



Architecting in the Gaps

A Metaphor for Architecture Work

Eoin Woods

EVER SINCE Dewayne Perry and Alexander Wolf suggested that software architecture comprises elements, form, and rationale,¹ people have been asking, “What is architecture, and what is design?” I’ve been asked this countless times and often find it difficult to respond with a clear answer.

architects must balance many competing concerns.

- *System-wide focus.* Many of the concerns architects address are system qualities (nonfunctional requirements), and so need to be considered across the system as a whole rather than at the individual component level.

Architecting in the Gaps

As I thought about this, a metaphor for software architecture occurred to me: “architecting in the gaps.” Architecture organizes, links, unifies, and constrains a system’s detailed design work and so is inherently about the system elements’ boundaries rather than their inner workings. A lot of architecture work is concerned with the system’s quality properties, which usually requires a focus on element boundaries and the interfaces and interactions that connect elements—bridging the gaps between those elements.

So, what sort of boundaries are we concerned with? I can think of at least four types.

Architects are often the only people responsible for bridging the gaps between a system’s elements.

When I recognize good architecture work, it normally exhibits these characteristics:

- *Design-centric.* It’s often said that “all architecture is design, but not all design is architecture.”² Architecture is fundamentally about making design decisions.
- *Balancing concerns.* Architecture work usually involves satisfying the needs of a varied community of stakeholders, so

- *Leadership.* The architect’s work involves making decisions, so they must possess strong leadership skills.

However, even when I was able to use these characteristics to identify architecture work, I still had difficulty clearly explaining the architect’s job and how it fits into the overall project. I realized I needed a different way to explain it—I needed a good metaphor for architecture work.

- Architecture can be realized as a set of technical boundaries, where the architecture is embodied in a run-time structure. For example, consider a system built on top of middleware products, where the system’s architecture is defined largely by the middleware’s configuration and deployment.
- Architecture can reflect organizational boundaries, such as when a domain architect is responsible for the systems in a

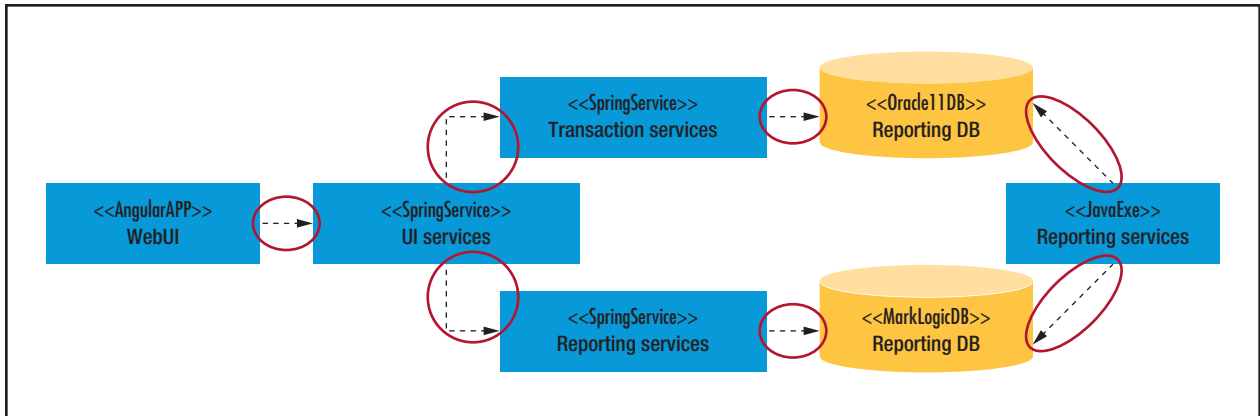


FIGURE 1. System boundaries. The red circles indicate typical areas that would be worth searching for architectural concerns—they tend to be around the interfaces and interactions of a system’s elements.

business area and negotiates the interfaces between that domain and others.

- Architecture can define conceptual boundaries, such as those in a domain’s reference architecture, which helps people understand the domain’s essential scope, structure, and relationships.
- Architecture is often found at software design boundaries, describing the components and connectors that make up a system’s fundamental structure.

Using the Metaphor

How can we use this metaphor in our day-to-day lives as software architects? Here are some common situations in which I’ve found it can help.

Justifying Architecture Work

Although the term “software architecture” is widely used, the question of its value still often arises. This is particularly true in the context of agile delivery, which often focuses on providing functions to product owners.

The metaphor can help high-

light the contribution of architecture work and why it matters. Architects are often the only people responsible for bridging the gaps between a system’s elements. Even when software developers are organized into feature teams and work across components, they’ll still be concerned with the internal details of each part of the system and naturally focus on what’s needed to support the features they are working on. In contrast, architecture work helps maintain the integrity of the overall system, which requires a very different focus from that required of a feature developer.

