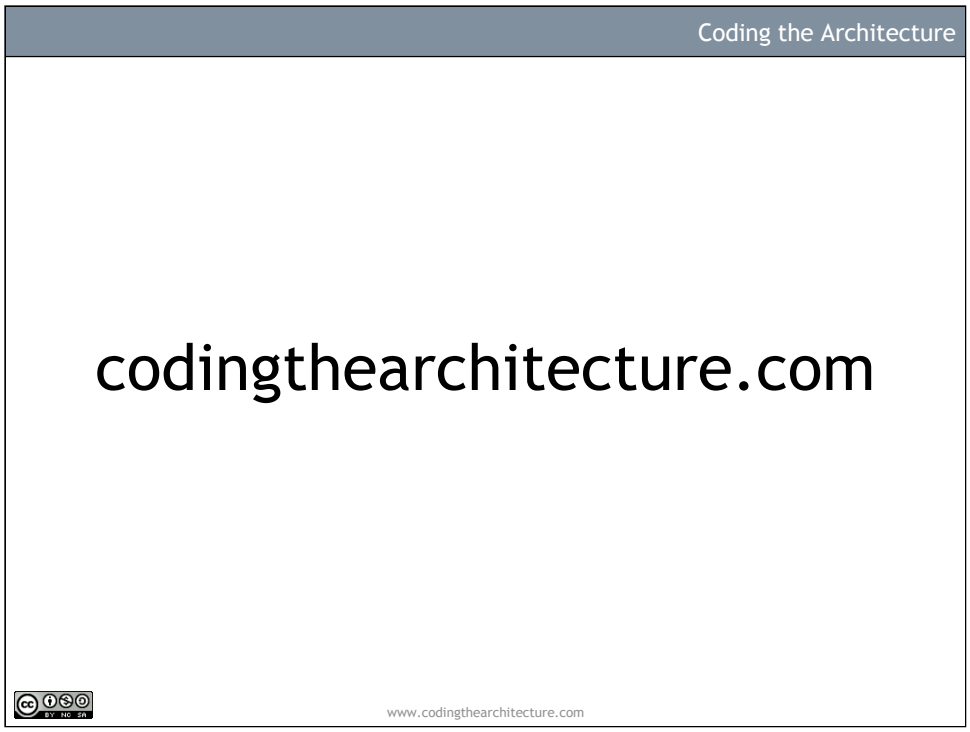
Coding the Architecture London User Group

Wednesday 3rd October 2007



Formerly thepragmaticarchitect.com.

A site providing content for aspiring and experienced software architects.

All of the contributors are practicing software architects, who primarily work within the finance and capital markets industry within the City of London.

The role of the software architect, rooted in real-world experiences.

The majority of the content on the site revolves around the role of the software architect.

All of the content is deeply rooted in real-world experiences.

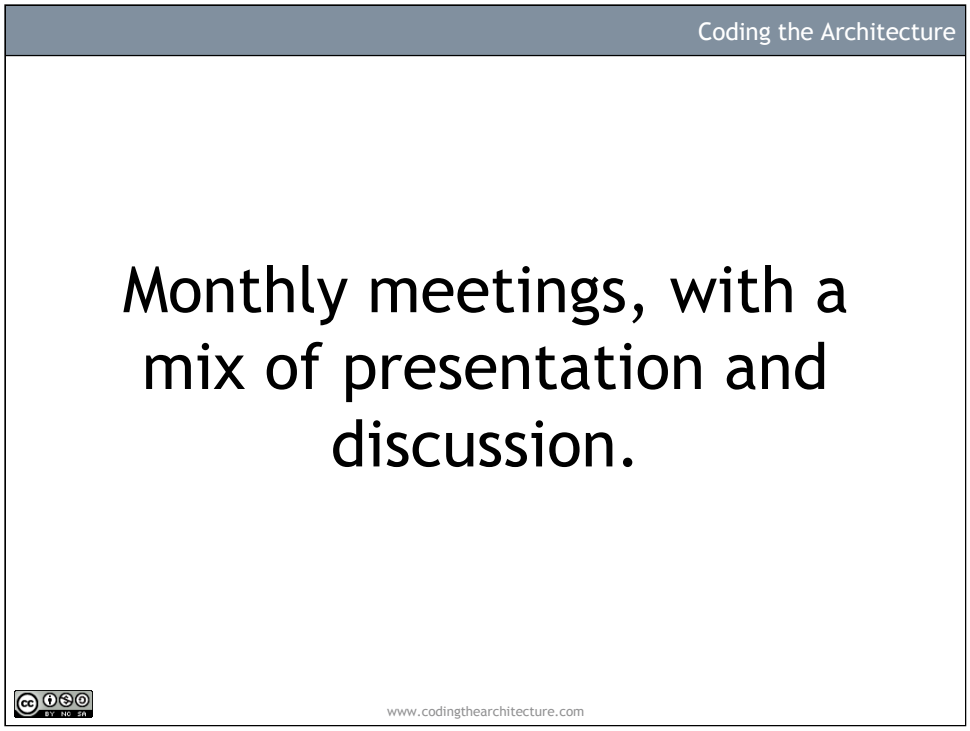
This isn't academic content – it's us sharing our experiences, good and bad!

