
@PLAINPROGRAMMER #OREILLYSACON
BEYOND ACCIDENTAL ARCHITECTURE

Welcome, and thank you for coming to my talk this morning. Today we will be discussing a bit about how to move our teams beyond accidental architecture.

By way of introductions, I am James Thompson. I am a Principal Software Engineer for a company called Mavenlink. We build project management software for professional services company and we are hiring for all skill levels in San Francisco and Salt Lake City.

You can find me online fairly easily. So, please feel free to reach out if you have any questions, feedback, or the like to share.

Before we get too far along I want to make sure everyone knows what we will be covering here and how we're going to work our way through it. First, the goal and purpose I have is to improve our ability to cope with architectural change and encourage on-going intentionality.

[CLICK] We will start with some definitions around the subject, including defining what I mean when I say Accidental Architecture.
[CLICK] We will also explore ways that I think the role of the Software Architect can be misunderstood and rendered less effective.
[CLICK] Then I will try to provide a high level way for us to realign our thinking about the role of the architect.
[CLICK] Then we'll delve into approaches to achieving the realignment practically and specifically examine how they serve the goal of improving our ability to cope with architectural change and maintaining intentionality.
[CLICK] Finally, we'll conclude with some restatement and some recommended resources.

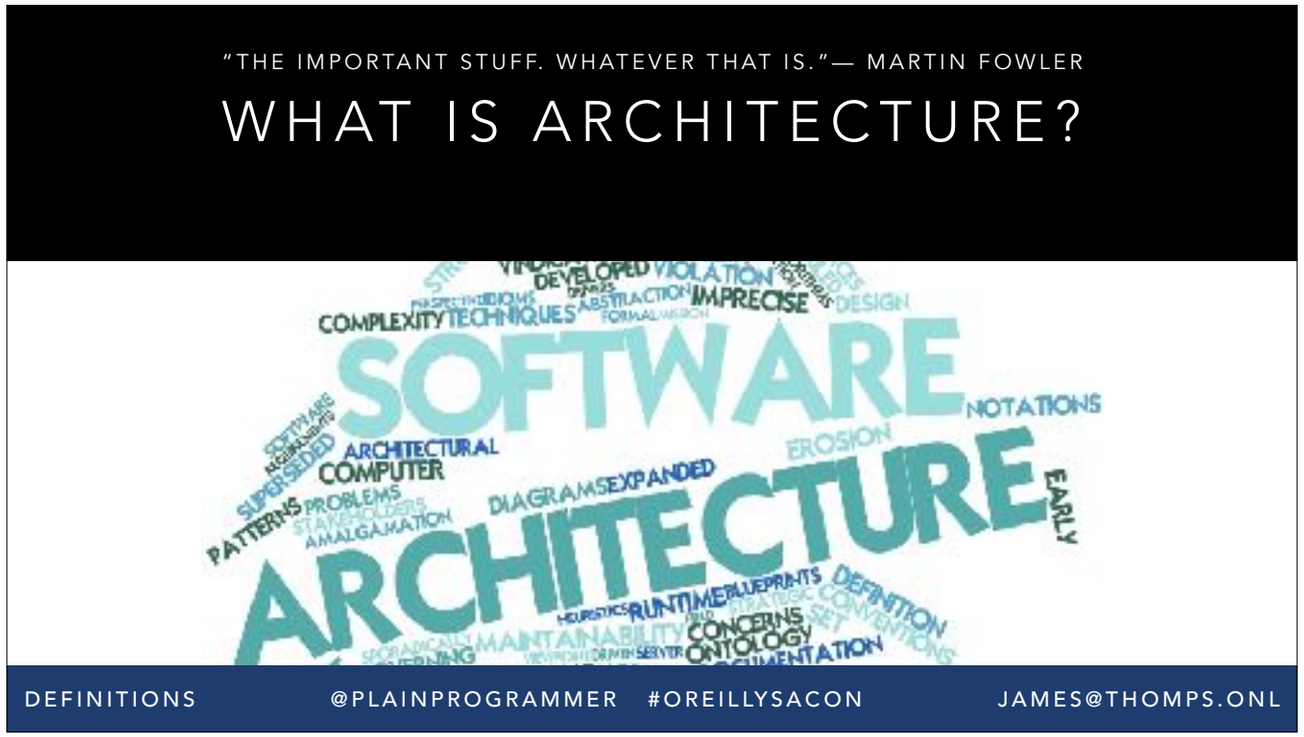
IF YOU HAVE QUESTIONS

Also, so that everyone has appropriate expectations. I am not going to be doing a Q&A portion as part of the presentation. I will be available following the presentation for anyone who wants to discuss, but I will have a handful of pauses in the presentation where I'll encourage us all to engage with one another about the material and give some provide breaks to process what I'm presenting.

You are all invited to reach out to me via email and Twitter, and my details will be placed along the bottom of the slides, so as you have thoughts you can send them along and I'll do my best to get back to you promptly.


