What Makes a Good Software Architect?

Table of Contents

SEI WEBINAR SERIES | Keeping you informed of the latest solutions

Carnegie Mellon University

Carnegie Mellon University

Copyright 2016 Carnegie Mellon University

Copyright 2016 Carnegie Mellon University

What Makes a Good Software Architect?

What Makes a Good Software Architect?

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

John Klein and Andrew Kotov
Hosted by Will Hayes

**003 Presenter: And hello from the campus of Carnegie Mellon University in Pittsburgh, Pennsylvania. We welcome you to

the SEI virtual event, "What Makes a Good Software Architect." Depending on your location, we wish you a good morning, a good afternoon, or a good evening. My name is Shane McGraw. I'll be your moderator for the presentation, and I'd like to thank you for attending.

We want to make today as interactive as possible, so we will address questions throughout the discussion and again at the end of each separate discussion. You can submit your questions to our event staff at any time by using the Questions tab on your control panel. We will also ask a few polling questions throughout the presentation, and they will appear as a pop-up window on your screen. The first polling question we want to ask is: How did you hear about today's event? And that will be posed now.

Another three tabs I'd like to point out are the Download Materials, Twitter, and Survey tabs. The Download Materials tab has software architecture-related work and resources from the SEI available now. For those of you using Twitter, be sure to follow @saturn_news. Once again, that's @saturn_news, and use the hashtag #seiwebinar.

Now I'd like to introduce our speakers for today. First, our facilitator. Mr. Will Hayes is a principal engineer at the SEI. He provides direct lifecycle management support to major Department of Defense programs. Throughout his

26-year career at the SEI, he has supported numerous commercial, government, and defense organizations, providing consultation and coaching for a wide range of roles. Mr. John Klein is a senior member of the technical staff at the SEI where he does research and consulting in enterprise and system architecture, working with commercial and government customers in domains that include healthcare, analytics, financial trading, and command-and-control. Andrew Kotov has recently joined the SEI as a senior member of the technical staff. He works with government customers to evaluate and improve their architectural solutions, and prior to joining the SEI he worked as a software engineer and architect in various domains, such as reliability engineering, supply chain management, role-based systems, finance, and healthcare.

Now I'd like to turn it over to our facilitator, Mr. Will Hayes. Will, welcome. All yours.

Presenter: Thanks. Welcome everyone. So let's dive right in to the focus of this event. So John, what makes a good architect, and how does one find their way into that role?

Presenter: Well Will, like a lot of questions that relate to software architecture, the answer to that is, "It depends."

Presenter: And John just demonstrated the very basic skill of software architects, to say, "It depends," because it gives you five seconds to provide a more meaningful answer.

Presenter: Right, right. So it depends. There are a number of skills you need, and the key thing is matching the skills to the point that the system is in in its lifecycle. Different points in the lifecycle, as we'll talk about, have different skill needs, and so it's important to match those, and it's a mix of technical skills and also other kinds of skills.

Presenter: Actually, communication skills are important, leadership skills are important, because you need to communicate with various stakeholders, including business stakeholders as well as technical stakeholders, and you have to do it throughout the whole lifecycle of the system.

Presenter: So maybe we can do the polling question.

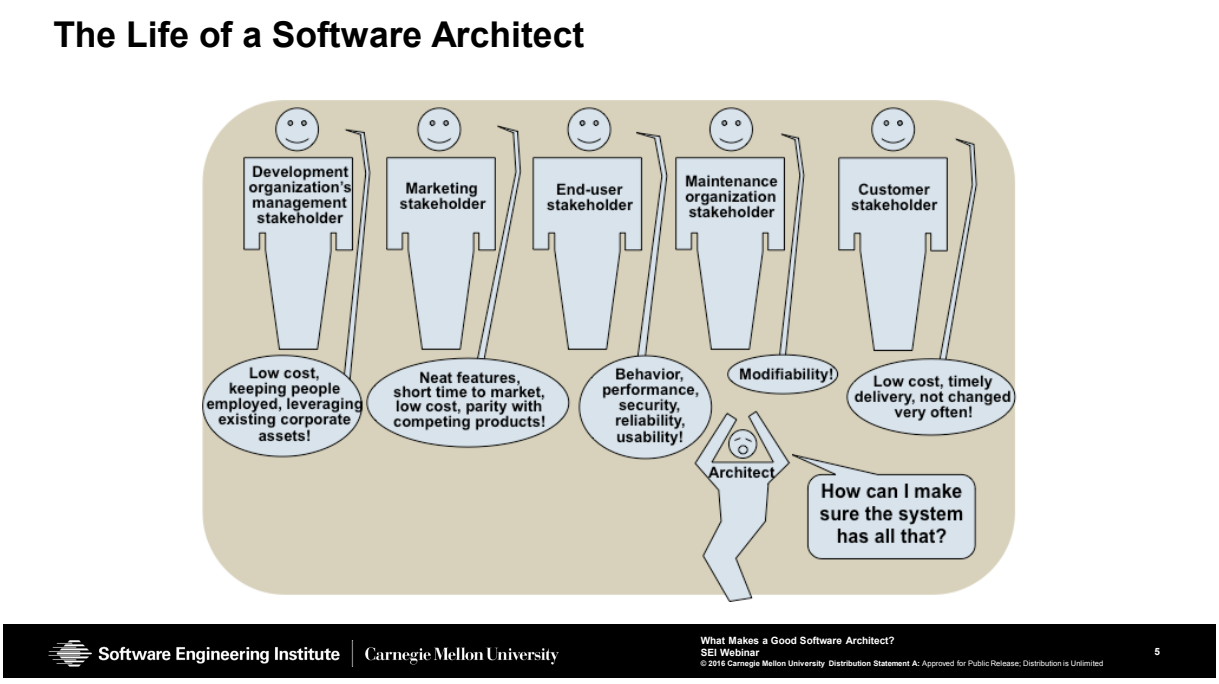
Presenter: So there's a polling question that's a natural follow-on to this, and I'll ask Shane to go ahead and--

Presenter: Okay, we're going to pose that question now, and the question we want to know is: What is your relationship with software architects? As I mentioned at the beginning, we're going to launch some polling questions throughout

today's event. Particularly this one's going to help us get an idea of who's here with us live and where to steer the conversation. We want to maximize your time here with us, so take about 15 or 20 seconds to vote. We'll turn it back to Will.

Presenter: So while the audience is address threat question, Andrew, let me ask you a little bit about the important things that an architect does in this role that they play.

The Life of a Software Architect



**005 Presenter: Sometimes it's probably easier to say what he doesn't do, but it's a technical leadership role where you have to provide a technical solution to a business problem. So first of all, you have to define what the problem is, understand what the business drivers for the solution are, their quality

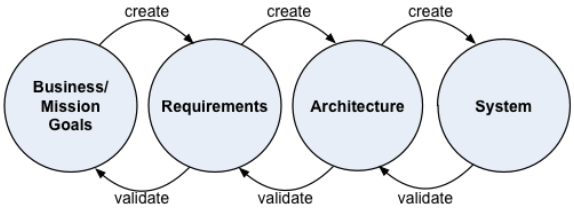
attributes, provide design concept, select the technology, and work with developers to make sure you can actually implement that solution, and work with other various business stakeholders making sure that solution can be supported in the field.

Presenter: Okay, we're going to chime in with those results. We've got 44 percent that are software architects, 17 percent want to become a software architect, 8 percent managing software architects, 21 percent "I work on projects with software architects," and 10 percent at Other. So hopefully that will give us an idea of where to take the conversation today.

Presenter: So if we think about the things that architects do, in light of who we have in the audience, how would you want to elaborate on that, John?

What do architects do?

What do architects do?



**006 Presenter: Well, the main role of software architecture is to bridge between the business goals and the system that we're building, and to do that there's a couple steps we go through, taking business goals and extracting the architecturally significant requirements, the functionality and qualities. From that we've got to build an architecture, design it, create it, analyze it, evaluate it, document it, and then finally we're going to use that to build the system, and we can do this in an iterative fashion, we can do it in a little bit bigger design up front. So lots of ways you can go through it, but these are the basic steps that we need to pass through. And there's a set of technical skills that you need to do this. As an architect, you need to be able to do architecture design, design at scale. You need to be able

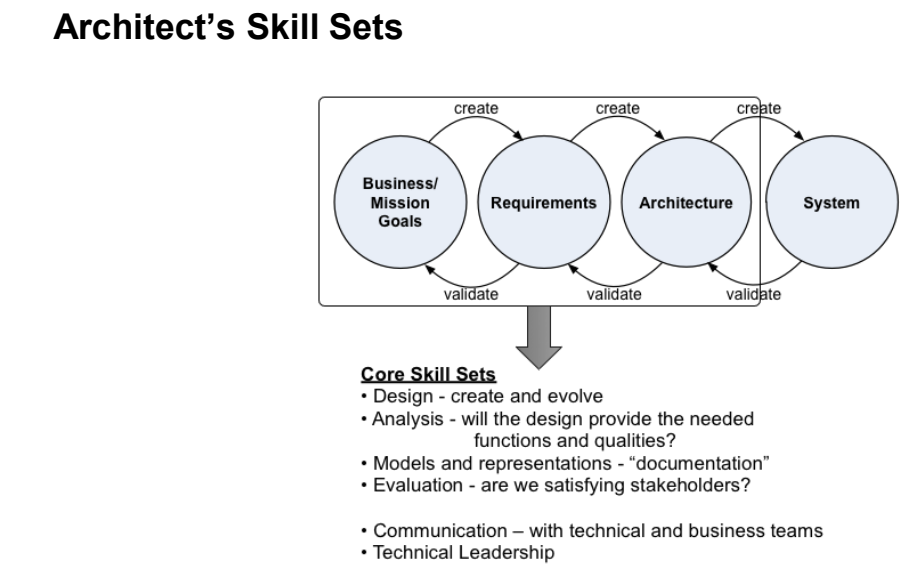
to analyze your design, decide whether the design is going to meet the functionality and qualities that you're looking for. You need to do some modeling and representation. Some people would call that documentation, but it doesn't necessarily end up as big, thick paper documents. And you need to be able to do evaluation. "Is this going to satisfy everybody's needs?"

Presenter: So there's a pretty high demand on the technical acumen for an architect. Andrew, if you could talk a little bit about the role you play in dynamic, ongoing efforts-- the role the architect plays-- with these skill sets.

Presenter: Sure. I think it's important to understand when you're transitioning from a software engineer to a software architect, the technical skills are still going to be used but you're going to use a different scale, but you can start with overall design of a system, so it's a really high-level view of the system, and you still need to provide technical solutions. So in the same time, you have 360 degree of the system, view of the system, and you're concerned with different aspects. So a lot of times it's zooming in and zooming out. You're zooming out, provide a big picture, create proper concepts, create proper elements of the architecture, relationship between them, and so on and so forth; but then you're zooming in sometimes to a different level. If you use off-shelf program,

can it be licensed? Can it satisfy your needs? So there's a lot of things like that. And throughout the whole process the architecture becomes the conduit of communication between the various parties. So we have to stay in touch with them. You have to listen and understand what they say, and especially when you start talking to business users, which you switch from technical users, they start using different language. So you have to be cognizant of that change and make sure you custom your message to those users. That's where I personally struggle a bit with that, and as a good ratio, we have two ears and one mouth, so that's a good ratio to keep between communications.

