Three Roles and Three Failure Patterns of Software Architects
featuring Bill Thomas reading a blog post by John Klein

Bill Thomas: Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

The text of this podcast originally appeared on the SEI Blog.

In today's podcast, we will highlight the blog post titled Three Roles and Three Failure Patterns of Software Architects by John Klein. In this post, Klein refers to an article he wrote for *IEEE Software*. A link to that article is available with the original blog post at insights.sei.cmu.edu. Click on the Authors tab, and find John Klein's name.

And now,

Three Roles and Three Failure Patterns of Software Architects
by John Klein
Senior Member of the Technical Staff
Architecture Practices Initiative

> When I was a chief architect working in industry, I was repeatedly asked the same questions. What makes an architect successful? What skills does a developer need to become a successful architect? There are no easy answers to these questions. For example, in my experience architects are most successful when their skills and capabilities match a project's specific needs.
>
> Too often, in answering the question of what skills make a successful architect, the focus is on skills such as communication and leadership. While these are important, an architect must have strong technical skills to design, model, and analyze the architecture.

As this post will explain, as a software system moves through its lifecycle, each phase calls for the architect to use a different mix of skills. This post also identifies three failure patterns that I have observed working with industry and government software projects.

What is a Software Architect?

Before exploring the three roles of successful software architects, it is important to start with an accepted definition of software architecture, which is this:

> *Architecture comprises the structures needed to reason about the system. Each structure, in turn, comprises elements, the relations among them, and the properties of the elements and relations.*

Software architects should certainly be interested in the ideas presented in this post, but I hope to reach a broader audience. Software program managers, who often inadvertently assume that all architects are alike, and therefore interchangeable, could benefit from a greater understanding of how to better match architects to projects. Also, this perspective could benefit project stakeholders who may need to step in and take certain steps if the architects are not well matched to a given project.

The Three Roles of the Software Architect

As I detailed in the IEEE Software column What Makes an Architect Successful?, there are three roles for the software architect, and these roles change based upon where the project is in the lifecycle of the system.

- **Initial designer**. During initial system design, a successful architect must be able to define architecturally significant functional and quality requirements, and then use the requirements to design abstractions that achieve conceptual integrity. Considered by Fred Brooks to be "the most important consideration in system design," conceptual integrity provides an internal consistency or internal logic that promotes uniformity in implementation and operation—the same things are done the same way throughout the system. This role is particularly important in the early stages of a software development project, when it is critical to help multiple teams efficiently collaborate on the implementation. There is a tradeoff here: conceptual integrity can introduce extra layers or generalize interfaces, and implementers may begin to erode the conceptual integrity through optimization and specialization. The architect must defend the conceptual integrity by demonstrating the approach's benefits.

  Beyond design, a successful architect will create models and analyze these models to ensure that the design meets the system's functional and quality requirements for performance, availability, usability, and other properties.

These models can range from simple box-and-line drawings or UML diagrams, to sophisticated performance models using AADL or availability models using TLA+ , which is a formal specification language used to design, model, document, and verify concurrent systems.

While architects in this phase may contribute to system implementation, their contributions typically focus on prototyping and pathfinding, or implementing widely used functions such as messaging or failure recovery. This coding helps the architect communicate how the architecture should be used, and it helps to keep the architect grounded in the realities of system development. This grounding enables the architect to adapt the design to what is achievable through the development team's tools, skills, and experience.

- **Extender**. After the initial release, there is usually a push to add value quickly. For example, the initial release of a commercial product comes after a significant up-front investment. In the system extension phase, there is often a push to quickly begin to recoup that investment. In many commercial systems, adding integrations with other systems, such as Facebook, Dropbox, or PayPal, can provide a big payback with relatively low development costs. In enterprise IT environments, integrating the new system with other enterprise systems can improve automation and add value quickly.

  Given this reality, it is important for a software architect to have an understanding of the as-built system and its interfaces. Software architects also must have an understanding of the technologies that are being used to integrate a system, such as particular middleware, application programming interfaces (APIs), or communication protocols. This understanding allows the architect to know which types of integrations can be added easily and quickly, and how to best integrate with a particular external system. The individual who is often the most successful in this phase was often a member of the initial development team and understands how the system was built, as well as undocumented capabilities and side effects.