INTRODUCE
YOURSELVES
02:00

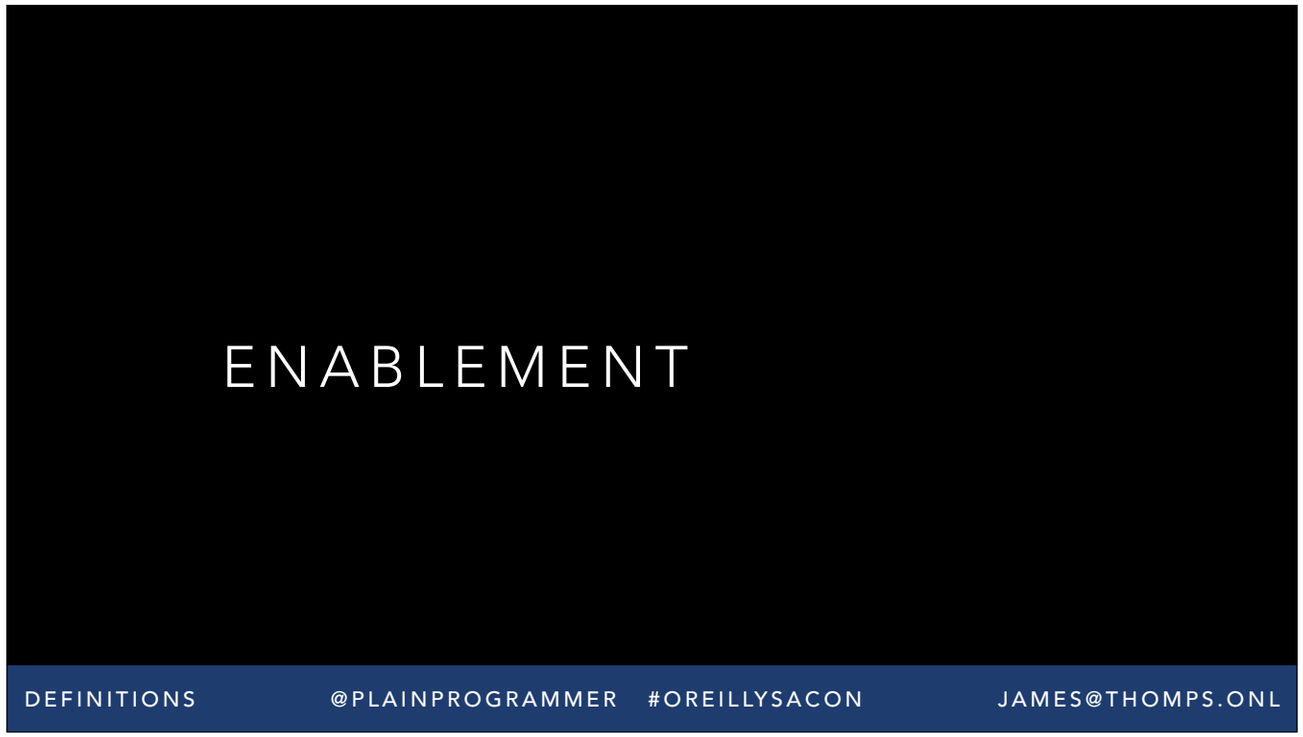
DEFINITIONS



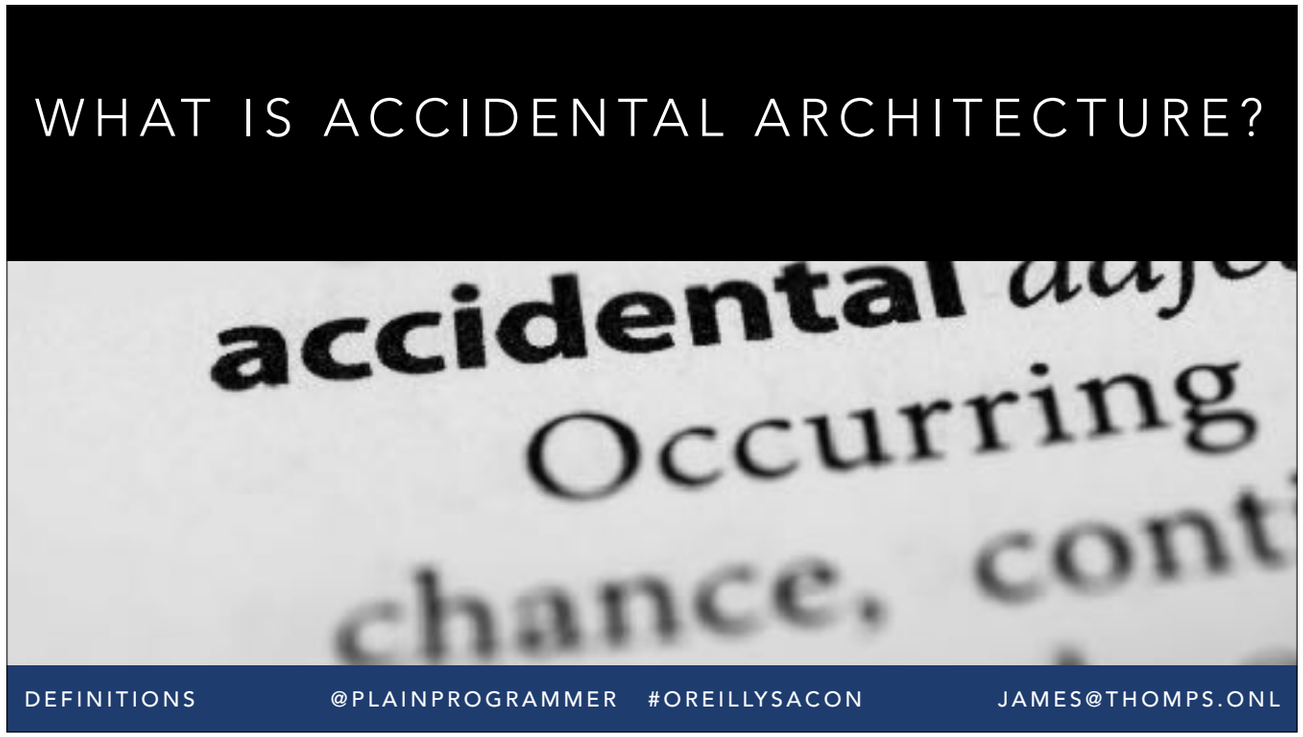
Software Architecture has a number of definitions floating around, and I'm not going to make a meaningful dent in that space. But, I do want to mentions the elements that I think are most relevant for my topic as it relates to what the important stuff is, as Martin Fowler put it several years ago.

First, I think Software Architecture must be tied to the purpose of a system, the why of its existence. A system that lacks purpose will always be in some state of decay or disarray. So, I think it is important that we recognize that Software Architecture, and the role of the architect is concerned with defining, promoting, and perpetuating the purpose of a given system. And, that means that the architect has a key role to play if and when the purpose needs to change.

Second, I think Software Architecture must be informed by and help communicate the fundamental constraints inherent to the system. This is where a lot of the translation from business to engineering takes place, and where trade-offs are most important to be aware of and managed. The architect has the burden and responsibility to discover, communicate and help their teams abide by whatever constraints are necessary, and sometimes to seek to change those constraints where possible.