Architect's Skill Sets



**007 Presenter: So there's a pretty high demand on understanding who your communicating with and

being able to articulate these fairly complex subjects and still keep an eye on the trajectory for the system. That's a demanding job.

Presenter: Absolutely. And it's really ongoing process. You can't just put the stack of documents, say, "This is the architecture, and I'll see you guys when you're done with the release."

Presenter: Right. So the communication and the leadership skills are important, but the technical skills are equally important, I think, because at the end of the day your job is to deliver a system, and so all of the communication and leadership-- and the thing that distinguishes a product or a program manager from an architect is the technical skills, the ability to design that architecture, the ability to analyze it, model it, evaluate it. And so those are really important things. Otherwise you don't have anything to say, so.

Presenter: And you have a finite amount of time, because the time to market these days is pretty critical, so all these activities are tied to a particular program and you have a business goal in mind, then you have to deliver that system .

Presenter: So there's another polling question-- ready to go to that?-- that helps elaborate this point.

Presenter: So we're going to launch that question now. It should

be up now. We'd like to know exactly what the architects in your organization do. So you'll see that on your screen now. Now, some of your organizations may do more

Polling Question

**008 --More than one of these things, but we could not set it up as multiple choice, so pick the one that's most, and then if you do more than one, feel free to type it into our question box, just so we get an idea of some of the other results coming in. And while we give you about 30 seconds to vote for that one, we've got some great questions coming in already, so I'm going to fire them off to you guys and see what we got. From Scott, asking, "How best to integrate software architect into an agile dev/ops environment?"

Presenter: Oh, great question. What do you guys think?

Presenter: Do you want to start?

Presenter: I think probably dev/ops should be renamed to arc/dev/ops. So you just allow sometimes for the architecture, and really you design, you implement, and then you support, and this is the lifecycle. So basically you have to allow some time for the architecture, for big view and design up front, and you can distribute that throughout the different sprints or have a different swim lane, and then just have it as ongoing process. It's a practice, so it's just not a state-- continuously, dynamically moving, and it's a practice.

Presenter: Yeah. So I think there are some technical skills. One way to look at dev/ops is it brings in a different set of qualities for your architecture. Deployability, incremental deployability, and observability become very important qualities in your architecture that you need to build in and treat those as first-class things.

Presenter: So then a successful architect working in a setting where dev/ops is a dominant concept, the skill set they have to be successful with dev/ops is a slightly different one than different environments, perhaps?

Presenter: Maybe a little bit different skills, a little bit different

priority. I mean, at some level a lot of architecture is understanding what are the important qualities for a particular system and then constructing an architecture that's going to satisfy those qualities-- different qualities for different contexts. And then bringing in the technical background to be able to satisfy those qualities. So when you get into dev/ops, there are a lot of off-the-shelf tools that you want to be taking advantage of, some of the incremental deployment tools, configuration management tools, those sorts of things.

Presenter: Helps you speed up the progression that you need to go through.

Presenter: Yes.

Presenter: And at the same time the ops part of it gives you an ability to get the direct feedback. That's raw feedback from the use in actual environment, or maybe even various environments, making sure that decisions that you made early on were correct ones and you correctly identify market of customer needs.

Presenter: That seems like a very important connection to the implementation, connection to how the system is benefiting the user of it. Architecture is not something that's just dreamed up in advance and you fire and forget. This is an ongoing interaction.

Presenter: Yes.

Presenter: Exactly, and you judge by the result. It's not just a stack of paper or drawing on the board. You judge by the result and the system actually can be performed in a real market environment and that gives you an idea if you were right or not.

Presenter: Great. Should we go for the poll results?

Presenter: Yeah, let's show the poll results, and I'd like to work in one more question if you don't mind, too. So let's see here. We got 41 percent focused on architecture design, 7 percent development, 7 percent architecture analysis, 7 percent modeling or documentation, 9 percent architecture evaluation, and 3 percent communicate architecture and 26 percent provide technical leadership, and we had a number of people typing in "All of the above" into the box. So the other question I'd like to get to real quick just while we're in this session was Steve's, because it ties in so well here: "Are there MBTI-- so Myer-Briggs personality types-- that have higher aptitude to be software architects? Perhaps INTJ or ENTJ. In the inverse, are there personality types who have a low aptitude for this work?"

Presenter: So what makes a bad software architect.

Presenter: If you're a severe introvert and can't communicate, that makes it very difficult. That's still possible, but it makes it very difficult.

Presenter: Yeah, I'm not sure how successful people would rate me. I'm moderately successful. I'm an INTJ, so.

Presenter: So there are some observations that despite your preference for how you might behave in a default setting, when you take on a profession and identify with a role that your obligation and your passion for that role doesn't necessarily find itself limited by what you might do at a resting state, if you will.

Presenter: That's right. That's right. The thing to remember about Myers-Briggs is, as you say, it's what you would do at rest, and as long as you recognize that your tendency is going to be in that direction, you can behave differently at the office.

Presenter: Do webinars live.

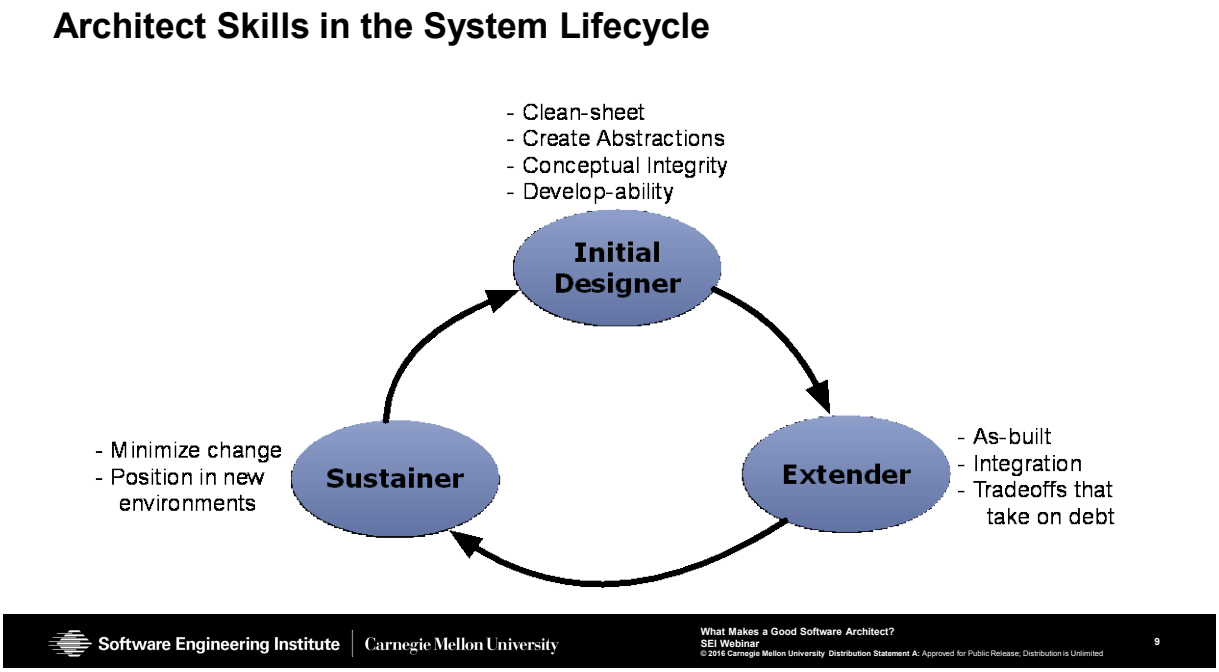
Presenter: Yes.

Presenter: Okay.

Presenter: Back to you, Will. Yeah.

Presenter: Yeah. So what more would you like to say about the different kinds of skills and roles that people play over time? I think that's the next theme we want to work in.

Architect Skills in the System Lifecycle



**009 Presenter: Yeah. So one of the things that I've observed is that systems go through different phases in their lifecycle. You start with some initial design of either an entire system or a major new subsystem, and so you've got to make this transition from nothing to something, and then once the system is built, there's a tendency to want to make small changes to extend it, and as we build those extensions on, we're adding value. The cost of each increment of value is lower because we're building on top of this big base. And then over time the system gets old. It gets a little bit fragile. Maybe we've accumulated some technical debt, and the original use of the system-- the technology that surrounds it may have passed-- and so we move into the sustainment mode, where we want to keep the

system around, keep it producing value, but we don't want to change it very much, if at all. And so as the system goes through those different stages, the skills that you need as an architect to be successful are a little bit different. At the beginning, to get from that nothing to something, the clean sheet design, you need a set of skills to be able to determine and create the right abstractions, maintain the conceptual integrity of the architecture, keep it looking as if it came from one mind, keep things consistent across it, and it's got to be developable.

Presenter: And oftentimes it's a favorite part because architecture is-- essentially that stage is modeling, the modeling behavior of a system. You want to make sure that you create proper abstracts. And at the same time, you always model for change. Right? So you know whatever you develop now is going to be changed later on, but when you go to the standards stage, that you're going to test your skills, how you designed the system, because the quality of your design is going to be inversely proportional to the number of changes you need to do every time you implement a new feature.

Presenter: So at the outset, the successful architect is skilled at communicating the vision of what is not yet visible to everyone. It's still at a conceptual level. And then as we enter into this middle stage, it's how to understand the tradeoffs among the competing value drivers?

Presenter: Yeah, so there's a couple of things there. One is to have a deep understanding of the as-built system, right? What are all of the interfaces, what are all of the side-effects on interfaces, what are the places where you could squeeze in a little bit of functionality. And so this is a place where often a senior developer steps in and starts taking on this role of making system-wide architecture decisions because they've got that great knowledge of the as-built system. But as you say, you've got to make good tradeoffs.

Presenter: So that must be particularly challenging if you've been very successful in the early envisioning of the system. You've attracted more people to be interested in modifying your system and to provide more paths for value through it. It must get much more difficult in that middle stage.

Presenter: Yeah, definitely now software architecture basically is a team sport. So you can't do it all alone, and the architectural role is just really wide, so when you go deep you have to make sure that you got support and you have the buy-in from various stakeholders, and the main one is definitely technical staff who are working on the system, and you have to have a team of people you can consult with before presenting to a larger audience, maybe. You have to work with people. You have to work with younger developers, making sure they understand and adopt your

ideas. So there's a lot of coaching going on. So you basically work with many parts of the organization.

Presenter: And you bring up a good point there, Will, about the difference in the vision, because at the beginning part of your job as an architect is to maintain the purity of the vision and the conceptual integrity, and a lot of what happens as we extend the system is we make compromises. And so one of the anti-patterns that I've seen is the architect has a little bit too much ownership of the system and isn't willing to make those compromises, not willing to give up that purity. And so as you move into the extension phase, they propose doing things the right way as opposed to maybe incurring a little bit of technical debt in order to produce value faster, and if you're making the wrong tradeoffs, from a business point of view you're not going to be successful. So it's a transition.

