The Role of an Application Architect on an Agile Project

ccpace
Committed Partner. Creating Results.

The news that an organization is adopting agile application delivery often prompts team members to ask, "How will my role change? Will it even exist anymore?" This is especially true of architects. With all the talk of "emergent architecture" and "self-managing developer teams" what role does the architect play? Is there a place for an architect at all?

(The industry typically distinguishes between an enterprise architect and an application architect. This particular whitepaper discusses only application architects; a separate paper will discuss the enterprise architect.)

If you're an architect, rest assured: you most definitely still have an important role to play on agile projects. But your role may change.

Simply stated, the architect's role on an agile project is exactly the same role that everyone else on an agile project has: to enable the delivery of value to the customer. Does this sound too glib? Perhaps, but never lose sight of it. Your role isn't to oppose fast delivery, or to act as a check and balance against the business team, or to act as a disciplinarian to the development team. You may need to be all those things at various times on the project, but your overall objective should be to enable the delivery of value to the customer, period. Those other things may be means to that end, but only the means. Keep sight of the end goal and be flexible on the means to achieve that end.

OK, so how can you best enable value delivery to the customer? By making sure that your architectural vision, guidelines, mandates, and best practices actually make it into the delivered software product. Too often, architects communicate these to development teams in meetings or with documents. The development team reads and absorbs them with the best of intentions, but as pressure to deliver mounts and time grows short – or maybe because the team lacks the experience and knowledge to understand the vision – the team starts to take shortcuts and ignore the architect's advice. Doing the occasional code review or dropping by the team once in a while with stern reminders to follow architectural guidelines won't make any difference. That just adds to their workload and sets up an antagonistic relationship between architect and developers. At CC Pace, we call this drive-by architecture, because once in a while, the architect drives by the developer team, rolls down a window, fires architectural requirements at them, and then vanishes, leaving the developers feeling even more stressed out and uncertain how to proceed.

Don't be a drive-by architect.

Communicate your vision for the application architecture and best practices early in the project. You can do this as part of the kickoff meeting, or if the product owner team doesn't want to sit through a lot of technical presentations, hold a separate technical kickoff meeting. Always explain the reasoning behind your vision – what are the drivers? Invite discussion and questions. Be clear what you're mandating and what you think are best practices but are willing to leave to the team to decide on. For example, you might mandate the use of Java Server Faces and session EJBs because of constraints from the rest of the organization, but leave it to the team to decide whether to use PrimeFaces or MyFaces. Be sure to leave some decisions to the team so they feel some ownership over the decisions; otherwise, they lose interest because they see themselves as just robots carrying out someone else's decisions.

Be a leader for the development team. Sometimes that means you correct them or you tell them what to do or how to do it. But most of the time, it means you serve them, in the role of the servant leader. You very likely have more experience in almost every aspect of software engineering than the development team, which may be why you're the architect. Use that difference in experience and insight constructively. Make yourself available to the team to answer their questions or even debate your vision, guidelines, and

mandates. Check in with them every day to see if they have questions for you – go to them, don't wait for them to come to you. Let them see that you're there to make their jobs easier and more rewarding, not just to scold them when they stray.

If the developers have difficulty understanding how to implement your vision, work side by side with them for a short time. Pair program with them, and show them what you mean and how your guidelines benefit them in the long term by guaranteeing higher quality code, a more maintainable system, and a system that meets performance and scalability requirements on day one without requiring a ton of rework.

What about "emergent architecture", "self-directing teams", and "do the simplest thing that could possibly work"? Don't these agile practices mean that the developer team is free to ignore you and do whatever they think is easiest to crank out? No, we don't think so. Emergent architecture from self-directing teams has to operate within bounds. Those bounds are laid out in your vision for the system. As an analogy, think of a country's constitution. The framers of the constitution laid out a set of guiding principles, values, and boundaries. Legislators pass any laws they can, but those laws have to operate within the bounds of the constitutional principles. The framers may be long gone, but the nation has bought into and understands the purpose of the constitution, so new laws that continually emerge still adhere to the vision laid out there. So it must be with an architectural vision and guidelines. Now you are the framer. Show your team why they should buy into your vision and how it will benefit the product. As they make decisions, help them in their decision-making process to stay within your vision. Trust them to make the right decisions, but let them know you are there to help guide them.

"Do the simplest thing that could possibly work" is a statement about avoiding complication, not about avoiding complexity. You may already know that because your approach is mature, but it falls on you to explain the difference to the developer team and mentor them on distinguishing simple solutions (which may require complexity) from easy solutions (which may be complicated). Again, it's about getting your team to buy in to your vision so that it shapes every decision they make.

Stand up for your team. If you know that code needs refactoring or technical debt is mounting, be their advocate in planning meetings with the business team. Explain to the business team why there is long term value in paying off the technical debt, and insist that the team be given time to address the debt. You probably occupy a more senior position in the organization than the developers or even the lead developers, which makes it easier for you to make this argument than for them. This means you must attend planning sessions and participate actively. If you don't, the developers may feel unable to push back against the business team. If you then come by and remind the developers that the technical debt is out there and must be paid off, you're guilty of drive-by architecture.

So, is there a role for an architect on an agile project? Yes, there is. But rethink it. You have years of experience and education, and you can tell right away whether a particular software solution will truly meet its business goals when it goes live, and then over its planned lifetime. It would be a waste of those talents if you tried to impose your knowledge on the developer team from afar, trying to force them into following the true path to success. Lead them to that path instead. Both you and they will feel a deep sense of accomplishment and reward.