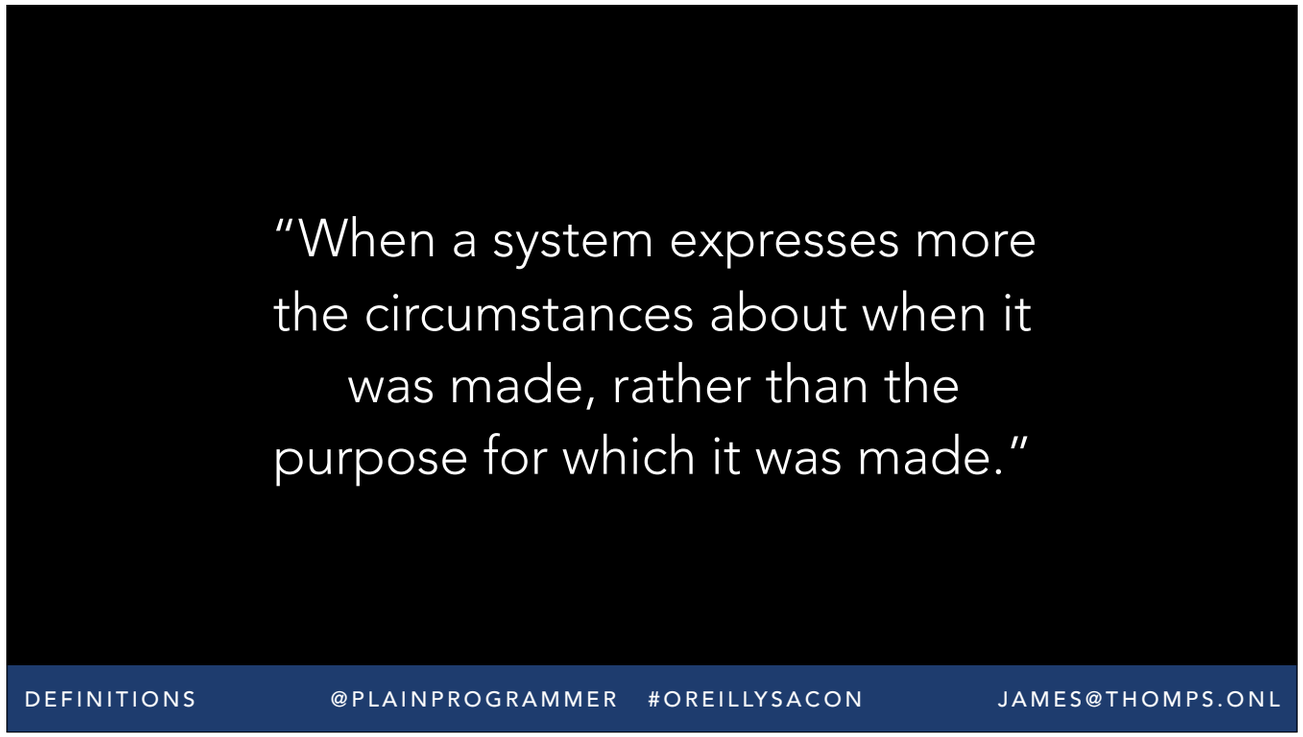
Finally, I think Software Architecture is fundamentally about enabling engineers to do their best work in an efficient and thoughtful way. This gets to the place of the architect in removing constraints where possible, but also to the place the architect has in not being a bottleneck themselves. If the architect holds certain things too tightly they can stifle their projects in ways that hurt everyone. So, the architect must be aware that they are in a position to do both great harm and great good to the systems they oversee.



Every system has an architecture.

But, what distinguishes an accidental architecture?

I think the primary criteria is the influence of circumstances winning out over intentionality. Every project tends to start with a certain amount of clarity and, as circumstances intrude that clarity can be undermined. And, if it is undermined to a certain point the architecture becomes accidental, as opposed to intentional.



Many, if not most, software systems will have some elements of Accidental Architecture just by virtue of how preferences, technologies and other facets of development change over time. But, when those incidental choices define how your teams work with and maintain the software as much or more than the purpose of the system as a whole, you’ve strayed into Accidental Architecture.

I will share with you some examples next.

Many accidents of architecture emerge when constraints prevail over the expression of the system’s purpose.

In a monolith, I’ve seen this happen when time constraints force a team to follow too closely the patterns of their framework, specifically Ruby on Rails, and neglect to consider more appropriate ways to separate the concerns of business logic from such things as persistence. Ruby on Rails makes it easy to concentrate logic in place that are convenient, but not desirable for long-term maintainability.

In a microservice ecosystem this same tendency can be expressed with the perpetuation of God Services that tend to gather lots of behavior to themselves because consolidation is easy, or we build services that are too tightly coupled to each other.

But, Accidental Architecture can also emerge when we lack strong constraining influences like time pressures.

We can prematurely abstract things and produce overwrought, over-engineered solutions where the abstractions and complexity we create are bake full of assumptions that can't yet be responsibly justified.

I've seen this happen when we assume that just one abstraction can meet too wide a variety of needs, and the complexity is turned into ridiculously detailed configuration. The wrong abstraction can be just as, if not more, expensive in comparison to duplication or no abstraction. So, it is important to not let a lack of constraints lead us to introduce more complexity than we need.

Let’s take another brief time to interact and share with each other where we’ve seen accidental architectures arise in the systems we’ve worked on.

MISUNDERSTANDINGS

Authority is something that I think a lot of architects misunderstand, both in terms of its basis and its ability to work. Architects who see themselves as authorities engage in all sorts of gatekeeping behavior like enforcement of style guides, injecting themselves into code review and approval processes, and seeking to exercise heavy oversight. But, authority depends on credibility, otherwise what you have is authoritarianism or monarchism, not legitimate authority.

Here's are some questions to consider:

* For those that are architects by title, how many DO NOT write code regularly?
* For those aspiring to be architects, how many see it as an opportunity to move away from feature work?

In both of those cases, I want to propose that you're not in, or seeking to be in a very credible position. I've worked with architects who have divorced themselves, effectively, from the work of the average programmer in their organization. Know how much I care about their opinions about how I build software? **Not much at all.** Because they lack context and credibility. Their ideas are nothing more than ideas. They're not regularly getting into the nitty gritty of day-to-day feature development, so I have a hard time taking them seriously.

The overzealous use of authority is a great way to alienate and demoralize your team. So, be careful. Too often I think architects grossly misunderstand the nature and proper use of authority in their roles.

But there is a place for authority in the role of the architect, and I promise we will come back to this in a positive sense when we get to our approaches.

Autonomy is another key facet of the architects role that I think is very misunderstood. It can be used as a way to focus on low-priority and low-value, but personally interesting endeavors.

Tech Debt is real. But, most of what we think of as technical debt is just code we don't like. Unless there is a describable cost to whatever the concern is, it's not debt. It may not be modern, or even clear code. But, it's only debt if there is a cost to it. This is where have architects paired with a product manager to act as a check on their perceptions can be very useful.