Presenter: At the same time, you already have-- at that time you have probably gotten certain patterns, certain styles that do work, so you can just make sure that you continue using them or make adjustments if they do not work. So the architect needs to have a pretty large tool set that they can use to their advantage, including also how the architectural process works within the organization; make sure that the architectural process can support his design decisions, or if it's a group of architects, design decisions of that group.

Presenter: So the fast feedback that might come from employment of dev/ops, for example.

Presenter: Sure.

Presenter: You just mentioned a team of architects. It would seem to me that you would benefit from broadening the focal point for this activity when you're trying to deliver value at a faster pace and in small increments. Has that been your observation?

Presenter: I worked on a team where there was a group of architects, and the exchange of opinions was very valuable. In the same time, if a lot of people-- so this was a group of architects who were developing and designing. Sometimes it could be a challenge to keep a balance. It would be nice if you have a person who was just in charge over the architecture and then he relies on the opinions and works with a group of architects. That probably, in my opinion, would be the optimal way to do that, especially if it's a large system and there's a lot of different concerns so it's hard to keep track of all the various parts.

Presenter: Yeah, and that's an important point, because even in this extension phase, although we're not designing the new system, and the changes that we're making may be small in scope, the impact is still broad across the entire system. We're going to be impacting the qualities, performance, modifiability,

and so these are our architectural decisions, and often that's hard for developers stepping into this role to see that just getting it done fast is not enough. And on the other hand, getting it done exactly right may be too much, and having the experience to find that balance--

Presenter: Or sometimes notice of the change is just a local, or it's a systematic change throughout the whole system. Maybe it's time to review how your exception handling mechanism works in the system, how you can-- do you have a safety net to capture any problems with the system? Do you have proper auditing, proper logging, proper communication channels, and so on and so forth? So it's a good time to review early on cross-cutting concerns of the system.

Presenter: So as you enter into sustainment now, it seems the balance again changes among these things-- the architectural purity versus the functionality delivery. Can you comment on that?

Presenter: Yeah. So in this phase, the system is becoming more fragile and so we want to minimize the changes, and so one of the jobs of the architect is to be able to do some different modeling and representation to show how this-- what might be termed a legacy system-- is still relevant in these new environments. So being able to update the models, update the documentation, and back it up with analysis and evaluation to

show that the system is still appropriate.

Presenter: So you're deriving an as-built viewpoint on the architecture, which might differ from the vision that originally created it.

Presenter: Usually by the time you get to this point, the as-built system is very much diverged from what you originally intended.

Presenter: And probably a good indication that you have to come back to the initial designer role and start designing a new system, because you already know what works, what doesn't work, and probably throughout the lifecycle of the application, technology change or approaches change or even business change. So this is the critical time to start working ahead of time and maybe create a prototype of a new system and start throwing the ideas and see what works the best.

Presenter: Yeah. One of the things that I've seen though is that architects that are successful as sustainers, often they've got this huge domain knowledge and they understand how people are using the system. They may have gotten a little bit rusty about the current technology, current development practices-- out of practice, a little bit. And so what I've seen managers do often is they say, "Well, this person knows exactly what the old system does, so we're going to have them design the new system," and making

that transition is often very hard. Moving from a role where you're communicating outwardly as your primary task to a role of now having to go back to, "I've got an empty sheet of paper in front of me and I've got to start back at design," and pay attention to these things like developability and dev/ops, which a few years ago didn't exist and you may not be up to speed on all the nuances of that. And so it's a place where I've seen people fail. On the other hand, I've seen successes where managers put together teams that brought the architect who was the good sustainer together with some maybe younger architects that had the technical skills and the current technology. They were able to build a team that produced a new product very successfully.

Presenter: So before we ask about strategies for that skill development, let me ask Shane to go ahead and launch the next poll.

Polling Question

Polling Question

Does your organization offer or require specific professional development for architect (e.g., classes, apprenticeships, certificates)?

- Yes
- No
- Not sure

**010 Presenter: So we'd like to know: Does your organization offer or require specific professional development for architects? For example, classes, apprenticeships or certificates. So go ahead and vote for that, and while we're doing that, we have a ton of questions coming on, so we're going to handle this a little like a game show, if you don't mind. We're going to fire off as many questions as we can. If there's something you don't want to answer, let's just say "Pass" and we'll get to the next one. But Boris would like to know, "What is the role of an architect in the non-software cases to other situations, such as services?" And feel free to pass if it's something that-- because we got so many.

Presenter: So architecting a service system, considering--

Presenter: Again, service is modeling some kind of a business process, so that process needs to be properly modeled, and making sure that the business goals of the process are satisfied, and you have some measurable business goals, especially in services. There's some service-level agreements, availability requirements, and so on and so forth, and there's some quality attributes of the system that needs to be satisfied. So there's a lot of stuff on just the modeling of that that could be done by an architect, and considered to be architectural work. And then implementation, it depends on the situation, I guess.

Presenter: So just as with a physical system, the quality attributes really drive decisions you're making. In a service system, those tend to take on a different flavor.

Presenter: And it's important to consider them first so you don't have to change the system later. So later optimization usually does not pay off for a large or complex system.

Presenter: Then Claudia asking, "How do you describe roles of the software architect versus the solution architect, versus the enterprise architect?"

Presenter: Solution architect essentially works with the already-existing pieces to make sure that it's

composed into some kind of a form. They work and provide solution to the customer problem. Software architect usually develops a solution. That's some kind of developing work or extending existing architecture, starting the new one, or sustaining the other one. That's how I would see the difference. And solution architect is also much concerned with existing environment of the customer, making sure that the proposed architecture can fit that environment.

Presenter: Right. Yeah. And there's different levels of architecture and one person's-- one level's solution is the constraints and requirements for the level below it. So enterprise architecture creates the constraints and requirements on the solution, and the solution decomposes it into maybe software and other processes, and those become the constraints and requirements as you move down. It's a lot of the same skills. As you move up to the enterprise level, you do less modeling of actual processes and it becomes a little bit more policy than structure.

Presenter: But just as you spoke of, the architecture transitioning through phases for a physical system, we could think of an enterprise architecture similarly being stressed by the different needs being placed on the business, and there might be a time when we need to really consider a new viewpoint on the architecture for the enterprise.

Presenter: Definitely.

Presenter: Because you have more concerns. You have that huge chart of concerns.

Presenter: Right. Great question.

Presenter: All right, let's get one more and then we'll share the results, if that's okay with you guys. From Robert, asking, "What are the panelists' thoughts about risk analysis and mitigation during architecture design?"

Presenter: One way to view architecture is it's entirely a risk-reduction practice. Right? I mean, you could just put everybody in the room together and say, "Okay, let's start coding." The reason we do architecture is to reduce the risk that the dependencies in the code are not going to support incremental deployment, reduce the risk that the performance of the system isn't going to satisfy our initial set of requirements. So architecture as a whole is definitely a risk-reduction practice. In fact, I know an architect who works for a consultancy-- they do a lot of fixed-price work-- and he reports to engineering but also reports on a dotted line to the risk management subcommittee on the board of directors.

Presenter: Probably in the regulated environment it's a bit more important because risk analysis is a part of the quality process, so you have to identify risks that the system

is subject to or can cause harm or substantial loss. So there you have to work with the person doing that work to make sure that all proper risks are identified.

Presenter: Right. And so tangible things you can do with that though are-- part of architecture is developing structures, making decisions. You can isolate the high-risk parts of your system so that the impact of having to change them if the risk does take place-- your performance or memory or whatever the challenge that you have there. So separating out the high-risk from the low-risk, initiating prototyping and deciding what the goal of the prototyping should be based on the qualities that you're trying to reduce risk for-- these are all architecture-centric practices that will help you to manage risk throughout the entire lifecycle.

Presenter: You can work probably on the training materials to make sure that you work out the process. That minimizes that risk, and that becomes a part of a training program for the system.

Presenter: And it sounds like there's a very close connection to the concept of technical debt here, that the investments you're making, the forward-looking things you're doing, you're doing that to buy off or to prevent technical debt from accruing. Your choices to perhaps invest less might be balanced by technical debt that you have to then deal with later.

Presenter: Yes. So one form of technical debt is risk, that you decide to carry that risk forward and hope it doesn't happen.

Presenter: So the skill there is to be able to identify these particular issues of technical debt.

Presenter: Terrific. I think we have poll results?

Presenter: Yeah, and I wanted to just chime in real quickly before you share those poll results. I actually heard from someone on Twitter, a native Pittsburgher in Italy watching the webinar, so we wanted to thank today, and I wanted to let him know it's 70 degrees today here in Pittsburgh, in March. So.

Presenter: So come back.

Presenter: But it is raining, of course. Anyway, back to wrap up the polling question. The question was, "Does your organization offer or require specific professional development for architects?" So we had 30 percent with yes, 54 percent no, and 16 percent not sure.

Presenter: So if you could talk a little bit about what would that look like. What is professional development in this arena for you?

Presenter: Well, I think the first thing is to note, that we're talking about what makes a good architect, but architects don't work in isolation. One of the pictures we had earlier

showed them being beaten up by lots of stakeholders, right? And so they're working in this environment, and so you've got a set of individual skills that are important, but you also need to be working in a team that is going to be able to take advantage of the architecture that you build. So the team has to have certain competencies to be able to build the system as designed, and then you need to be working in an organization that respects the need and the role for architect, that does respect architecture as a risk mitigation mechanism, and is going to put together some sort of professional development track. So individual, team, and organization competencies are all important if you're going to be successful.

Presenter: Like we said before, it's leadership skills, it's the communication skills, the technical skills. So you can work on all of them individually if you want to. There are courses available and materials available, and of course in the technical skills, making sure that you're familiar with architectural principles and practices, that you're familiar with modern technologies used, approaches. So that's still collaboration and communication with a lot of development forums and see what kind of problems they have, how are they trying to solve them. Learning about new patterns, new approaches, new tools, and make sure you can incorporate. So you have to keep that toolbox fresh all

the time, making sure you can solve business problems.

Presenter: Yeah, and there's a core set of tools, right? And coming back to some of the things we talked about earlier-- design analysis, modeling and evaluation-- and those are-- they're eternal. You always need to be able to do that. Part of design is things like architecture patterns that Andrew just mentioned. They show up in different ways. For example, now in big data we've got streaming processing systems. Well, a lot of that looks like pipe and filter patterns, which go back to batch processing and card decks and things that maybe some of our audience has never seen. So some of these patterns are eternal, and so having good familiarity with them-- the qualities involved with those don't change, right? A pipe and filter pattern introduces latency at the expense of modularity and flexibility and composability, and that's true whether we're talking about decks of punch cards or a streaming architecture for big data. So there are some things that are constant. There are other things that are new. And so keeping that mix of the core things that you can get training for, but also a lot of it is practice, right? It's hard to become an architect. It's one of those things that teaching people, particularly people who are coming out of school maybe with a master's degree, as part of a master's of software engineering program, trying to teach them architecture-- because so much of it comes back to

this risk mitigation, and until you have enough experience to understand what the risks are, it's hard to know what to mitigate or even why you'd want to worry about mitigating it.

