Editor: **Eoin Woods**
Endava
eoin.woods@endava.com

What Makes an Architect Successful?

John Klein

WHAT MAKES a software architect successful? As with many such questions, the answer is, "it depends." Success is a function of skills and context. So, an architect whose skills and capabilities match the project's needs will more likely be successful. Moreover, each software life-cycle phase requires different skills. Here, I summarize a model that identifies the skills needed at each phase[1] and use this model to explain three recently observed failure patterns.

This article has three target audiences. The first is software developers wanting to advance to an architect role, for whom I'll identify possible pathways. The second is managers who assign architects to projects. My message to them is that every project is different and that architects aren't interchangeable. Finally, I aim to help developers decide whether their system's architect is right for the project and, if not, what they might do to ensure the project's success.

What's a Software Architect?

The title "software architect" might be given to the most-senior developer or to the trusted technical advisor to managers. Or it might actually refer to the team member who creates and evolves the system's architecture. In practice, many definitions of software architecture exist,[2] each leading to different definition of "architect." This diversity was reflected in a survey done by Paul Clements and his colleagues, which identified approximately 200 duties, 100 skills, and 100 knowledge areas for practicing software architects.[3]

For simplicity's sake, I'll use an accepted definition of software architecture:

> *Architecture is the structures needed to reason about the system. Each structure comprises elements, the relations among them, and the properties of the elements and relations.*[4]

An architecture is a bridge between the system's business goals and its realization. Architects' reasoning focuses on assessing the likelihood that the implemented system will achieve the functional and quality requirements to satisfy the system's goals.

Roles

Over a system's lifetime, our model has three distinct roles for software architects: initial designer, extender, and sustainer (see Figure 1).

Initial Designer

Architects first create the initial system design. This requires them to identify the architecturally significant functional and quality requirements and to use a method such as Attribute-Driven Design[5] to create the architecture structures.

A good architecture exhibits conceptual integrity, defining the key abstractions that enable consistent approaches to design and analysis. This integrity often comes at a cost—for example, by introducing extra layers or generalizing interfaces. Successful architects must prevent erosion of their design's

0740-7459/16/$33.00 © 2016 IEEE

conceptual integrity during initial implementation by demonstrating the approach's benefits.

Designing an architecture isn't enough; architects must also consider how the architecture will be implemented. This requires adapting the design to the development team's skills and experience, creating a design that can be developed incrementally, and using architecture approaches that allow continuous integration and deployment.

In this first role, good architects might also contribute to the system implementation. However, their contributions typically involve prototyping and pathfinding, or implementing cross-cutting functions such as messaging or failure recovery.

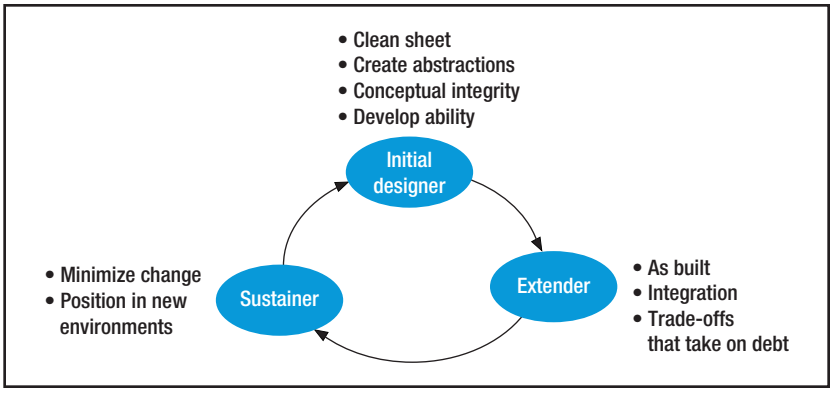

FIGURE 1. Software architects' three roles over a system's lifetime.

Extender

This role begins after the system is delivered or deployed in production. Often, developers can rapidly increase the value the system delivers by extending it or integrating it with other systems. For example, by integrating business systems, they can automate business processes and reduce costs. By integrating consumer products, they can create solutions that address new markets.

Here, architects need a good understanding of the as-built system to identify integration opportunities and approaches. For example, they must know whether an API can be extended easily or which workflow event should trigger a new action. So, they must be hands-on in the system's code.

Architects often make tradeoffs to accomplish integrations. Unless the initial design anticipated the integration, these tradeoffs can reduce the architecture's conceptual integrity by making an abstraction a little less general or by introducing a special case into a previously single-path execution flow. Whereas initial designers prioritize conceptual integrity, extenders must be willing to sacrifice this integrity and take on technical or functional debt to accelerate value creation.