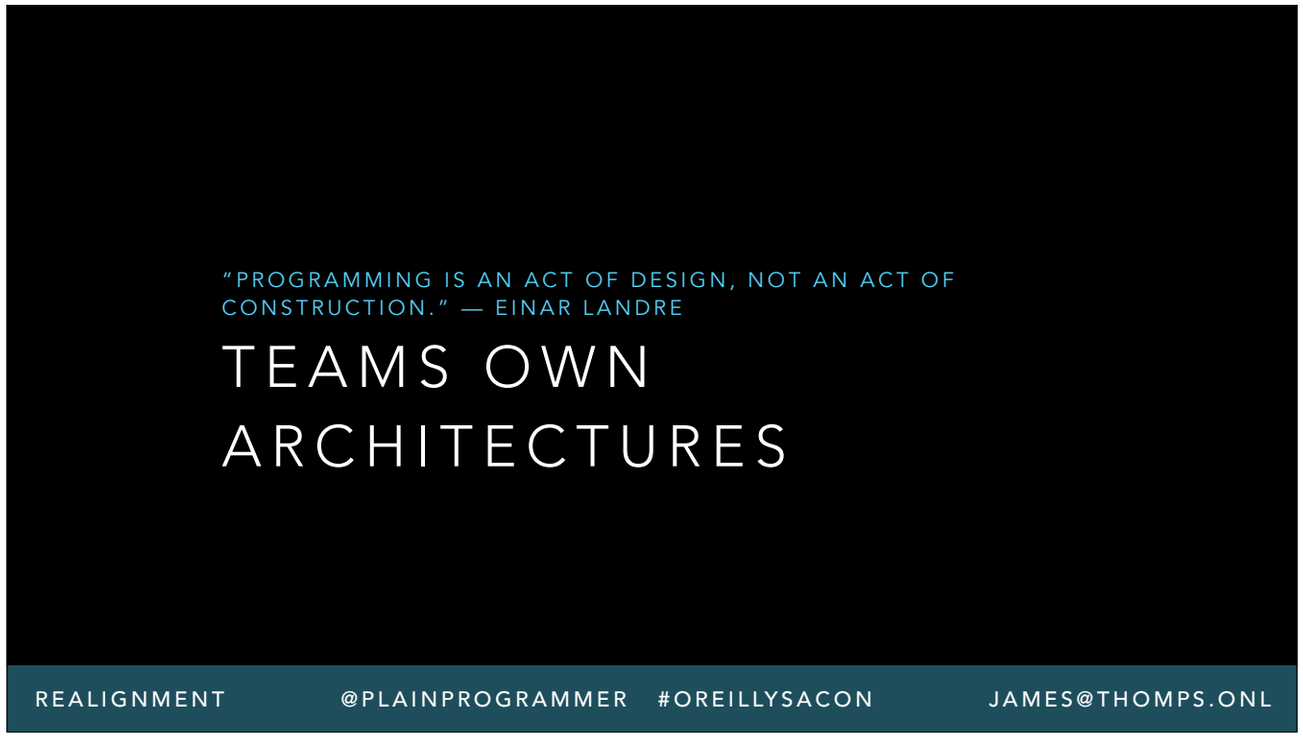
Some architects love exploring and experimenting with cool new tech. But, unless there is a business need that tech is intended to meet, the architect may well just be making up work to look busy. Exploration can be valuable when it is time-boxed, or when it has a clear goal. But, without at least one of those, it's just a hobby.

And, I there are things like pet projects, which I must confess I am guilty of indulging in. I see something I don't like, or that I think could be better. I don't have a business reason to pursue it. But, I do it anyways. Pet projects feed our egos when they go well, and they rob us of time when they drag on, or fail.

I've heard too many architects express a desire for autonomy so they can pursue initiatives like these. I've even heard for teams being formed to pursue vague purposes that will look good on someone's resume, but aren't likely to provide any real value anytime soon, nor demonstrate a responsible return on the investment of time and energy.

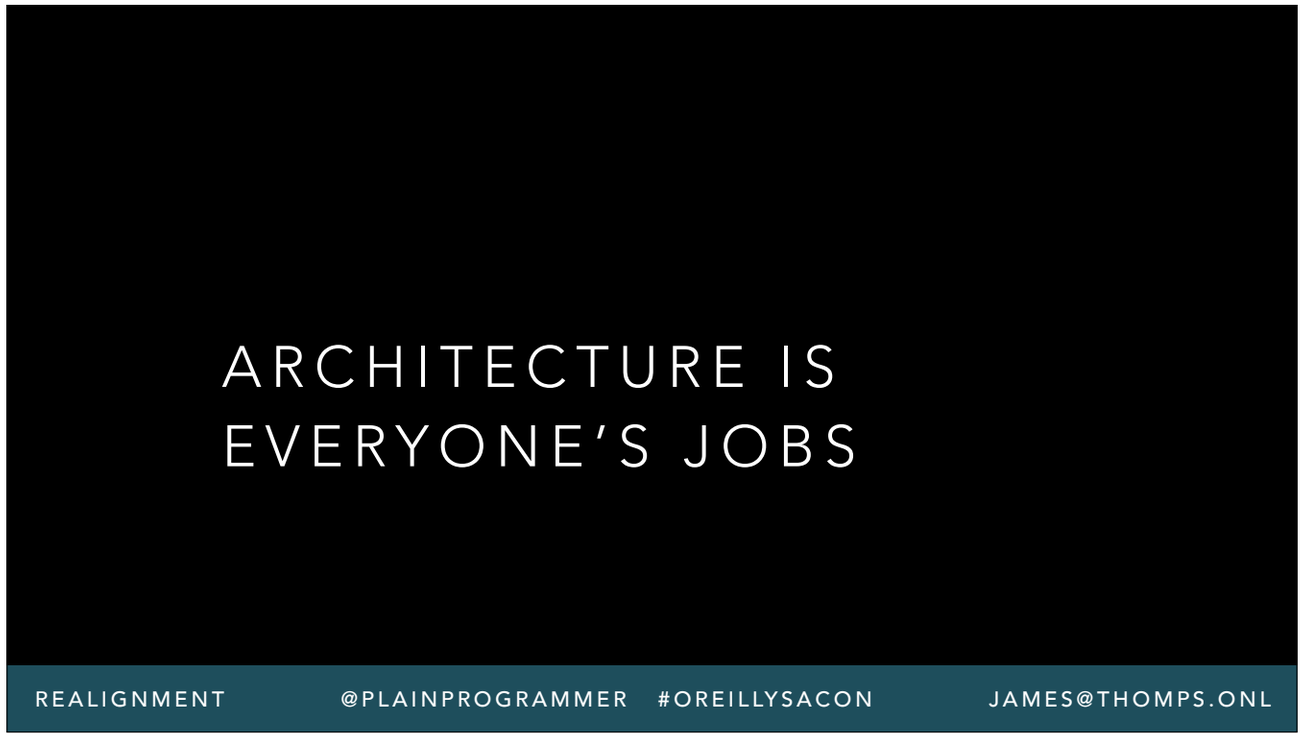
This is why I actually think that software architects need to have their autonomy and independence constrained, otherwise their endeavors can become expensive wastes of time.

Now, I want to move on to quickly cover some ground regarding how I think we can realign our thinking about architecture. I want us to think about ownership, responsibility, and the relationship of the architect to the engineering team.