Presenter: Or creating proper concepts in this case. So one of the areas where you also can work, if you can learn about the business area where you work, maybe there's some nuances that are not accounted for. At the same time, when you're talking about the principles, practices, design patterns, if that conversation is going on in your organization then it means that next time you try to communicate certain architectural artifacts, then everybody is going to be using the common language and they understand what you're talking about. So it again brings up the continuous practice of the architecture within the organization.

Presenter: So there's a core set of knowledge. It's an understanding of technology. There's the context that the organization provides. If you think about an individual person's career path, if you could trace perhaps some varieties of where people start, what kind of experiences really contribute to them being a successful architect as they go through their career, could you comment on that?

Presenter: Andrew, you just came out of industry, so.

Presenter: I think what was helpful to me working with the systems where it was a black box, so I have to learn the system just by myself or very little help, that gives you an idea if the system is organized properly. Can you understand the system by just looking at the modules? That gives you an idea about that organization. So then working through the ranks of various software architectural applications is very helpful. Different areas use different patterns and different applications. You get requirements in a different way. Quality process is designed a different way. System is tested in a different way. So that prepared you-- as soon as you get the breadth of experience, that's probably a good thing when you-- good stage when you can think about the architectural position. And the architect is not a position-- is just not a tag on the door of your office. It's a role. So the role needs to be played properly, and it's a leadership role, so there's a lot of things that are associated with leadership, because if you're a leader and nobody walks behind you, it's just a walk in the park.

Presenter: Yeah. Yeah. And that's a good point, that it's a role and not necessarily a title. So there are often agile teams where there is nobody who is the architect, but yet there is someone who is the architect. There is someone who takes ownership of those system-wide decisions. So you may not have it on your business card, it may not be on your door plate, but you need to take that role on.

Presenter: And it sounds like the variety of experiences, a collection of contrasting experiences, is a really valuable component of this.

Presenter: Yes.

Presenter: But for organizations, it's important to understand that if the system is complex enough then you probably need to have a single person or group of people responsible for that, because otherwise you're going to have lava patterns, when you're doing one thing for some period of time, then the other thing for a different period of time, and the system becomes very amorphous from the design perspective because of the different patterns used in different parts of the system.

Presenter: So should we take some more questions?

Presenter: Yeah, we've got about five minutes left with John and Andrew for this segment before we bring in Ipek and Michael, so we're going do some-- back to the rapid-fire questions. So Rodney would like to know, "What is the typical percentage balance an architect spends between design and coding?"

Presenter: Ooh.

Presenter: Depends probably on the stage. Initial design, there's a lot of design, and it goes to prototyping, and that's as much coding you probably can do at that point in time.

When you are an extender, you're probably better off pairing with existing developers, making sure that you can change that functionality properly. And when you're a sustainer, you work, again, mostly with developers. So I would say 80/20-- 80 percent design, 20 coding. But it could be more during the initial stages. It could be 100 percent coding.

Presenter: Yeah, it depends, right? So a couple things to keep in mind though is the architect has to do the design, because nobody else is going to do the architecture design. Other people will do coding. So getting sucked in and doing too much coding leaves the architecture design undone. The second thing is it's hard to make that shift, right? If you drift in and out of coding, it's a hard thing to do. It takes a little while--

Presenter: Context switching--

Presenter: Yeah, the context switch is a killer, so that's something else to keep in mind. And the third piece is because the architect's first job is doing the architecture design, often if you are going to do coding, and that is a good practice that architects code, for a lot of reasons, stay off the critical path, because that allows you a little bit more flexibility in balancing how you do it.

Presenter: And make sure you fully own recommendation and practices that you gave to others.

Presenter: Yes.

Presenter: That sure sounds like great advice.

Presenter: And often that's-- yeah, often the kind of coding that an architect does is sort of the exemplar. Create that first example of how to use the architecture, show how to use the framework that you've developed. Those are good places for architects to start.

Presenter: And programming and code reviews are the other two means of communication in terms of coding.

Presenter: So coding is partly to communicate the vision the architect has.

Presenter: It becomes a good way to do that, yeah.

Presenter: Nice. Another question?

Presenter: Okay, from Lee, asking, "Many years ago, Barry Boehm remarked at most maybe 80 percent of software problems are directly related to design flaws. Do you agree?"

Presenter: Wow. It's hard to say because design flaws could be caused by bad requirements. So if requirements were not adequate or the quality attributes were not defined the way they were passed, or the system became just a pack of features when there's no priority in how they were put together in logical

coordination between them. So the answer is it depends.

Presenter: I got to disagree with you there. You don't argue with Barry.

Presenter: Good answer.

Presenter: All right, we got about a minute. There's two more in here, one from Nicholas asking, "How, as an architect, can I push technical innovation in an environment that is technically stale?"

Presenter: Oh. Laden with technical debt, perhaps? That might be a good question for the next speakers.

Presenter: Okay, we'll go on. Jim would like to know, "What are the key ways software architects can influence others without being a manager?"

Presenter: Architectural role is-- again, it's a leadership role. You can influence, but you don't manage, you don't fire anybody. You can agree or disagree with a design, but you have to present a case, because it's you solving the problem. So you have to be able to explain why it's a better solution, how this solves the problem, and how you're going to implement it and how you can install it in the field. You have to think about all the aspects of that.

Presenter: Yeah, and the approach that we espouse at the SEI is to start

with business goals and derive things from that, have traceability back to business goals so we can talk in terms of business impact and not get into arguments about which programming language is better. So keeping it at that level is often a way to promote change.

Presenter: Great. That's a nice place to stop.

Presenter: Yeah, excellent discussion. Your time's up, so as we get ready to transition to our next set of speakers, we had tons of great questions in the queue there obviously we didn't have time to get to, but I wanted to let everybody know there is a software architecture LinkedIn group that we maintain. We'd love to see these questions posted to that group to get some feedback and discussion going there. If you're on LinkedIn, just search for SATURN on your groups, and that stands for SEI Architecture Technology User Network. Feel free to post your questions there. Couple thousand members there would love to keep this discussion going there. And also wanted to let you know, speaking of SATURN, our 12th annual architecture user network event, which is called SATURN, will be held in San Diego, California, and that will be May 2 through 5, and SATURN 2016 will feature the Internet of Things as one of the themes of its four tracks, and to see the great lineup of speakers, make sure you download the SATURN conference guide that is in your console now.

You can see keynote speakers-- Grady Booch, Joe Salvo from GE, and also wanted to let you know that all attendees today get 15 percent discount to the conference. For those of you attending late, I mentioned at the beginning there is that Download Materials tab. You can find lots of great information on architecture work from the SEI there. There's an IEEE paper there that John Klein wrote on what makes architects successful. You'll see training as well from the SEI that we offer. And lastly, there's a survey tab. We request that you fill out that survey tab upon exiting today's event, as that feedback is always greatly appreciated. So we're going to go back to Will. Actually, you know what? I'm going to introduce Ipek and Michael first as they join us onstage. So Ipek Ozkaya is a senior member of the technical staff. She works with government and industry organizations to improve their software architecture practices, focusing on research and development of software economics, agile development, architectural tradeoff and technical debt practices. Michael Keeling is a senior software engineer at IBM where he develops and mentioned IBM's Watson's Explorer and Watson platforms. Michael is an experienced software architect, agile practitioner and programmer, having worked on projects ranging from combat systems to search to web apps. Now I'm going to turn it back to Will. Will? All yours.

Presenter: Great. So we're going to transition now into a conversation about technical debt, and I think it's appropriate that we start with setting the table for what we really mean by technical debt.

Presenter: Well, I'll start off, Will. has been increasing interest in understanding what technical has been in the recent years, and I think it's very relevant to software architecture because there's an important aspect of it that relates to where the rubber hits the road--

What is Technical Debt?*

What is Technical Debt?*

- Exists in an **executable system artifact**, such as code, build scripts, automated test suites;
- Is traced to **several locations** in the system, implying ripple effects of impact of change;
- Has a **quantifiable** effect on system attributes of interest to developers, such as increasing number of defects, negative change in maintainability and code quality indicators are symptoms of technical debt.

* Term first used by Cunningham, W. 1992. *The WyCash Portfolio Management System*. OOPSLA '92 Experience Report. http://c2.com/doc/oopsla92.html.

**013 --And that comes with the aspect of technical debt that the architects as well as developers and the team have to be responsible of, which is it resides in an executable artifact of the system, such as the code, the build scripts, automated

test scripts. It's traced to several locations in the system because of the impact of the rework, and most importantly, if you understand the architectural tradeoffs well and architecturally significant requirements, you can actually quantify its effect, and I think that frames it well for the software architecture skills discussion as well. But what's your take on it, Michael?

Presenter: Yeah, I mean, certainly everything you said is true. I tend to look at it in a little bit-- I don't know, simplified. So technical debt for me and kind of the way we use it day to day is kind of the gap between the value that you need to provide and your ability to provide it. Right? So we've got some software, we've got some system that we have today. We've got features and things that we want to develop and that we want to provide, and sometimes there's things in the way that kind of prevent us from getting there, from providing that extra value, and that gap is kind of where I see-- or when we talk about technical debt, that's usually what we're talking about, at least on my team.

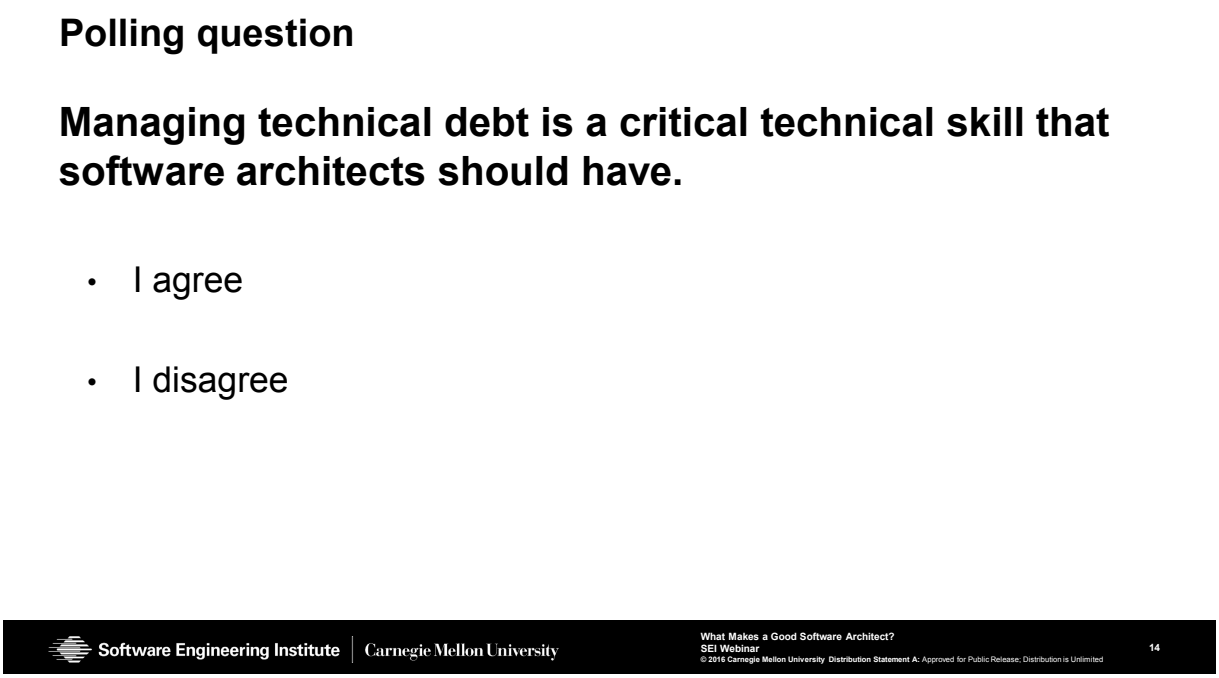
Presenter: Is it safe to think of it as an impediment to agility?

