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

Should Architects Code?

Eoin Woods

A FEW SOFTWARE architecture questions always light up the Twittersphere with controversy when asked:

- What is architecture, and is it just design?
- Do you need architecture in agile development?
- Should architects code?

I examined the first two questions in previous columns[1,2] but have avoided answering the third. I address it now, however, because it's an intriguing question, which doesn't have an obvious answer.

When people ask, "Should architects code?" or talk about "the coding architect," they might be referring to anything from an architect or designer keeping a working knowledge of the technology in use (and being able to review and write code if necessary) to an architect spending significant time writing a system's production code.

Let's assume that the question is simply whether the people performing the system's architecture work should also develop some of the system's production code.

Personal Motivations

Often, architecture work naturally diverts architects from spending large amounts of time developing a system's production code. Architecture is a technical management activity that involves a range of work, not just coding. So what are some personal reasons for architects to continue coding work?

First, to lead a technical team, architects must build and maintain technical credibility so that other team members respect their opinions. Displaying strong coding skills can help build these technical credentials.

Second, architects should continue coding to maintain and improve their development skills—to not only achieve personal satisfaction but also set high yet realistic standards for others. Nothing clarifies expectations about "quality" and "craftsmanship" better than a well-written example.

Finally, many individuals became software engineers because they like developing software. Coding can increase motivation while keeping skills current.

What Are the Benefits?

There are potential benefits to architects performing code development for their systems.

First, coding work offers a useful reality check about the the experience of working as a developer on the system. Are the technologies easy for developers to use? Is the build-and-release pipeline working effectively? Are there any serious impediments to developer effectiveness? By working as a developer, architects can get a good perspective on such questions.

Performing implementation work also lets architects see their architecture's realization. This helps them more deeply understand their architectural decisions' implications and spot possible problems and those inevitable places where the implementation strays from the plan.

Development work also helps architects stay current with their system's technologies. Over time, technologies

 0740-7459/17/$33.00 © 2017 IEEE

get replaced or evolve. When architects stop coding regularly, they can lose sight of these important details.

What Are the Drawbacks?

Coding while performing architecture work poses some difficulties as well.

First, the architects' priorities become muddled—whereas their architecture work serves to make the team more effective, their development work reflects more personal objectives. They must think about coding time in terms of its return on investment (ROI). The first few hours a week will likely yield a high ROI, but how about the 20th hour? By then, there are almost certainly other tasks architects should be doing to make the team more effective.

A project's scale will affect the ROI estimation. The larger the team, the larger the delayed architecture work's impact. This is why I've reluctantly shrunk my coding time to almost zero on some projects—too many other high-priority issues required my attention.

Second, development and architecture work differ fundamentally. Development work demands significant periods of focused attention. Interruptions make developers less effective. In contrast, much architecture work involves reacting to questions or concerns and identifying and responding to risks or problems. It's difficult to work in both ways at once.

Combined, these factors create the risk that a coding architect will block the project's critical path. An architectural decision might not be made quickly because the architect is racing to finish a critical module. Or, an important feature might not be delivered because the architect was constantly interrupted while trying to finish an important part of the code.

A final factor that we architects might not want to admit is that, perhaps, we aren't as effective at coding as we used to be. Both technology and our individual skills change over time. If we're not 100 percent focused on development tasks, are we truly still as productive as we once were?

How Can Architects Stay Involved?

By keeping their development work off the critical path, architects can mitigate problems caused by conflicting or changing priorities. To remain closely involved in their system's implementation while avoiding the problems I've discussed, architects can do the following:

- *Fix bugs*. Fixing defects can be instructive and directly valuable to the project. It provides insight into the developer experience and the strengths and weaknesses of the architecture and code.
- *Refactor*. Technical debt nearly always accumulates, so architects might tackle it in small, safe steps. They'll quickly uncover any weaknesses in the architecture, implementation consistency, or tests.
- *Investigate problems*. Architects can get involved in debugging and problem investigation. Whether it's a performance problem, poor scalability, or a subtle intermittent error, they can offer a valuable perspective while learning about the qualities their architecture provides.
- *Test*. Architects might well find that testing isn't as thorough or sophisticated as they'd like. So, another opportunity for involvement is to improve automated tests. This will let architects hone their coding skills while developing a shared understanding of how to test systems.
- *Create architectural spikes*. Perhaps the most obvious task to choose is carrying out the proof-of-concept exercises that support architectural decision making. Doing so can deepen architects' knowledge of their decisions' implications.

Architects should pair with developers whenever possible on these tasks. Not only can they share expertise, but the architects can also learn from those closer to the state of the art.

It's still important to keep an eye on the schedule, even for tasks off the critical path. If architects notice they're running out of time or are about to be distracted by another priority, they must quickly reassign development tasks—no project manager likes a surprise that has become difficult to mitigate.

So, should architects code? My experience is that there's generally a positive ROI when architects do carefully selected implementation work, whether it's testing, refactoring, architectural spikes, or simply some part of the system where they're the best person for the job. Provided the project's scale allows it, doing some coding helps to root architecture work, keep architects' technology knowledge up to date, and sometimes save their sanity!

References

1. E. Woods, "Return of the Pragmatic Architect," *IEEE Software*, vol. 31, no. 3, 2014, pp. 10–13.
2. E. Woods, "Aligning Architecture Work with Agile Teams," *IEEE Software*, vol. 32, no. 5, 2015, pp. 24–26.