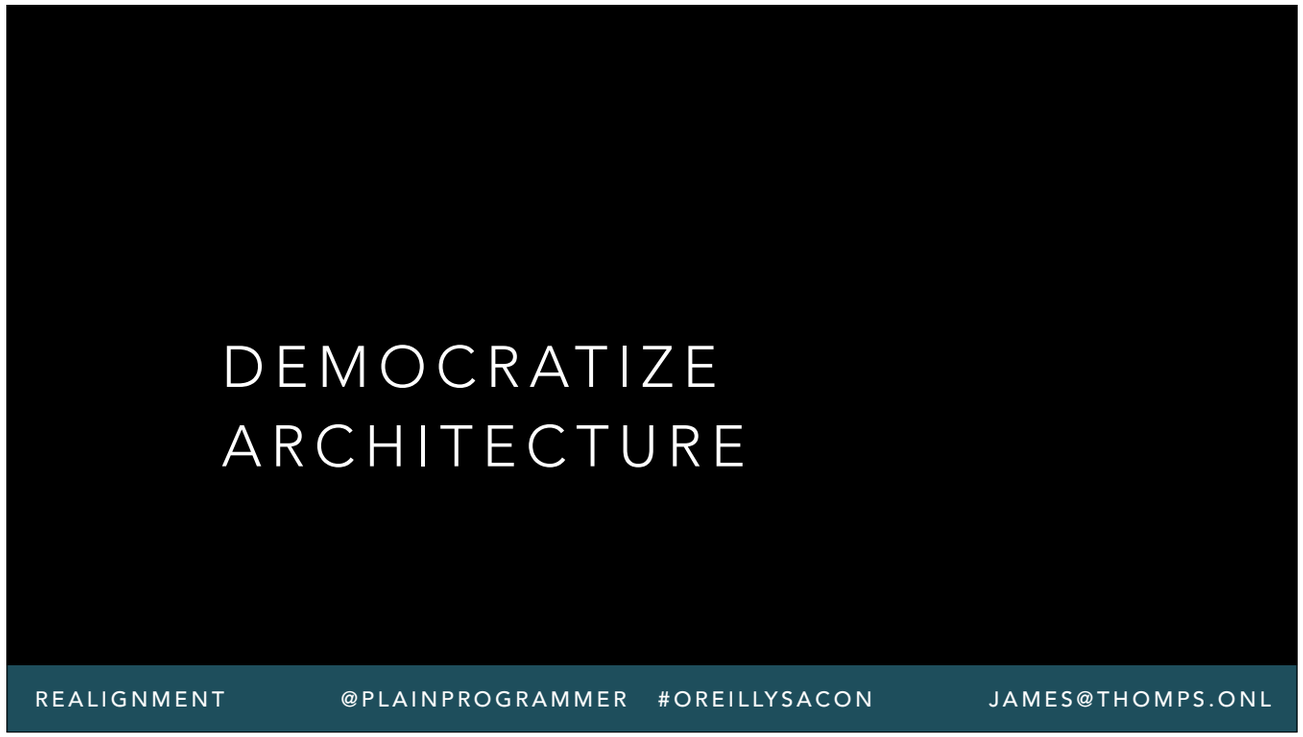


First, I believe that teams own architectures. When a team is building a piece of software the architecture is a collective responsibility and that means it is collectively owned. This is because programming is not construction, it is design. Diagrams showing how software is conceived in the abstract can not deliver value. So, what we readily identify as design is only the crudest and most abstract expression of it. The work of actually writing the code that will eventually be executed in some fashion is the actual blueprint and design of our systems. And, the team is made up of the developers who produce that, so they are the truest owners of the architecture, because they are the ones actually drawing the blueprints, so to speak.

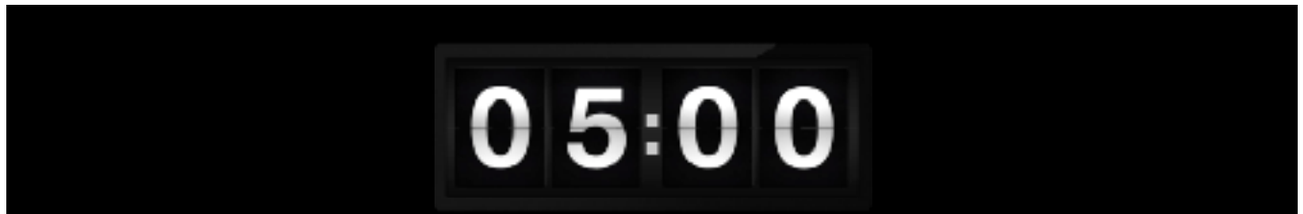
Since architecture in its most expressive form happens when the teams write the code, that architect should be there with them. This helps establish credibility, grounds the architect's ideas in the same context that they are designing for, and puts them in a better position from which to adapt their ideas to the realities of their projects.



If the code is the architecture, and I believe it is, then the whole team needs to be equipped to do architectural work. This is part education, part demonstration, and wholly collaborative. Architects should be leading teams to understand the same things that they do. Architects should not be silos of knowledge or experience, but rather distributors and educators. The best thing I believe an architect can do, is to equip their team to do the work of architecture.



What I am advocating is that we should seek to democratize the work of software architecture. I think we should endeavor to make the tasks of software architecture accessible and actionable by all developers. This is how I believe we can more thoroughly empower developers, and rightly exercise the authority that tends to come with the role of software architect. This doesn’t make the architect irrelevant, it makes them an empowering force and can greatly increases their value to an organization.

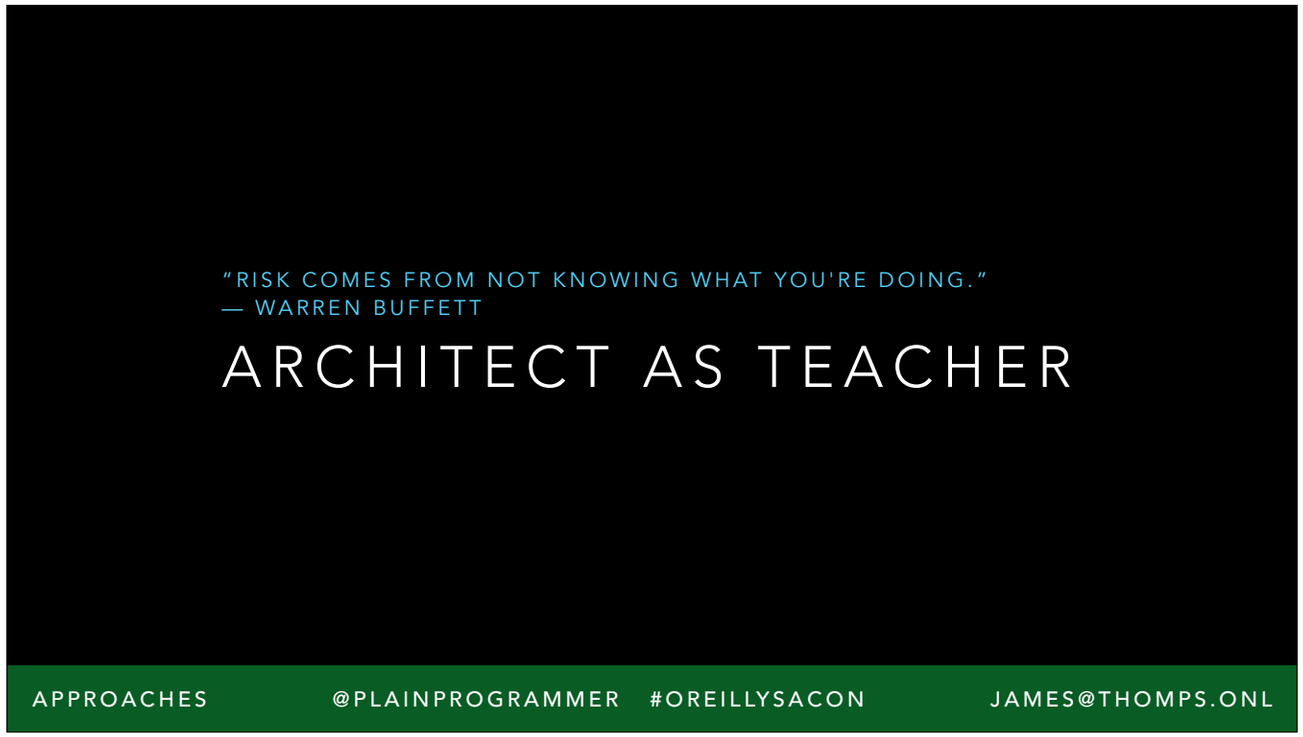


INTERACT

DO YOU THINK DEMOCRATIZING ARCHITECTURE HAS ANY MERIT?