Presenter: Sometimes. Right? So it depends on what you're dealing with. Yeah, it can definitely cause problems.

Presenter: It's impediment to actually many of the developments.

Not just agility, but it's also sustainability and extending the system as well.

Polling question



**014 Presenter: So I think we have a polling question we'll jump to now. I'll ask Shane to introduce that.

Presenter: Okay. That polling question is going to be on your screen now. We'd like to know if you agree or disagree that managing technical debt is a critical technical skill that software architects should have. So we'll give you about 10 seconds to vote, and we'll turn it back to Will here, and then we'll get the results.

Presenter: So in our previous discussion, technical debt came up in speaking about the successful architect and the role they play.

Could you help make some more connections for us to architecture and the architect's role with technical debt?

Presenter: So it's all about tradeoffs. If you're able to understand the architecturally significant tradeoffs then you're able to get ahead of technical debt, because there's a view of the technical debt that it's actually code quality-- not following code quality rules or not being able to develop good code. However, we see over and over again it's really in long-lived systems these key architectural decisions that are made on early that start biting you back as the time goes on, and that really goes back to understanding what are the quality attributes of the system, which ones change, and actually how do you quantify that in the aspects of the system. Is your performance starting to degrade because you didn't take the right measurements or did not pick the right infrastructure at the right time? Or is it that you're not able to scale the system that's biting you? Or is it just a maintainability churn that you're having to deal with? So that connection is very important to recognize in understanding and getting ahead of technical debt.

Presenter: So the challenge of quantifying it, there are a number of different approaches people have been looking at, and if I could ask you to comment a little bit on that, Michael.

Presenter: Yeah, quantifying is tricky, right? So when you come back to this idea of value-- right? So this question you're asking, is this something that architects need to worry about-- I mean, architecture being at this intersection between technology and business, I think it puts it squarely in the architect's kind of role. Quantifying it though can be tricky, right? Because we're talking about what are we trying to do, where are we trying to go, and then how much is it going to cost us to get there, and you have to-- I don't know, you have to be able to estimate this in some way, or at least make a reasonable guess. Right? Because you are-- at least in my opinion, when you're thinking about technical debt in the right ways, you're making an investment. Right? So you're saying, "I'm going to take on a little bit of debt now." Right? You're making an active choice. "I'm going to take on a little bit of debt now, and it's going to get me this much value now." Right? Or soon. At some point, you may or may not have to pay that back. And what does that mean and how is that going to affect you later, that's when things-- measurement gets really difficult.

Presenter: So it's really another consideration in the business drivers that lead you to deploy a certain capability or not, and to try to get to a common framework of dollars with technical debt is an ambitious thing.

Presenter: Well, I mean-- yeah, maybe dollars. I mean, so-- okay, look. So we work with a marketing team and with our product managers, and they do a really good job of trying to quantify the amount of value that a certain feature could bring. Right? So if we go to market with a feature, what is that going to bring us? My job, working with my team, is to understand what is that going to cost to be able to do that, and part of that is understand short-term versus long-term, I guess. So if I'm able to deliver this faster and it gives us this value, to our users, and then users are willing to give us money to get that value, what if that then puts me in a position where the feature after that is not possible? Right? So we're constantly trying to work with this balancing act. Right? And sometimes we get it right, sometimes we don't, and that's really where this idea of managing our technical debt comes into play.

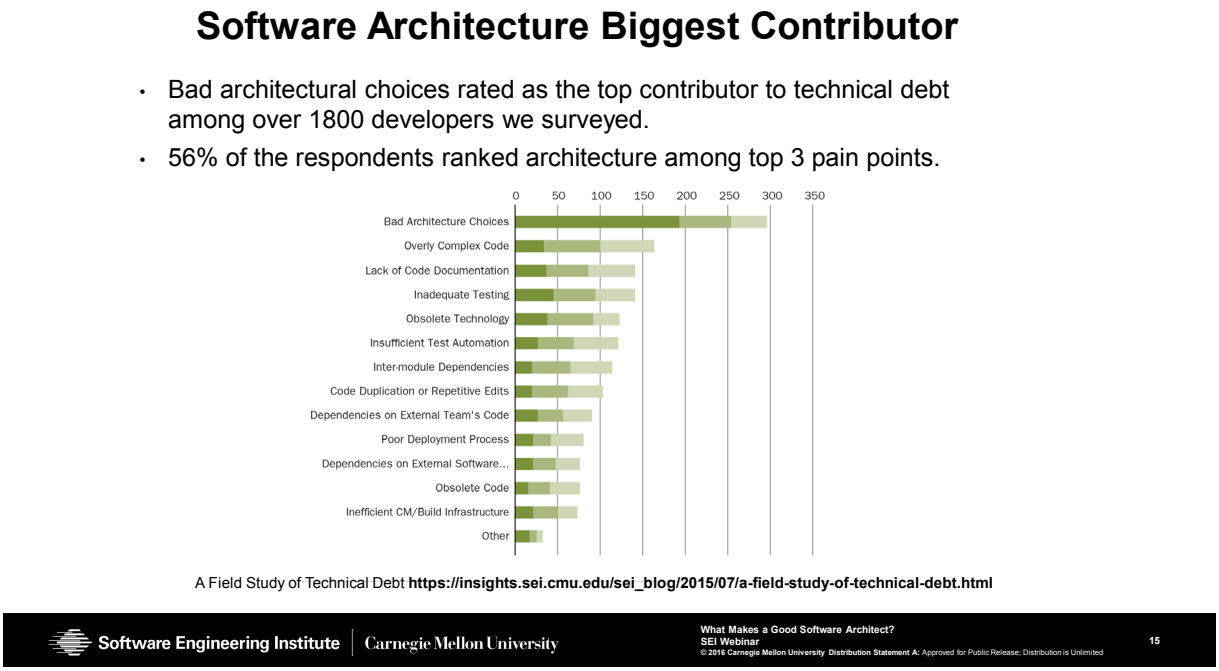
Presenter: Let's see how the poll came out.

Presenter: So we had 95 percent agree and 5 percent disagree.

Presenter: Well, I like the audience already. And it actually goes back to the discussion we were just having, Will, because what we find over and over again-- and that's really the answer to the riddle of the quantification. If you're able to quantify your short-term decisions and how-- what much you anticipate in long-term, it's based on the types

of systems as well. If you're working in a safety-critical system, those might be tighter in terms of the short-term if you're really developing for the long-term, whereas if you're working with a new system that's going to be out there and you're going to get feedback from the people who are using it, then that's a different set of choices. And when we actually ask the developers...

Software Architecture Biggest Contributor



**015 --This actually was confirmed as well. The number one leading reason of our technical debt in several kinds of systems-- and the audience can read this going through the link on our blog as well-- it's a survey with several organizations, about over 1800 people responded, and the number one leading biggest contributor of technical debt turned out to be bad architecture choices. So it's one thing for us to say

architecture and technical debt is related, but I think it's more concrete, and I think there's more evidence when we hear it over and over again from senior developers, architects and teams.

Presenter: So the choice you make near-term may limit choices you want to make later, just as Michael was speaking of earlier.

Presenter: Yeah, and this is an interesting graph-- right?-- looking at this. So when I look at the top things that people were indicating there, they're talking about things that in my opinion are-- well, with the exception of bad architecture choices-- relatively easy to chance. Right? So when you're talking about code complexity, I can refactor code. Right? Usually pretty easily. Testing. I can add more tests. Right? But if you've made a major structural decision, refactoring your way out of that can be extremely costly. So that's, I don't know, kind of interesting.

Presenter: Do we want to try to take some questions?

Presenter: We have lots of questions. You may get one-- there's so many coming in, you may get one that was left over from John and Andrew. So I'm still going to pose it to you guys, just because it's a good question. Owen wanted to know, "What general career advice can you give to someone who's doing software engineering master's, but

has worked in sales for the last seven years? What general steps do they need to know to follow, at the age of 32?" And the answer can't be just attend SATURN. So we have to something more--

Presenter: Oh wow. Well, there are lots of opportunities to be involved in open source development today that actually puts the skills of a recently-- I guess junior in terms of a career path and who does not have the experience, or he may not have the opportunity to have experience-- get involved with one of those projects. They actually have a ladder, and you could actually get really pretty decent both architecting as well as development experience, and that can actually help you understand what it takes. So that's one thing that comes to my mind.

Presenter: It might help you meet a mentor.

Presenter: Yes, that's another one.

Presenter: Yeah, number one thing is just write code, read code. Especially if you're coming from sales, you probably already have the business side of things down really well, so it's a matter of getting to know the technology and just getting some experience on the technology side of things.

Presenter: And just one more from Tim, a quick one. "Do you see architects as active coders?" And

then it says production code, reference implementations.

Presenter: Yes. Well, I think "active" probably has a balance. Architects should definitely code, and depending on the context of it how "active" really means. It's not a 24/7 job for the architect because of all the other roles and responsibilities, but architects must definitely code.

Presenter: Yeah, I said something at SATURN last year about this, and then George Fairbanks and I argued for the rest of the conference about it. Absolutely. Architects have to write code. That doesn't mean that you are on the critical path necessarily, right? Because a lot of your job as an architect is enabling others, helping them to write code or kind of educating about what the core qualities are and the properties that you want to promote in your system. But you've got to write some code every once in a while just to stay in touch.

Presenter: So in the previous segment we spoke about writing code as a way of helping to make more tangible and communicate the architectural concepts you're espousing, and here I think we're also talking about how you stay connected to the downstream consequences of those choices, and understanding feasibility, understanding what it's going to look like.

Presenter: Correct, and really it's about the craftsmanship and to be able to champion that craftsmanship and be the mentor to the team for the craftsmanship, and if you don't understand the details of it, there is no way you'll be able to advocate them and make the right tradeoffs. And on that subject, there's actually a very good paper in IEEE Software, I believe, by Frank Buschmann, who really touched upon why an architect should code and to what extent. We can give the audience a reference to that. That would actually give some concrete advice there too.

Presenter: So I want to, I don't know, pivot this a little bit. It cuts both ways, right? So yeah, an architect needs to code, but an architect also needs to be aware of the business side. So you can't ignore-- as annoying as the marketing guys might be sometimes, you can't just ignore them, right? You have to understand the market, you have to understand your users. Maybe you should read the marketing plan. Right? It's not just the technology side of things, but you kind of have to have your hands dirty in both of these areas. And from a communication perspective, right? So we're talking about technical debt. When I'm talking about it with my team from an engineering perspective, everybody usually gets it, but then there's that other side of the conversation, talking to my product manager. He wants Feature X. "Eh, you can't get that." Right? "I need to do some technical debt

pay-down." Right? How can I communicate with that person in a way so that they understand the value of making this investment?