Sustainer

This role comes into play after the system has been in production for a while. At this point, the system still delivers value but is becoming more expensive to maintain and evolve. The priority here is to avoid change, so the architects' job is to demonstrate the system's continuing relevancy and appropriateness as the environment changes.

The architects might create new representations (using new viewpoints, in the terminology of the ISO/IEC/IEEE 42010 architecture description) of the system's structures—without changing the system's implementation—to communicate and support reasoning about the system in new contexts. So, they must understand the business value the system creates, the technology the system implements, and the evolving technology context in which the system operates. Then, they must reconcile these with little or no change to the system's implementation.

Comparing the Roles

Clearly, the skills required of the architect during each phase differ. Figure 2 highlights these differences.

Initial designers' success hinges on designing "in the large." This involves creating abstractions that the developers can use and that capture the essential characteristics of the system's function (so that the abstractions will endure as the system evolves). It also involves analyzing the entire architecture to demonstrate that it's sufficient to create the desired system. Architects usually acquire these skills through a combination of practical experience (often including an apprenticeship under an experienced architect) and formal training.

In contrast, extenders' success usually requires a deep understanding of the as-built system and the technologies used by the system and by those systems that must be integrated with it, such as databases, middleware, and frameworks. In many organizations, the initial designer is reassigned after delivering the system's first versions, and one or more developers take over as extenders. Success here

Initial designer	Extender	Sustainer
• Create and defend conceptual integrity	• Compromise conceptual integrity to accelerate value creation	• Assess and communicate conceptual integrity in new environment
• Understanding of all system structures and abstractions	• Understanding of as-built system, including undocumented side effects	• Understanding of system's environment and of competitive systems
• Proficiency with architecture design patterns	• Proficiency with architecture integration patterns	• Knowledge of how the system creates value for users or customers

FIGURE 2. Comparison of software architects' required skills across the three roles.

requires careful compromises that enable integration with other systems but that don't make key system functions and qualities fragile and expensive to evolve and maintain. Analysis skills are less important than during initial design. This is because the limited system changes reduce the required analysis's scope and because prototyping and demonstration often replace predictive analysis.

For sustainers, the priority is communication. They select appropriate viewpoints to explain the system in the evolving context and use those viewpoints to analyze the system and demonstrate that it can still deliver value. They don't exercise their design skills and often focus on mature or outdated technologies.

In three failure patterns I've observed, the architect's skills didn't match the role's needs.

The first pattern is the "wraparound." An architect who was a successful sustainer for a legacy system becomes the replacement system's initial designer. This practice is common in many organizations because the sustainer "knows the system we're replacing." However, while in the sustainer role, the architect's design skills got rusty, and he lost touch with development practices. During the new system's initial design, he struggles to analyze and create the new architecture, designing one that doesn't fit the development team's skills and processes. So, the replacement system is late and over budget.

The second pattern is the "rising star." A developer steps into the architect role during the integration phase and successfully delivers profitable integrations. On the basis of this success, she's assigned to initially design a new system's architecture. She has had little formal design and analysis training and no experience designing at the system level. However, her development experience helps, and she produces an architecture that fits the development team's skills and processes. Nevertheless, it doesn't satisfy key functional and quality requirements.

The final pattern is the "overprotective parent." The architect who was the initial designer remains as the system transitions to the integration phase. However, he can't compromise his architecture's conceptual integrity, so his approaches for extending the system and integrating it with other systems involve large-scale, expensive changes. The system can't deliver new value rapidly and falls behind more nimble competitors.

The model in Figure 1 helps explain why an architect who was successful in one role might be unsuccessful in another. Different system life-cycle phases call for different skills, and it's a rare individual who can seamlessly make the transition.

Acknowledgments

This article is based on research funded and supported by the US Department of Defense under contract FA8721-05-C-0003 with Carnegie Mellon University for operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0002893.

References

1. J. Klein, "How Does the Architect's Role Change as the Software Ages?," *Proc. 5th Working IEEE/IFIP Conf. Software Architecture* (WICSA 05), 2005, p. 141.
2. "Modern Software Architecture Definitions," Software Eng. Inst., Carnegie Mellon Univ., 2015; www.sei.cmu.edu/architecture/start/glossary/moderndefs.cfm.
3. P. Clements et al., "The Duties, Skills, and Knowledge of Software Architects," *Proc. 6th Working IEEE/IFIP Conf. Software Architecture* (WICSA 07), 2007, p. 20.
4. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed., Addison-Wesley, 2013.
5. "Attribute-Driven Design Method," Software Eng. Inst., Carnegie Mellon Univ., 2015; www.sei.cmu.edu/architecture/tools/define/add.cfm.

JOHN KLEIN is a Senior Member of the Technical Staff at the Software Engineering Institute at Carnegie Mellon University. Contact him at jklein@sei.cmu.edu.