The Five Properties of Successful Architectural Oversight

Nick Rozanski

ARCHITECTURAL LEADERSHIP is probably the most important part of a software architect’s role. A key aspect of architectural leadership is architectural oversight, which helps ensure that development teams design and build their systems in the right way for their stakeholders—the people who have an interest in the system’s success. Unfortunately, “the right way” means different things to different people.

Obviously, a system’s architectural components must meet its functional requirements. For example, a sales-ordering system will probably need order entry screens, an order manager, and an order database. These components will need to be wired correctly so that the system can perform correctly. New orders will need to flow from the order entry screens to the order database via the order manager.

However, the system will also need to exhibit the right quality properties, such as scalability, resilience, and security (often called nonfunctional requirements). Orders should be accepted only from authenticated customers, and the system should be able to scale to adapt to the volume at busy times. It’s necessary—and often more difficult—to put as much thought into these architecture aspects as you would into functional and information architecture.

Architectural oversight is about keeping a watchful eye on projects as they move from concept and architecture to building, testing, and deployment, ensuring adherence to the original vision and architecture. (And if not, ensuring there are good reasons for this and that architectural changes are properly evaluated, communicated, and agreed upon.) Overseeing in a way that ensures that the system has the right architecture can easily become overly bureaucratic or poorly leveled (for example, focusing on some characteristics in detail while ignoring others). Worst of all, this process can fail to ensure that the delivered systems are sound and fit for the purpose.

Effective architectural oversight has five important properties: it must be timely, objective, systematic, constructive, and, most important, pragmatic (see Figure 1). Here, I examine each property to understand what it means, why it’s important, how you can achieve it, and what might happen if you don’t.

For this article’s purposes, I’ll assume you’re an architect with several systems in your portfolio, including new builds or systems undergoing substantial changes. You might be the architect of a collection of systems that perform a particular business function, or of the systems in one location. However, if your role is different—for example, you’re a...
solution architect looking after a single system or an enterprise architect responsible for the whole systems landscape—much of what I describe here will still apply to you.

**Timely**
Architectural oversight must occur at the right times. Get involved in development projects early, when the architecture is still being formulated and important decisions are being made. Such decisions include

- what the system’s fundamental structure will be,
- how the system will manage data and use other systems’ data,
- how it will scale and be resilient and secure, and
- how it will use advanced or unfamiliar technologies or approaches.

These decisions are difficult to overturn or change later during the project. For example, if your team starts out building a monolithic system, turning it into a collection of loosely coupled, semi-independent components will be a lot of work.

If you’re using waterfall or iterative development, expect to scrutinize the architecture at the start of high-level design. There are strong arguments for you to be involved even earlier, while the architecture is still being formulated. This lets you give your team the benefit of your experience and knowledge of what works (and what doesn’t).

If you’re using agile methods, your interaction model will be a little more fluid. Most agile methods include some architecture envisioning early on, although this might be less precise than it would be for waterfall or iterative development. The architecture will likely evolve and crystallize during subsequent agile sprints or iterations. You’ll need ongoing oversight to ensure that the architecture is moving in a sensible direction and that important aspects aren’t being ignored.

Architectural oversight performed at the wrong time (for instance, after the big architectural decisions have already been made and acted on) has little chance of improving the system’s architecture.

**Objective**
Architectural oversight must be based on clearly stated principles, guidelines, and patterns, rather than on a subjective opinion. We all have our own ideas about how best to build software systems. An old joke says that if you put two architects in a room, they’ll come up with at least three ways to design a system. Although there are common architectural principles and patterns that we all work toward, there’s more than one “good” way to build a system. Although there are common architectural principles and patterns that we all work toward, there’s more than one “good” way to build a system.

The goal of objectivity is to come to the same overall conclusions about the architecture regardless of who does the actual oversight and, specifically, how that person would have designed the architecture. This helps ensure a fair, objective process. A good way to achieve objectivity is to have a set of written guidelines, standards, and patterns that project teams can follow (or are required to follow). These guidelines should quickly lead team members to the right decisions, while leaving room for innovation and creativity.

Subjective architectural oversight can lead to inconsistent, arbitrary guidance, which is confusing for development teams, leads to resentment, and will likely be ignored (or worse, blindly followed at a disproportionate cost).

**Systematic**
Architectural oversight must follow well-defined, repeatable processes whose objectives and outcomes are clearly understood. Systematic oversight, whether lightweight or thorough, will follow a defined sequence of steps with known inputs and outputs to achieve a well-defined goal in a finite (and hopefully relatively brief) time period. The participants are known, and everyone understands their role and responsibilities. The tasks are clearly defined, and all participants clearly understand the overall objective.

Ensure your outputs are recorded in enough detail so that they are understood, can be acted on, and are clearly communicated to your stakeholders. Itemize actions and close them when complete, and record decisions along with their rationale (why the decision was made) and implications (what must happen next). Clearly document exceptions involving noncompliance with policy, standards, or general good practice, along with the reasons for the exceptions.

If your oversight isn’t systematic,
THE PRAGMATIC ARCHITECT

your stakeholders won’t clearly understand who’s involved, why the oversight is being done, when it was started or finished, and its outcomes.

Constructive
Architectural oversight must lead to better architecture. Constructive oversight results in real, significant, and valuable change. In practice, this often means that the resulting architecture is more scalable, resilient, highly available, and secure. These qualities often receive less attention than they deserve when developers are ensuring that a system meets its functional requirements. However, a system that is unreliable, is too slow, or has security holes won’t be viewed as a success and might even be abandoned.

The most constructive oversight leads to specific architectural changes or improvements. For instance,

• functional components might be added,
• the system might be better designed for resilience or high availability,
• the system might be given scale-out or scale-up features so that it can continue to perform under high loads, or
• security features might be improved to better control access to sensitive functions or data.

If architectural oversight doesn’t lead to specific improvements, your stakeholders will view it as a waste of time or a bureaucratic form-filling exercise they must endure, rather than an important part of software development.

Pragmatic
Architectural oversight must take into account real-world constraints such as time, cost, and the availability of skills, without diluting the oversight’s purpose or effectiveness. You’ll most often have to deal with time and cost constraints. There’s no point in insisting on an architectural capability or feature if implementing it would substantially overrun or blow the project’s budget. You must be ready to compromise while ensuring the architecture’s overall integrity.

Start by ensuring that you and your stakeholders have a common understanding of the business importance of the system you’re looking at. This will help you pitch your oversight at the right level. You can then assess the benefits and risks to help you come to a decision.

For example, a proposed business-critical system with no disaster recovery capabilities would be a high risk because a significant incident could shut down the business. This aspect should be remediated before the system goes into production. On the other hand, it’s probably reasonable to not implement such a feature for a proof-of-concept system that will be rewritten anyway if it’s successful.

In any case, don’t expect everyone to understand and agree on all the details of a system’s architecture, especially at the early stages of a project. If some architecture aspect is still being developed, focus on tracking this to a decision rather than labeling the system as “non-compliant” in some way.

Don’t aim for architectural perfection or demand unnecessary architectural capabilities. Be prepared to compromise when it’s right to do so, but ensure that these compromises are understood, agreed on, documented, and communicated to all stakeholders.

NICK ROZANSKI is a lead architect in a major UK bank’s Chief Technology Office. Contact him at nick@rozanski.org.uk.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.