How Much Is Enough?

Knowing when enough architecture work has been completed is also difficult. How much architecture work do we need to do? When does it happen? How do we know when we’re done? The metaphor can help here because if we’ve considered all the “gaps” in a system and resolved the problems we’ve found, we’ve probably mitigated the architectural risks, in which case we’ve done enough architecture work.

Maintaining Focus

There are always many areas in which software architecture techniques might be useful, but there is rarely time to address them all. This makes it difficult for architects to know where to focus their efforts. The metaphor can help architects concentrate on the most important aspects of their work. There’s always a temptation to get involved in everything. However, if architects focus on the system’s structure rather than on the details of each element’s implementation, they’ll be able to effectively influence the system’s quality properties—a key responsibility of architects.

Figure 1 shows an example of this. The red circles indicate typical areas that would be worth searching for architectural concerns—they tend to be around the interfaces and interactions of a system’s elements.

Collaborating Effectively

Finally, it can be hard to know how to work effectively with other people. In some cases, a number of people in development teams do all the architecture work; in others, a software

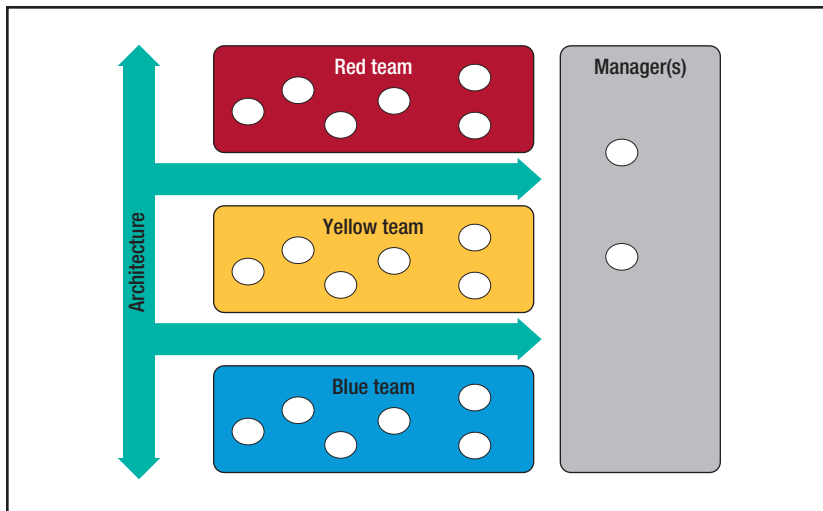


FIGURE 2. Boundaries guiding an architect's collaboration with other project team members. The architect concentrates on system-wide concerns and interactions between the teams, rather than the work inside the teams.

architect works alongside (rather than in) the development teams. In either case, how does the architecture work get integrated with other work on the project? How does an individual architect work with other people on the project?


Here, the metaphor can be a guide to effective collaboration. When architects work with development teams, a lack of clarity as to where responsibilities start and stop can lead to confusion or even conflict. It can help if team members visualize the architect's focus as being on the gaps between the system's elements rather than on their internal implementations. An architect generally defers to team leaders when a disagreement or question relates to a system element's internal workings. The team leaders generally defer to an architect when a question is about element boundaries and interactions. Figure 2 suggests how architecture fits into a project's organizational structure to allow for effective collaboration.

Taking it Too Far

Like any metaphor, mine can be taken too far to where it becomes misleading and unhelpful. Although most software architecture work is found at the boundaries of elements, architects can't always stop there. They might need to delve into details in the system to understand an element and its impact on the system as a whole.

Another danger of overusing the metaphor is the possibility of ending up without ownership of anything tangible. If architects focus on only the gaps, they might end up doing a lot of valuable coordination work but with the appearance of not having contributed anything specific. To avoid this, architects must be clear about which practical aspects of the project they own, such as the design of particular system qualities.

Although I've found "architecting in the gaps" to be very useful for visualizing software architecture work, it should be used like any metaphor—sparingly!

A metaphor can be a powerful aid to learning and communication, and I've found that the idea of "architecting in the gaps" helps people visualize where architecture work fits in the software development process. So the next time you're struggling to find an architectural focus or the next time someone asks you why they need an architect, just remember: the answer is in the gaps. 

References

1. D.E. Perry and A.L. Wolf, "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Eng. Notes*, vol. 17, no. 4, 1992, pp. 40–52.
2. G. Booch, "Software Architecture, Software Engineering, and Renaissance Jazz," blog, 2 Mar. 2006; www.ibm.com/developerworks/community/blogs/gradybooch/entry/on_design.

EOIN WOODS is the chief technology officer at Endava, a European IT services company. Contact him at eoin.woods@endava.com.



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

Software

VISIT US ONLINE

computer.org
/software