  In this phase, architects must also be able to make tradeoffs during the design of the integrations. In the initial design phase, the emphasis was on creating and preserving conceptual integrity. As the system transitions to extension and ultimately sustainment, it is often necessary to tradeoff some conceptual integrity to add value. This scenario introduces one of the failure patterns that I have seen: when they move into the extension phase, architects responsible for the initial design refused to give up any of the conceptual integrity or allow the system to take on any technical debt. The integrations they designed were expensive and took too long to complete, and the system was too slow paying back the large initial investment.

This extension phase presents an opportunity for developers to step into the role of architect and get some experience in this role. The size of the changes to the system in this phase may be small, but the scope is still architectural-these decisions have a global impact on the functionality and quality of the system. The design changes warrant modeling and analysis, and although it may be necessary to take on some technical debt in this phase, this should be a conscious decision, backed up by an analysis of the tradeoffs.

- **Sustainer**. After the system has been in production for a substantial amount of time, the system often becomes expensive to maintain. The focus shifts to long-term sustainment, with the goal to continue delivering value with little or no changes to the architecture or implementation.

  For architects, the focus in this phase becomes analyzing, representing, and then communicating the continuing value of the system. Activities in this phase include modeling the technologies and patterns in the system using the new patterns that match current practices. Needed documentation skills in this phase involve an ability to explain to stakeholders the system's continuing relevancy and how it can continue to add value in this new world.

  In this phase, software architects need to focus on understanding the technology environment in which the system operates as well as the business context and mission context to understand and articulate the system's value without significant changes to the software.

Eventually, the sustainment phase concludes, a new system is commissioned, and these phases begin again. At this point, I have seen another failure pattern: Organizations sometimes mistakenly assume that an architect that has been sustaining the old system would be the best choice to design the new one. During the sustainment phase, however, a focus on the legacy environment can lead to the architect losing touch with current development practices and technologies. Consequently, this architect may not be well positioned for the challenges of the initial design phase. I have seen smart organizations recognize the value of the knowledge and experience of sustainment architects, and the organization has found ways to involve them in building the new system without making them responsible for it.

Three Failure Patterns

In examining the three roles of the software architect, I also identified failure patterns. As detailed in my recent IEEE Software column, failure patterns result from the mismatch of the architect's skills and the role's needs at a particular time. While two of the patterns

have been touched upon above, let me briefly reintroduce them as well as the third pattern here:

- **Wraparound**. This pattern describes a scenario where an architect was a successful sustainer for a legacy system and then is selected as the de facto initial designer for the replacement system. Unfortunately, in the sustainer role, the architect may not maintain skills and knowledge in current technologies and development practices and tools. As a result, when designing the new system, the architect is out-of-sync with the development team's skills and practices. The result is that the replacement system is late and over budget.

- **Rising star**. This pattern describes a scenario where a developer steps into the architect role during the integration phase and successfully delivers profitable integrations. On the basis of that success, the developer-turned-architect is tapped to initially design a new system's architecture. The only problem is that in this new role the architect has neither the appropriate design and analysis training nor adequate experience designing at the system level. While the resulting system architecture matches the development team's skills and processes, it does not satisfy key functional and quality requirements.

- **Overprotective parent**. In this pattern, the architect who served as the initial system designer remains responsible as the system moves into the integration phase. In the integration phase, however, a reluctance to dilute the architecture's conceptual integrity often results in a system that can't deliver new value fast enough and falls behind competitors.

Wrapping Up and Looking Ahead

Architects don't work in a vacuum. They work with organizations. While there are skills that an architect must have to be successful, there are also skills that a team working on a software project needs to achieve success. That individual architect and team also work at an organization that will need certain capabilities to use architecture as a strategic tool. While the individual architect is an important component, it is but one piece of the project.

Different system lifecycle phases require different skills from a software architect. Rare is the architect who can seamlessly transition through all three phases, and software architects, developers, and program managers must be aware of these limitations moving forward.

.

Thank you for joining us today. This blog post is available at insights.sei.cmu.edu. Click on the Authors tab and find the name Klein, K-L-E-I-N.

The blog post featured in this podcast is based on the column *What Makes an Architect Successful,* that John Klein wrote for the January/February 2016 edition *IEEE Software.*

John Klein also co-presented a webinar titled *What Makes a Successful Architect?* with Ipek Ozkaya, Michael Keeling, and Andrew Kotov.

Links to these resources are available in our transcript.

This podcast is available on the SEI website at sei.cmu.edu/podcasts and on Carnegie Mellon University's iTunesU site. As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you.