decisions, but don’t create ways to formalize and manage them.

On the other hand, some existing research has been focused around how to take existing modeling methods, and adapt them to meet the standards and complexity of today’s software systems. For example, the majority of architecture modeling methods that exist today are focused around *static* software architecture. Static, by definition, means the composition of components during a computation don’t change **[1].** However, many of the software architectures that exist today are *dynamic* in nature, meaning the composition of components during a computation do change. This type of dynamic architecture is seen primarily in cloud computing and was the primary focus around the research conducted by Ding **[1].** To combat this, Ding proposed a modeling method that uses a two-layer high level Petri nets called CPrT nets, which are extended with communication channels. These CPrT nets allow for static and dynamic architecture modeling through the use of a uniform formal notation, and an easy to use executable, providing developers with the ability to build complex models with relative ease.

Creating ways to formalize and manage architectural decisions is only as good as the quality of the model they produce. In order to test for model quality, tools are needed to identify common errors and omissions committed by the development team and architects. In **[3]**, Giammarco created 38 analytics that enable managers, developers, and architects operating in an MBE environment to test for model quality. The analytics enable quick and automatic identification of typical architectural model properties that impact model quality in order to correct them, and because of this, it enables time formerly spent on labor-intensive and error-prone activities of manually finding and fixing these issues to be reinvested into more advanced tasks.

***State of Research and Analysis***

As software systems become more and more complex every day, it is important that the ability to model their architectures appropriately increases at an equal or greater rate, especially when it comes to mission critical systems. For example, today’s vehicles are becoming autonomous at a rate quicker than many of the traditional software development methods can keep up with. Almost all of the computations done in autonomous vehicles are completed in real-time, and the composition of their architecture’s components change as necessary, meaning their architectures are dynamic as explained in **[1].** Due to the nature of this domain, any errors found in code-implementation that could have been avoided with better architecture modeling, would have vastly larger costs than most other domains. Therefore, analytical methods like the one provided in **[3]** combined with dynamic modeling methods in **[2]** could provide sustainable ways to appropriately model mission critical systems, like ones found in autonomous vehicles.

It may seem redundant to say that the software architecture design processes should be decision-making activities **[2]**, the redundancy in itself is one of the main reasons these decisions sometimes don’t get the necessary attention they should. Some of the decisions made in the design processes are subjective and uncertain, and remain intuitive and informal in nature. Their quality sometimes depends on the lead architect’s experience, and it can cause inconsistencies in the design process. Therefore, creating ways to formalize and manage those decisions like in **[2]**, not only allows for consistent decision making throughout the design process, but it also allows for appropriate decision making, i.e. nothing gets over looked, no assumptions are made, every decision is justified and explainable etc.

*Common Theme*

Although the existing research presented in this paper varied in different ways, as **[2]** focused on the decision-making activities of modeling software architecture, **[1]** focused on adapting current modeling methods, and **[3]** focused on analyzing the architectural models for correctness, there was a common theme; the value of modeling and the model based engineering process cannot be overlooked. Not only will software with quality architecture save time, and money, if errors are found in code-implementation stages, but the models associated with the architecture will be transferable. Often, many software systems will share underlying common features associated with their architecture. Therefore, having models that are transferable, i.e. reusable in other products, will help development teams to be more productive, reduce costs, and permit early exploration of alternatives.

***Conclusion***

Modeling software architecture is one of the most important processes of a software’s development lifecycle. Architecture plays a large role in early design decisions, and it drives the remaining software development, deployment, and maintenance. Therefore, using MBE to model it in a way that not only the stakeholders involved in the product can understand, but the members across the development team as well, is essential to the quality of the final product.

***References***

[1] J. Ding, "An Approach for Modeling and Analyzing Dynamic Software Architectures", 2016. Available: http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?arnumber=7603503&tag=1

[2] G. Garcia, H. Astudillo and D. Correal, "Modeling Software Architecture Process with a Decision-Making Approach", 2014. Available: http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?arnumber=7559662

[3] K. Giammarco, "Architecture Modeling Software Analytics: Model Quality and Maturity Assessment Using Automated Tools", 2017. Available: http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?arnumber=7994962

[4] "Model-Based Engineering", *Sei.cmu.edu*, 2017. Available: https://www.sei.cmu.edu/architecture/research/model-based-engineering/