DO YOU AGREE OR DISAGREE WITH THE AREAS OF MISUNDERSTANDING (AUTHORITY & AUTONOMY)?

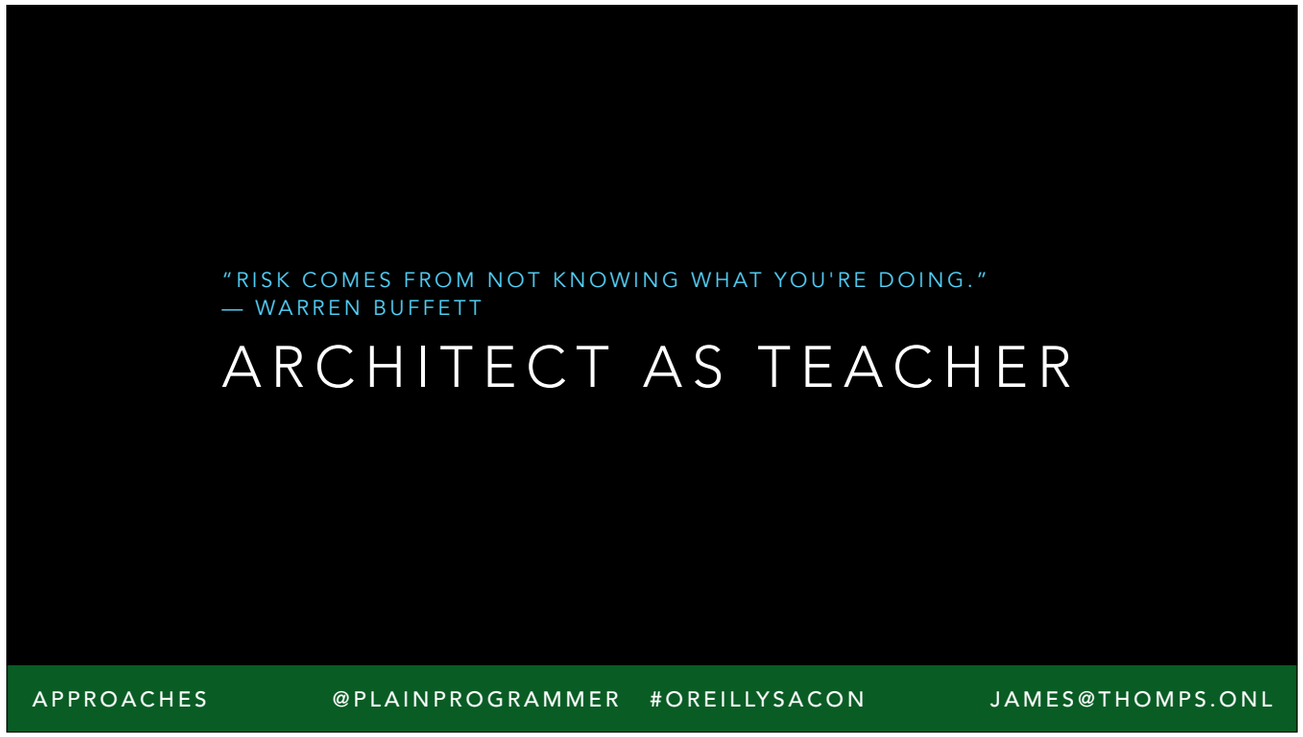
Now I want to endeavor to talk about approaches we can take to help put this realignment into action. How can we take the framing of the architect as a means of empowerment and apply it for the benefit of an engineering team? I think there are three related metaphors that will help us put the notion of democratized architecture into practice.



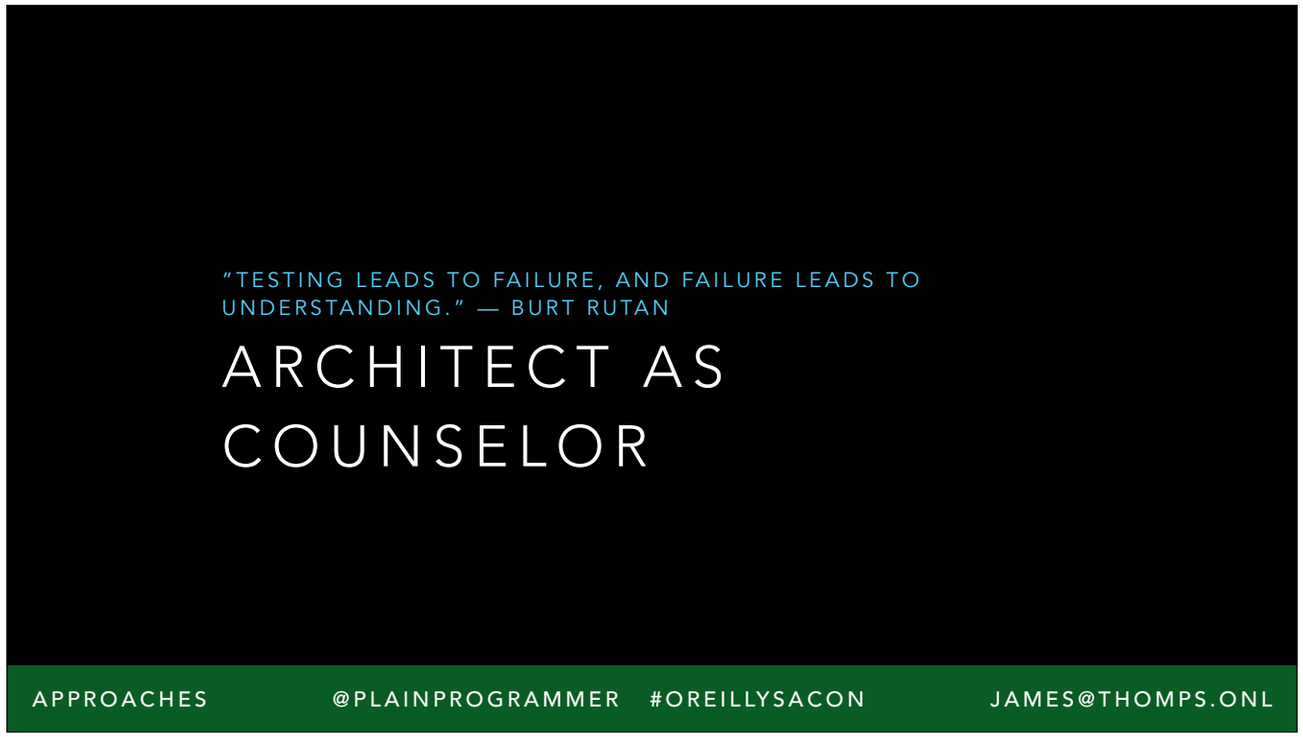
First is the metaphor of the architect as teacher. Architects, owing to their experience and knowledge, are often in a natural position to share both with their teams.