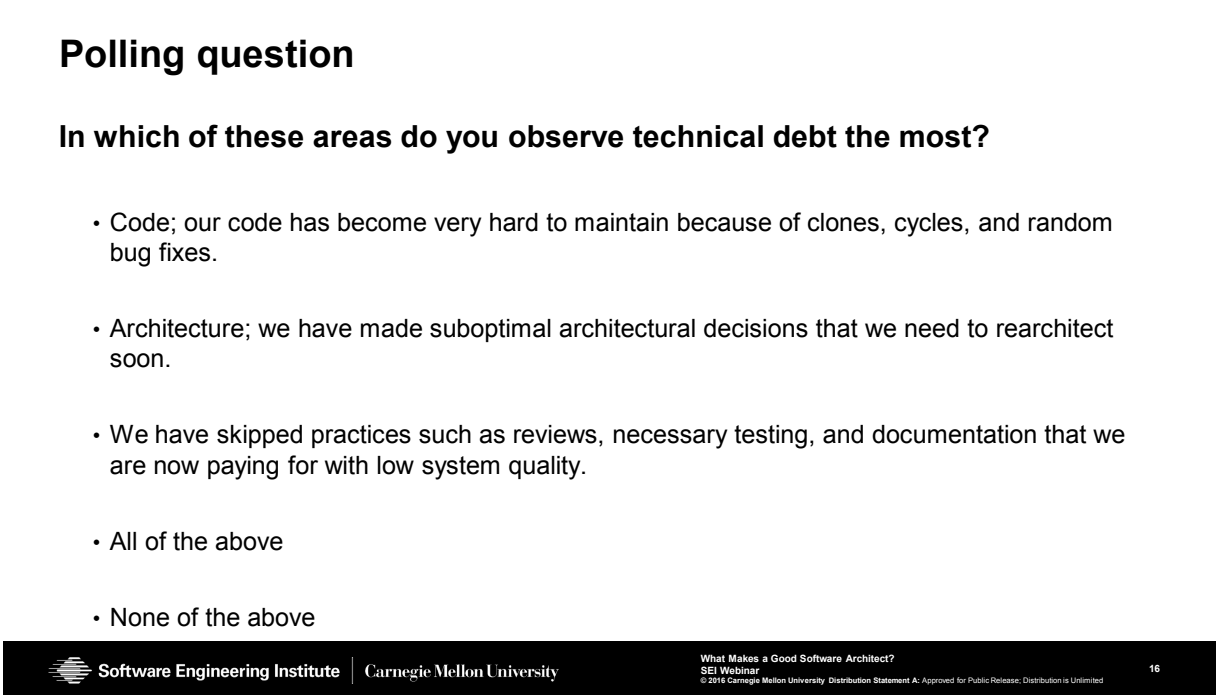
Presenter: So perhaps it makes it more tangible for you in the role of architect to have that experience in the code and being able to articulate what technical debt really truly means in an operational sense, not just a conceptual sense.

Presenter: Yeah, I mean, my point is that you have to be able to speak to your audience, right? And an architect has many different people in the audience that they need to be able to talk to. So code and technology is one, but there's all these other things around business and value on the other side of this.

Presenter: And when it comes to code as well, to bring it back to the architectural skills and technical debt and coding, there are lots of issues that might creep into the code, for example, with the skill sets of the developers. That's the team that you get assigned with, and to be able to understand where those maybe unintentional issues might be creeping into your system early on, you really need to be able to understand it, reflect to it, and interfere at the right time to be able to make sure that it doesn't creep in and accumulate.

Presenter: This looks like a nice lead-in to the next polling question we have.

Polling question



**016 Presenter: Okay. So let's get that queued up here. And that question is: In which of these areas do you observe technical debt the most? In your code; in your architecture; we have skipped practices such as reviews, necessary testing and documentation; all of the above; or none of the above. So we'll give you about 15 or 20 seconds to vote on there. Will?

Presenter: So this is really about different manifestations of this concept we're talking about here, and if I could ask you to kind of contrast the different ways that you see it.

Presenter: So we've reflected that more and more we see the critical technical debt to be architectural. We also-- the way we work on this and the reason that we're excited

with our team at the Software Engineering itute with a lot of the colleagues out there that we work with about technical debt on, is because it can be strategic and conscious. Unfortunately, the reality of the software development today is we see a lot of unintentional technical debt creeps in, and that's where the discussion, "Well, is it code, is it architecture, is it my manager that brings the technical data into the system?" comes in. So that's probably one thing, to make sure that the teams really clarify and trace the roots of technical debt to their system rather than talking about it only at the abstract level. So that's one of the experiences that I can offer based on our work.

Presenter: So do we want to take the poll results?

Presenter: We do. So we had 4 percent none of the above, 62 percent all of the above, 14 percent have skipped the practices, 7 percent with architecture-- I'm sorry, 7 percent with code, 14 percent with architecture, 12 skipped the practices, 62 percent all of the above, and 4 none of the above.

Presenter: Surprising, or is that what you would expect?

Presenter: Well, I don't know if you recall, we were discussing about this a couple days ago and I said that I predict that we'll get all of the above, and that seems to be consistent. Yes, it's all of the above because I don't think currently teams have the

right skill set to balance it off and they're not sure where it creeps in, and that's where it becomes really important that there's ownership about the skill set of understanding those architectural tradeoffs, and making sure that you separate-- well, it's not just skipped practices that creeps the technical debt. If you have skipped practices, if you're not doing code reviews, if you're not training your developers, just do it. Calling it technical debt is not going to help, and I think that resonates with Michael's team's experience as well.

Presenter: Yeah. I mean, I'm a little disappointed that anybody unfortunately is skipping practices. It's like this idea of essential-- or what is it?-- accidental and essential complexity. Like there's these things that you just should do. Like there's no excuses for not doing reviews, right? Skip a review, it's going to bite you later, right?

Presenter: So it's the old adage.

Presenter: Those things we have control over, is my point. Like we should-- any kind of process thing, any kind of methodology type of thing, we should be able to do that kind of stuff. Those top two I think are interesting because there's a whole bunch of things that, "Well, it seemed like a good idea at the time," or maybe it was the right idea at the time. Right? Like you chose to take a shortcut maybe to get a time to market-- faster time to market or

something like that. Or maybe you didn't know what the right answer was and you've got to ship something, so you develop a hypothesis of what you think is the right design moving forward and you go with that and then you learn. So those things I think we can't avoid, but the practices we definitely should.

Presenter: So it's the old adage in terms of skipping practices. We never have time to do it correctly, but we find time to do it over.

Presenter: Yeah.

Presenter: If you're in that situation, technical debt is going to mount. So let's talk a little bit about how it manifests. I think you have a next slide that talks about this a bit.

Technical Debt is Not Simply Bad Quality

Technical Debt is Not Simply Bad Quality

"we have the source code static analysis tools, but this is to assure proper quality of source code. But how architectural changes are impacting I don't know."

Original interpretations of technical debt led us to think it is bad code quality.

- Low internal code quality is a problem, but claiming it as technical debt should not and does not legitimize it!

**017 Presenter: So the initial interpretations of technical debt led a lot of the development teams to believe that it's really about code quality, and if you get ahead of code quality, you'll get out ahead of technical debt, which there's a significant aspect of it that might be true if you really have this lack of software craftsmanship, there's lack of skill set, but it's really-- the code quality by itself is not enough, and today there are quite a lot of tools that help you with code quality. Use them as they are fit, and make sure that you don't inject those kinds of issues into your system. That's really bread and butter, and I think if there are-- teams are finding that they are struggling with it, it's I think a different problem. It's really where you cannot really use those tools to be able to detect the architectural

issues, where some of them might still be related to the way you actually maybe structuring the system, the modifiability and maintainability of the system. And again, some of those tools could actually start helping you, but someone needs to be able to understand how you're going to instrument those tools and what you're going to get out of them, because there is no solution out there that's going to help you answer those questions just by the click of a button. So that's-- and we have evidence over evidence over evidence that confirms this, both from research as well as practice.

Presenter: No silver bullets.

Presenter: No silver bullets.

Presenter: So if you think about agile development processes and the desire to quickly learn from the experience of short iterations, how does that play out vis-a-vis this view that technical debt goes beyond just what you can see in the code? If you could comment on that please.

Presenter: Right. So the main thing, bugs are not technical debt, right? Bugs are bugs. Technical debt-- I mean, really the way it manifests is your ability to deliver. With agile processes-- I don't want to pick on agile, I guess, because it has nothing to do with agile. It's really when you're making decisions in uncertainty. It's very-- let's see here it's very easy for you to deliver some kind of outward, externally visible

kind of value in a kind of bad way, in a suboptimal way, and backward and forward, how do you understand this--

Presenter: So I guess where I was going was success at agile requires you to be conscious of technical debt and if you're not conscious of technical debt, you're just getting in trouble faster.

Presenter: Well, I mean, with those things it's-- as Ipek was kind of mentioning, your number one weapon against these things is really caring about what you're doing. Right? Having a strong sense of craftsmanship. Even if you have delivered the wrong thing-- which hopefully you don't, right?-- hopefully the code is still good. Right? Like you didn't just write terrible code to do it. But obviously there's a balancing act, because you don't want to invest too much in something that is potentially not useful. I don't know, it turns into a big swirl, I guess, but.

Presenter: And it's critical to bring it back to the architect's skill level, because as I think my colleagues John and Andrew commented out there is the designer, the extender, and the sustainer role, and those are really being able to understand the needs of the system today versus tomorrow. So evolution will creep in. Quality and maintainability will creep in. But there are these systematic issues that across the system might actually be starting early on, and it's really architect's role to be able to

understand them, because it really comes down to the refactoring and rework, and there is the rework that you can today understand by looking at the system, but there's also the anticipated rework. "If I don't do it today, what am I going to get myself into? If I don't upgrade the technology today, what will happen? If I don't really make these changes that keep slowing us down, what will happen?" Those are really skills, someone who has the ability to abstract the system's changes, who has the ability to understand some of the key architectural patterns and design choices that were made, and their impact not only on the single files that you might be looking into, the code quality, but structurally across the system what needs to change when I start making those changes, and that's what we're trying to communicate. Simply if you just feel that you have bad code quality, you probably have a lot of other issues that you need to deal with. Technical debt, that is not just one of them. Software architecture is not just one of them.

Presenter: It's a big iceberg.

Presenter: It's a big iceberg.

Presenter: I just want to throw in, this-- you can reason about them as well. Right? So technical debt is not always this bad thing you're not the victim, right? This is something that you can choose to do. You can choose to invest in certain ways. I think there's some work by Barry

Boehm, and I want to say there's a curve, and I'm not remembering what it is, but basically you can estimate what's the cost of me delivering something sooner versus later versus the cost of me taking more time to design it, doing more up front, and kind of-- I don't remember what the axes are-- of course, because we're in front of tons of people here-- but the curve kind of dips at some point and you're aiming for this--

Presenter: So U-curve optimization problem.