When the architect teaches what they know to their teams, whether through presentations, book clubs, discussions, or just working alongside them; they are helping reduce risk for their team by giving them more context and information to base their actions on.

The more architects are able to help teams know about the problems they are addressing, or the constraints that are required, the better equipped the team is to make good decisions. The architect is uniquely positioned to synthesize the information coming from different areas of the business with their understanding of a system's infrastructure, the potential trade-offs for different approaches, and the overarching technical vision. This allows the architect to more clearly inform their team about the purpose of their work.



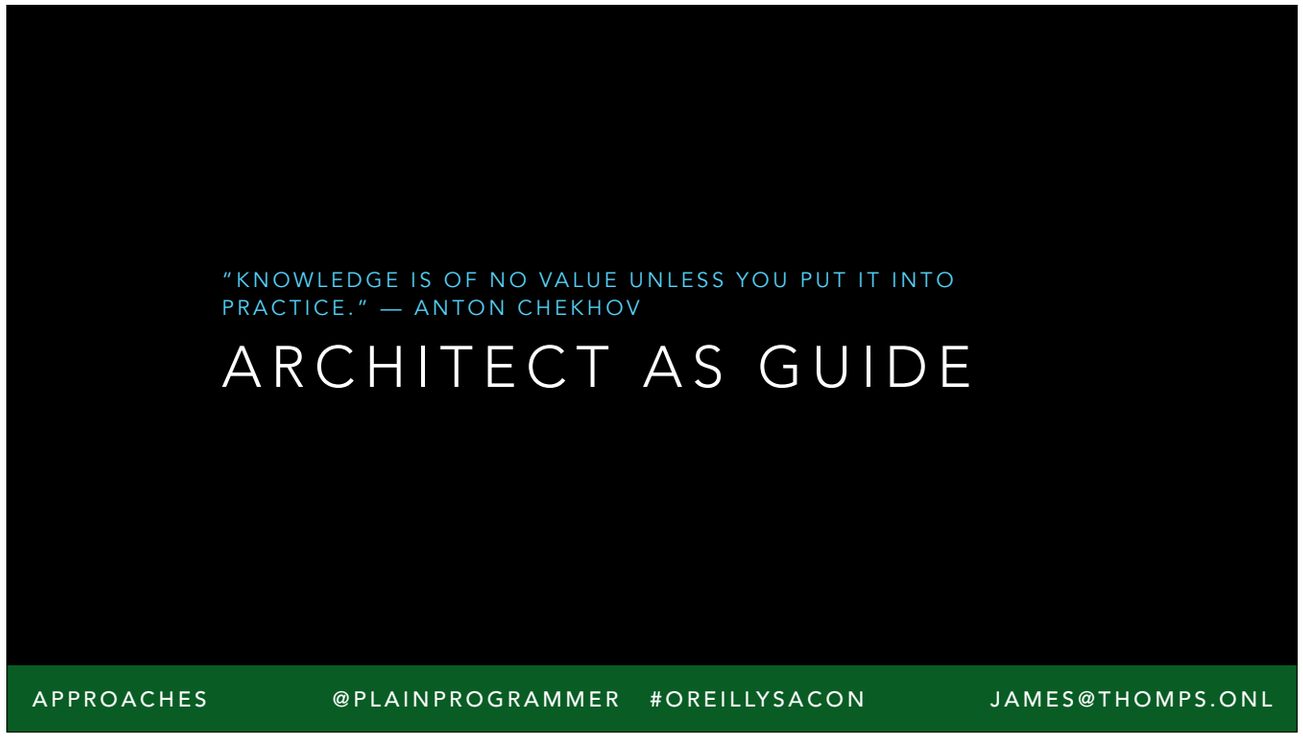
So, I think it is important for us to recognize the unique position the architect has for being able to share knowledge with their team. This helps them enable to team to better manage risk and know the purpose of their work. Both of these convey advantages on the team to make better informed and more sound architectural decisions.



Going beyond the architect as teacher I think the metaphor of an architect as counselor or advisor is also important. Like a teacher, a counselor helps bring together what we might know and synthesize understanding from it. This comes via a few means.

A good counselor challenges our assumptions about what we know, or think we know. An architect can use their experience to challenge the assumptions of their team as they explore their knowledge of the problem space and purpose of the software they build. The architect is thus able to help refine the thought process of their team.

If the architect is embedded in the team, they are in a strong position to provide advice and broaden the things the team considers and they design and implement aspects of the software system. This advice can be anything from recommendations of patterns or practices to consider, to how to effectively test the work being done. The important detail is that the architect is not dictating what the team out to do, but advising them about what they could do. In this way the architect is helping the team discover acceptable solutions, all while learning more along the way.

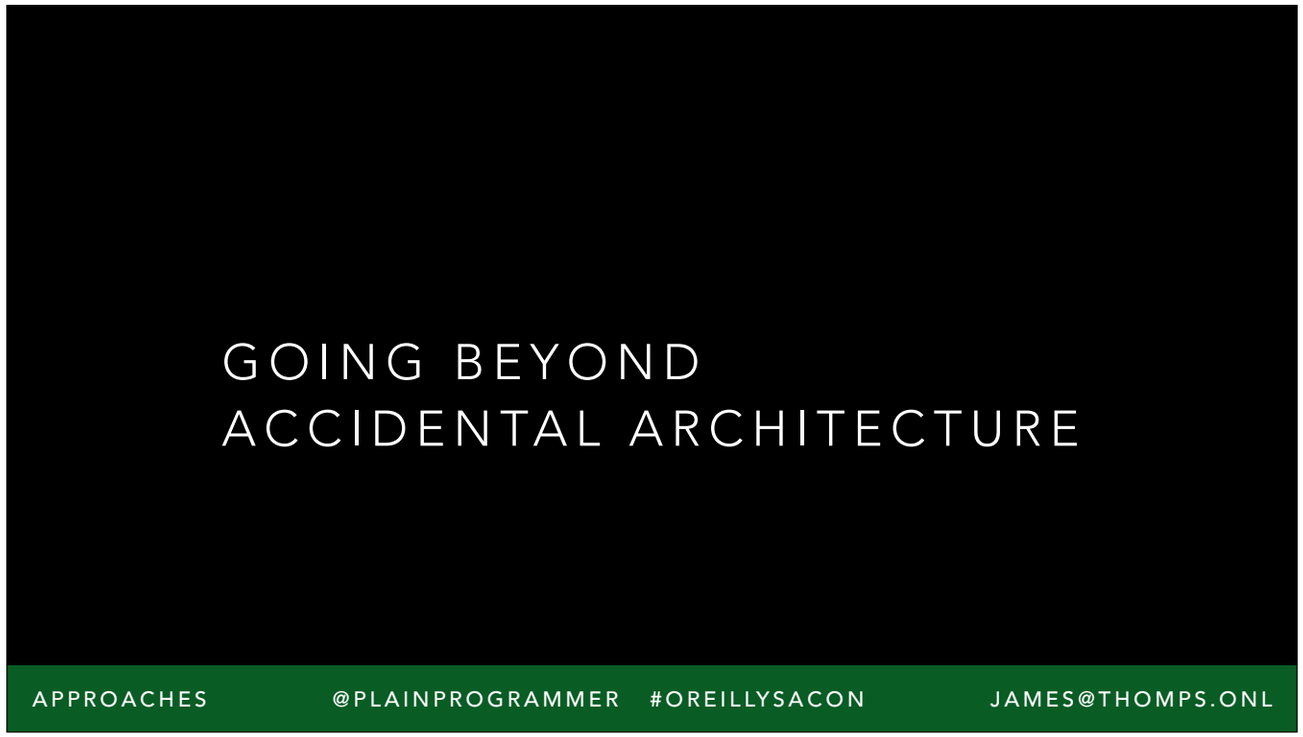


Finally I think perhaps the most important metaphor is that of the architect as a guide. And, I think this is the most powerful since it can subsume the former two approaches almost entirely.

A guide is responsible for showing a path to those who follow them. While on that path the guide explains its significance and warns of its risks. A good guide does not merely hand over a map and a list things to look out for, but rather they go along on the journey. They are fully present and available to not just follow a prescribe course, but also to improvise and elaborate on the experience for all involved. An architect embedded in a team has that very same capacity to help the team adapt to changes in circumstances and to keep their work moving in the face of a changing landscape.

A guide puts themselves into the same circumstances as those that follow them. They accept the same risks as those they lead. This provides the strongest basis for establishing trust and credibility. And, the risks the guide takes are not abstract, they are the same concrete risks as their group. The architect, when embedded in a team, shares the risks of the team and so the team knows that the architect understands and can empathize with them.

And, after a journey is done, the guide and their group arrive together. The entire team succeeds along with the architect who went with them, and they all learn and grow together. The team has gained insight into the architects process and value, and the architect has built trust, credibility, and greater understanding for themselves.



I hope, at this point, it is somewhat clear how I am proposing we can move our teams beyond accidental architecture. If architecture is done by the teams, and if architects are helping to strengthen those teams, then the teams are in a much better place to make more intentional decisions about architecture. They don't need to be told what to do if they've been shown and taught how to arrive and good decisions themselves. And, because the teams have worked directly with the architect, there is a basis for trust and cooperation that should embolden them to reach out when they need the additional support.

And, here is our final time for interaction.

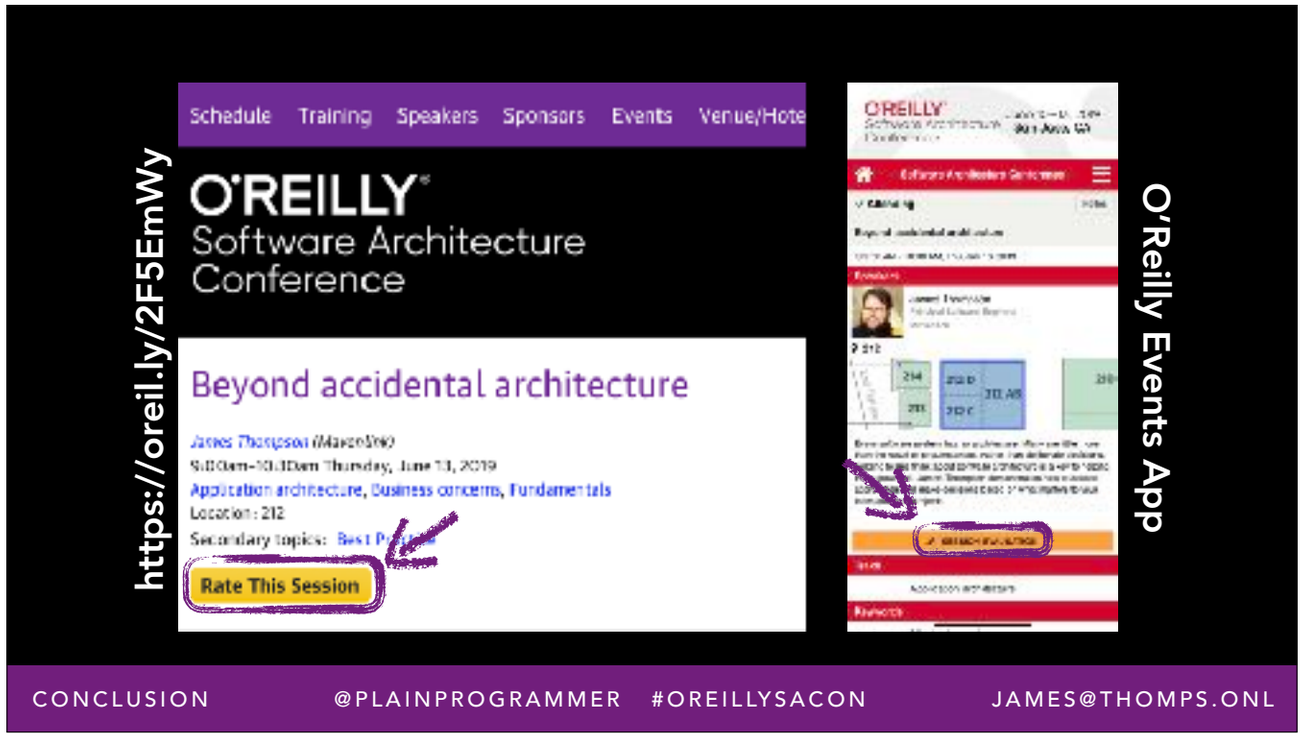
Do you think these approaches would be helpful for you and your team?
Are there other metaphors or approaches you think would be more helpful?

CONCLUSION

WHAT IS ACCIDENTAL ARCHITECTURE?

"When a system expresses more
the circumstances about when it
was made, rather than the
purpose for which it was made."

RECOMMENDED RESOURCES



Please, remember to go and rate this session. Your feedback helps me improve my presentations for future audiences.