Presenter: Right, exactly. You're aiming for kind of this lowest point of how much you invest versus how much-- your ability to deliver. So you can think about that. You can reason about it. You can estimate. And sometimes rework we generally think is bad, unless you get a lot of value from that, and if you're able to ship sooner-- right? So something that I like to talk about is, "Oh, that's a problem I'd love to have." Right? So, "Oh no, our system's not going to be able to scale to a million concurrent users." Right? Awesome. We've got a million concurrent users. Let's deal with that, because right now we've got two. Right? Let's get two users concurrently, and then we can deal with the scalability issues later. Right? So when you're kind of looking at the world from this "You ain't gonna need it" kind of perspective, or build it just in time when you do need something, now you're starting to look strategically at

these ideas. Right? And so if you've built a system that isn't scalable to a million users, is that necessarily a bad thing, because you shipped and you're getting value today? Right?

Presenter: And so this probably drives further to the benefit of quantifying and to finding ways to monetize some of these things. It really helps you to make strategic choices, along the lines of what you said.

Presenter: Correct. And quantification, as I keep repeating it, it's not just a press of a button. It's really understanding what you're quantifying from a quality attribute perspective as well as business perspective and how it maps to your system. That's really that balance. And in earlier days of software architecture there were the teachings that we said, "Well, your system has an architecture whether you know it or not." I think that could very well be reflected in technical debt as well. Your system will have technical debt whether you're aware of it or not, because if you're dealing with a successful, long-lived system, you're dealing with change, you're dealing with technology upgrades, you're dealing with developer turnover, you're dealing with software architects who might have or may not have the right skills based on the time of the development. So I think that's how they really relate very well together in design tradeoff analysis.

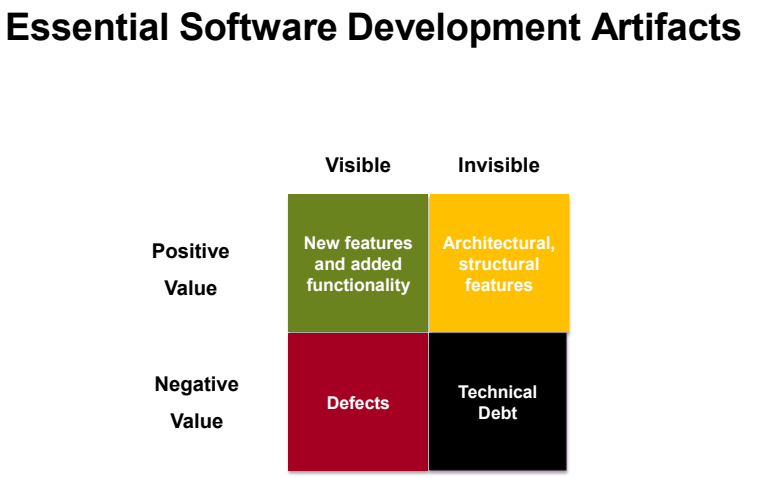
Presenter: Should we take a couple more questions? I think we've probably got a list of them.

Presenter: Yes, lots of good questions coming in. So Nita's asking, "How do you get the balance between growing technical debt and the delivery speed?"

Presenter: Okay. So that I think goes back to whether you understand your business goals and whether you understand what growing technical debt really means. One of the things that we are advocating is, similar to how you define your architecturally significant requirements, your defects or features define your technical debt, and that actually makes a very significant starting point because you can anchor it, and then decide whether it's really tipping off or maybe it's still okay. And one of the other aspects of technical debt, when used right, it could really be a very powerful strategic design tool. So that really comes to the teams to be able to define what that means. I think Michael's example in terms of, "Well, if you're just dealing with two concurrent users versus a million, well, then maybe you can live with some technical debt initially."

Presenter: Do you want to speak a little bit about visibility and different kinds of technical debt? I think you have a good chart on this.

Essential Software Development Artifacts



Kruchten, P. Nord, R.L., Ozkaya, I. 2012. Technical Debt: From Metaphor to Theory and Practice, IEEE Software, 29(6), Nov/Dec 2012.

**018 Presenter: Well, I think one of the things that we're realizing as we think and collect more empirical evidence on technical debt and software architecture is technical debt should be treated as part of a software development artifact, similar to how we treat new features, defects, architecture and the related aspects of it. So initially this chart was put out there by our colleague, Philippe Kruchten, and he calls is "What Color is Your Backlog?" So an ideal software development environment has a balance of these aspects of.... You need to be able to spend time in developing new features. You need to ensure that the architecture supports those new features today and upcoming, and you need to minimize your defects and be able to respond to them timely, and because long-lived

systems go through these evolution cycles, technical debt will creep in as well and you need to be able to balance that too.

Presenter: Yeah, I think this is a cool chart. Right? So "What Color is Your Backlog?" is kind of-- it's a neat idea, right? So you should be-- let's see here. So most of the time, on the visible side of things, we're talking with our product managers about what sorts of things, or we're really thinking about our customers. What we don't always do a good job of-- and those things, under the visible column there, they make their way into the backlog as a very obvious thing that we track, and I think that we've done a very good job as a software community of making that visible and creating artifacts that we can collaborate with among different stakeholders. These invisible things I think are much more interesting, at least from an engineering perspective, and unfortunately though, we don't always do a good job of visualizing them or prioritizing them. And so I really like this idea of kind of zippering in a couple of defects, a couple of features, some architectural investment, some technical debt investment within your backlog and bringing that conversation out into the open. I think that's really the main thing.

Presenter: And what I also like about this chart is, as we said earlier, yeah, you might think your manager, because they're not making the right

decisions at the right time, is the reason of the key technical debt you have. Correct, but at the end of the day it really manifests itself in your system. What you need to change in the system is what technical debt is. It puts in your face that it's a key artifact that you need-- it's a byproduct of system development and as an artifact you need to manage it throughout your spin backlogs, release management systems, whatever process you're using.

Presenter: I think this is a nice elaboration that really comes from that previous question. Maybe there's another question that'll take us down another great path.

Presenter: Okay. So let's go with one from Gerald, asking, "Given that technical debt is an impediment to agility, do you believe that agile methods may actually be a cause of technical debt?"

Presenter: Oh, heavens.

Presenter: All right. I'll take that question, and I'm sure Michael-- there is-- you cannot say agile creates technical debt or waterfall prevents technical debt, and we actually-- one of the things that we're really working very hard is whenever we put something out there, we have empirical evidence, and we find over and over again there is absolutely correlation between one development process versus technical debt. It's really the architectural decisions and

the teams and the system needs that might create or not create technical debt. And if anything, if you're doing agile right, it might help you visualize technical debt earlier than later, because if you're making software architecture a part of your software retrospectives, then you're really uncovering those tradeoff issues. You're talking about, "Is it short-term, is it long-term? Can I live with this? It is priority?" You're really negotiating them. So it's really-- if you're finding yourself dealing with technical debt, before you blame the process, I would really step back and really assess what's going on. So, Michael uses agile day in, day out, so I'll let him comment more on this.

Presenter: Yeah, so it-- agile doesn't create technical debt; you create technical debt. Okay? So you put it in the system--

Presenter: Own in.

Presenter: Yeah, own it. Don't blame the process, right? It's the decisions that you make day to day, and maybe they are the right decisions, and that's fine, right? If anything, agile, when you really embrace the agile principles and the values, it's one of your best ways that you can manage technical debt, right? And some people, I think anyway, are less comfortable with this ambiguity or the uncertainty than others. Yeah, so if anything, agile, it really helps you, and I'm kind of reminded of the-- there's the story of the boiled frogs, or whatever. So this

is-- it doesn't make any sense when you think about it, so let's just go with it, right? If you want to cook a frog, you throw him in a frying pan. First of all, don't cook live frogs, right? But if you want to cook a frog, you put him a frying pan, he hops right out. It's too hot. Right? So instead you put the frog in cold water, turn up the heat. Over time it rises. Next thing you know, your frog is boiled alive, right? Totally dead. Agile practices have these inspect-and-adapt methods built in. It's baked in as a part of the methodology, right? So at no point should you ever find yourself as a boiled frog where suddenly you're overwhelmed by massive amounts of technical debt because hopefully you've been inspecting your processes, inspecting your system as you go, and making conscious decisions to pay down or manage that as you're moving along. So agile doesn't create it, you create it, and if anything, I think agile really helps you to manage it in a much more positive way.

Presenter: So if you do agile well, the feedback you're getting, the pace at which you're getting feedback, may help you to be more proactive more frequently. But if you're doing agile poorly, you're making more mistakes more quickly, and it's not because of agile, it's because of choices you make.

Presenter: Yeah. Now, we want to be careful about pacing and stuff though, right? Because to me, a

week-long or a two-week-long sprint is really good. For other people, that's way too fast. The point is that you're kind of embracing this idea that there is change and you're kind of inspecting and adapting, and that's your process, your practices, but also your system, your software system. So whatever timescale that is.

Presenter: Probably rooted under that question is also agile and software architecture compatibility question, and I'm really hoping we've discussed that enough and we are beyond that. Yeah, there were earlier writings of agile software development process that misled us to think that they were incompatible, but I think today both the agile community as well as the software architecture and the developer community recognize that this is not necessarily a byproduct of the process; it's really a byproduct of how you decide the priority of your backlog, and if the software architect, they are not-- either developers or software architects-- that are really embracing their role well, yeah, correct, your system will suffer, but that's not necessarily the process. It's really the software architects that's probably not putting the right things in the backlog at the right time.

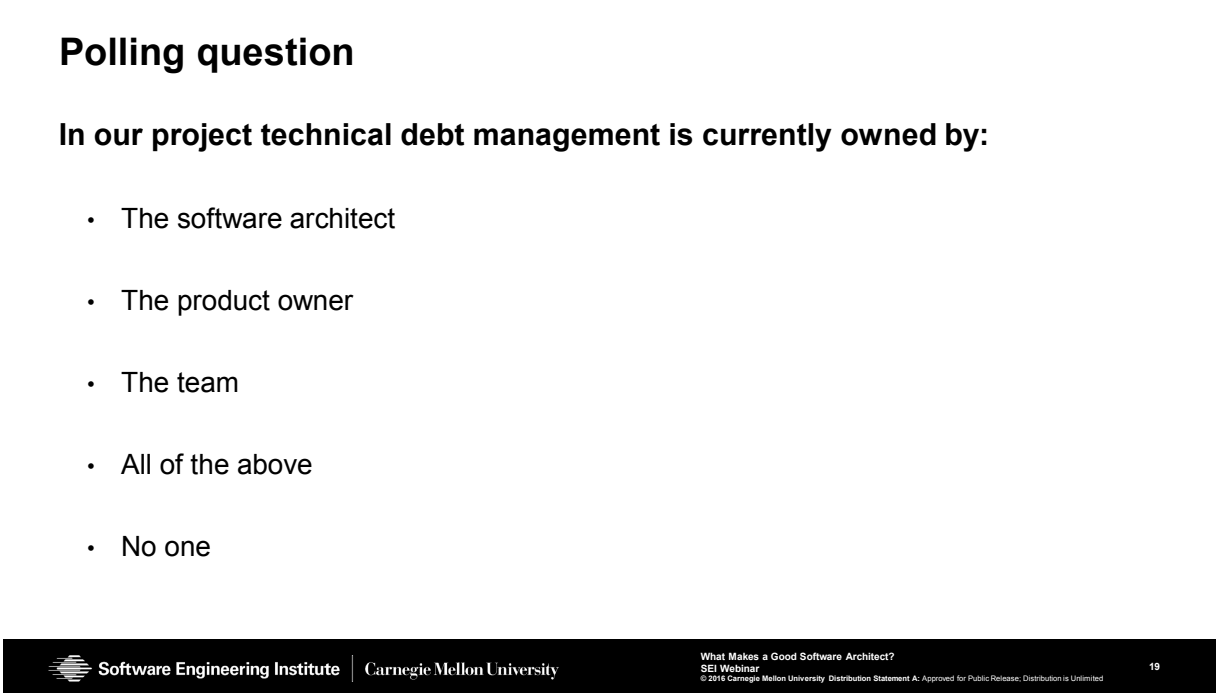
Presenter: So coming back to this nice two-by-two matrix, it's a matter of making more visible the things that have previously been invisible, and irrespective of the methodology you choose.

Presenter: Exactly.

Presenter: Well, we're also growing as a community, right? So I think there's been a lot of work in the patterns community and groups building out frameworks. Like we have more frameworks than we've ever had. They kind of embrace more-- they're more opinionated about how you build systems today. Right? And I think that's also contributing to some of our ability to talk about technical debt and deal with it. So we talk a lot about agile, but modifiability and maintainability, you probably don't actually have scenarios for those. You probably don't ever talk about them with stakeholders. They're just assumed to be true, right? And I think that's wonderful that as a software community we're finally getting to that point where just good craftsmanship and good design is just assumed, right?

Presenter: Correct. And that also goes back to how software architects should be able to understand it, because if I don't understand what the framework-- what quality attributes come with a framework I choose, then I might actually be injecting technical debt that nobody bought into to start with.

Polling question



**019 Presenter: So I think we have a next polling question that really brings this to who's involved and what roles are we going to be relying on. So Shane, if you would?

Presenter: The final polling question for today is asking: "In our project, technical debt management is currently owned by the software architect, the product owner, the team, all of the above, or no one." And we'll give you about 15, 20 seconds to vote there, and if we can work in a quick question from Jim asking, "What are the best tools for smartly identifying and removing inherited technical debt. If you can recommend a tool."

Presenter: Oh, wow. So, you really need to start with the goals of the system. So before the tools, I would

start with my business goals and my quality attribute requirements, and then maybe do an architecture evaluation to start with, and from there on, if you really want to assess the quality of the code and pick the tool that maps your quality attributes. Like for example, if security is really important, then there are lots of tools out there that help you do some security analysis of the implementation aspects of it, versus some of the architecture. So that's the right way to go.

Presenter: So I think what you're saying is it depends on which types of technical debt really matter to you, and then the tools follow.

Presenter: Correct. Correct.

Presenter: Anything from your experience you want to comment on there?

Presenter: Yeah, I mean, just to echo that, with legacy systems especially, you don't actually know why they were built sometimes. So getting some kind of a foundation established of like, "What are the business drivers? What were the actual quality attributes?" Understanding the lay of the land or I guess how you got to here is important, but also understanding kind of where you want to go with the legacy system, and I don't know, from some of the things that I've done in the past, I think tests are probably one of the easy entry points. So you can look at-- I guess

depending on how you define legacy system, there may or may not be any tests to actually show you how it works, and I know that that is something that is easy to quantify, easy to measure; there's lots of tools out there. So that at least gives you a window to kind of peek into what's going on.

Presenter: So if you had a suite of regression tests that have been built up over time, over the life of the system, you could really do an analysis on where has our focus been there.

Presenter: Yeah. I mean, you could-- assuming those tests are written in an understandable way, which sometimes they're not, you could even use that kind of as a second part of your analysis, right? So given your business drivers and your quality attributes, what's reflected in your tests? Do those things actually align? That can give you a bit of a hint at least at where the system really is versus where it needs to be.

Presenter: That's a really clever idea. Thank you for that.

Presenter: All right, and to wrap up our polling question, we had 13 percent that the technical debt is owned by the software architect; 12 percent the product owner; 20 percent the team; 31 percent all of the above; and 25 percent no one.

Presenter: Interesting. So no one owning technical debt. What does that really mean?

Presenter: It will keep creeping in.

Presenter: Find a new job.

Presenter: You know who you are. Is that what you're saying?

Presenter: Oh, that's sad. Yeah.

Presenter: Well, all of the above is probably how it should be, because it's the team's responsibility, and that's-- there are different ways it might creep in and everybody should contribute to visualizing it when they find it. Like especially if you're maintaining a legacy system. You might still have surprises despite all the evaluation and the assessment you did early on. But if it's no one, now is probably time to start discussing how you're going to deal with it, because without ownership, things will not progress.

Presenter: People are busy enough. They're not going to volunteer for stuff.

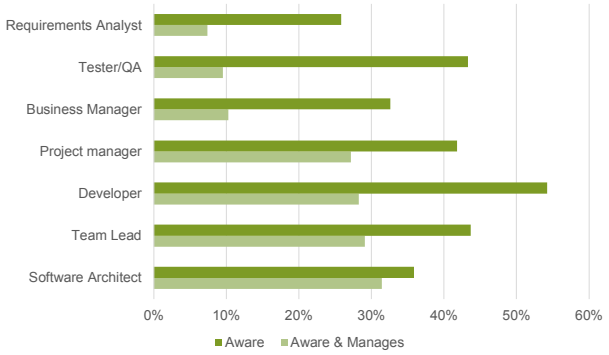
Presenter: Right. And thankfully, with this webinar, whoever you are, you now have some of the information you need to take ownership over the technical debt in your backlog.

Who is Aware and Manages Technical Debt

Who is Aware and Manages Technical Debt

Developers are most aware of technical debt.

While a joint responsibility, software architects are reported to own management of technical debt more often than other roles.



A Field Study of Technical Debt **https://insights.sei.cmu.edu/sei_blog/2015/07/a-field-study-of-technical-debt.html**

**020 Presenter: So, I'll bring some of the-- again, I like tying it back to not "our" data but like "your" data, and this again comes from this survey we did, and we continue to work with organizations and we'll put out them as we do the analysis. But so the questions we ask were two. One is: Who's aware of it? Because you might be aware but you may not have the power to do anything about it. And: Who's aware and manages technical debt? So the way to read the chart is the lighter ones are they're both aware and they're both responsible of managing. So dominantly, developers actually come at the top for being aware, and that's not surprising because day in, day out they work with the system, they suffer from the consequences of the technical debt in the system. On the flipside, although software architects

maybe tend to be less aware of it in a way, they're actually the ones who own it more. They manage it more. So that responsibility rests on both the team lead as well as the software architects. But we kind of have an equal balance of the project manager, developer, team lead and software architect as owning it, and I think that's reflected in the poll results as well. There's a collective ownership of it, but awareness but vary depending on how the developers are working with it. And if no one owns and manages it, that's a problem.

Presenter: Yeah, that reflects our experiences pretty well at my team at IBM. So obviously the engineers are feeling it, right? They feel the pain day to day, and so are much more aware of it. But really it's a strategic between the engineering, the developers, the product owner, and the design team, who kind of represents our user focus. So making strategic decisions between those three is oftentimes how we manage it.

Presenter: And it helps you to avoid being the frog in the pot perhaps.

Presenter: Exactly, yeah.

Presenter: Let's see if we have questions.

Presenter: We got lots of questions. So same thing as with Andrew and John; we'll do rapid-fire

questions. If it's something you don't want to answer, just say pass because we've got so many in here.

Presenter: Lightning round.

Presenter: From Ronnie, asking, "What are some ways an architect can win over their senior developers if they don't agree with the architecture design?"

Presenter: Pass. That's tricky. Everybody's different, right? So you have to get to know that person and you have to understand what motivates them. Right? What are their objections? Can you reason or can you persuade them using logic? Do you need to give something to get something? That's a very individual kind of thing, unfortunately, I think.

Presenter: Yeah. I think
for opinions.

Presenter: There you go.

Presenter: But people can get emotional.

Presenter: Correct. Correct.

Presenter: So data-- if the person, "Oh, I hate Java"-- nothing you say and no data you show is going to change--

Presenter: Correct. But still that makes it a little bit more objective rather than subject if you have that and if there's a reason of it.

Presenter: Get to the root of the problem.

Presenter: Get to the root of the problem, and then personal skills come into play.

Presenter: Convince them that you're in the same boat.

Presenter: Yeah, same team. That's probably the most important thing. Almost remind them.

Presenter: We are in it together.

Presenter: Great.

Presenter: We touched on it, but like I said, so many questions coming in. From Devaya asking, "Planned debt versus unplanned debt. Most of the technical debts starts out as planned debt but it tends to keep accumulating, like if you initially made a different choice and it wasn't important enough then, its importance only continues to decrease compared to newer requirements or opportunities. How do you manage this?"

Presenter: So first of all, if it's really planned, it needs to be well communicated. Because one of the things that seems to be really missing from what we've observed is nobody knows that it was intentional at the time, and if you don't know what was made intentionally at the time then you cannot trace it to the roots and you cannot react to it. So that's number one step that I would take.

And the creeping in is the continuous monitoring and decide what you're monitoring on so that you would be able to bring it-- if you're in an agile team, bring it to your agile retrospectives. If you're in a non-agile team, again, make it part of your software development process to be able to touch upon these and decide at the right time.

Presenter: I just want to comment on this. One of my favorite things for I guess unplanned is the To Do flag. Right? So you have this brilliant idea and you're in the middle of the coding, and then you realize, "Wow, I'm about to go off on some path that we don't need today." Comment "To Do"-- right?-- and then leave a brief description there, and oftentimes you can use the tools-- go back and review it-- you can get a lot of ideas like that for free from the IDE, from the compiler.

Presenter: So maybe there's some sophistication behind what it really means to have it be a planned technical debt.

Presenter: Correct.

Presenter: There's a communication obligation, there's a lifecycle to retiring it or to allowing it to persists. There are decisions that are made later as well.

Presenter: And maybe to wrap it up, every time you think you are making a planned technical debt

decision, you are also making a key software architecture decision.

Presenter: Yep. Nice.

Presenter: That's-- there's no way that it works otherwise. So that's actually really important to recognize, as I said.

Presenter: Great conversation, folks. Thank you very much. This has been great. I'll pass it to Shane.

Presenter: Yes, thank you Ipek, Michael, Will, great facilitation. Thank you very much to John and Andrew as well. That's going to be all the time we have for today for "What Makes a Good Software Architect?" We thank you for joining us. Again, please fill out that survey upon exiting the event, as your feedback is always greatly appreciated, and we hope to see you at SATURN 2016 in San Diego to continue this conversation. Have a great day everyone.

SEI WEBINAR SERIES | Keeping you informed of the latest solutions