We ran a session called “The Pragmatic Java Architect” earlier in the year and from the feedback decided to launch a more regular user group/forum.

Software architecture isn’t necessarily hard, but it can seem a little mysterious at times, particularly if you’re an aspiring architect.

We want to bring some of our experiences to the table so that it can help you.

At the same time, we’d like to build a community for aspiring and experienced software architects.



We'd like to aim for monthly meetings with a mix of presentation and discussion.

Some people like the former, some the latter. My hope is that the presentations will be thought-provoking enough to spark some good discussion afterwards.

We're definitely looking for other people to present – if you have anything you want to talk about, please let us know.

The most important thing here is that we're looking for people that are willing to share their experiences.

Future topics.

Making the jump from developer to software architect.
Coding and the role of a software architect.
What do you capture in your software architecture document?
Dealing with non-functional requirements.
Real-world experiences with SOA.
Agile architecture : How much is just enough?
Hiring software architects.

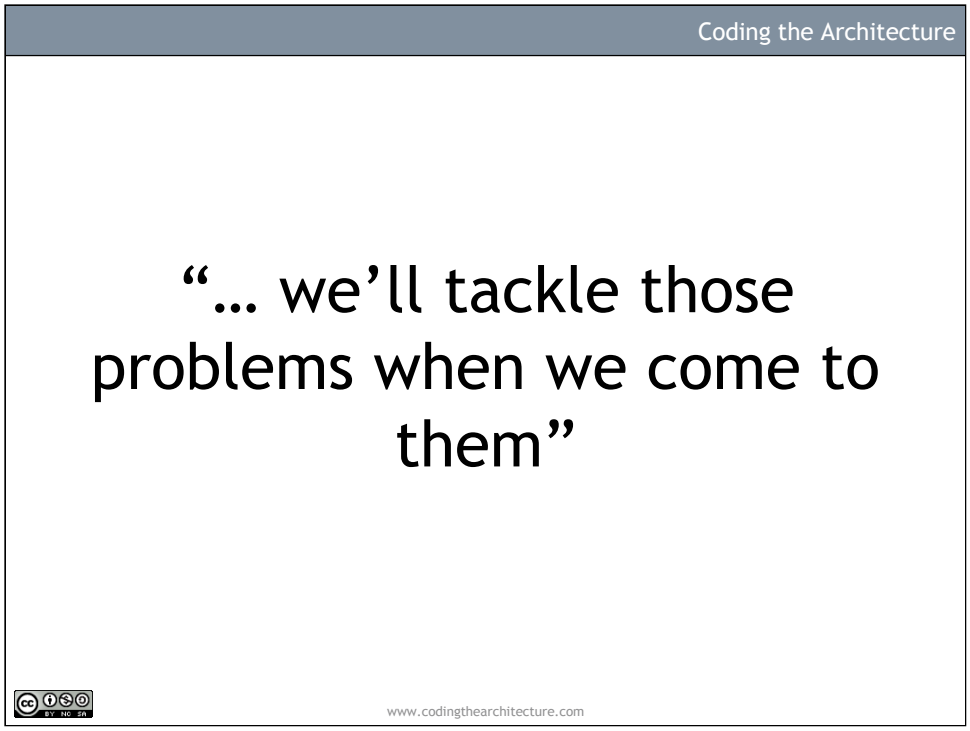
Architecting During the Implementation Phase

Kevin Seal
Wednesday 3rd October 2007

We're going to explore the consequences of making architectural decisions while implementation is ongoing.

These are simply my experiences of a few recent projects – they're certainly not prescriptive!

My goal is to talk for less than 30 minutes and then we'll break for discussion.

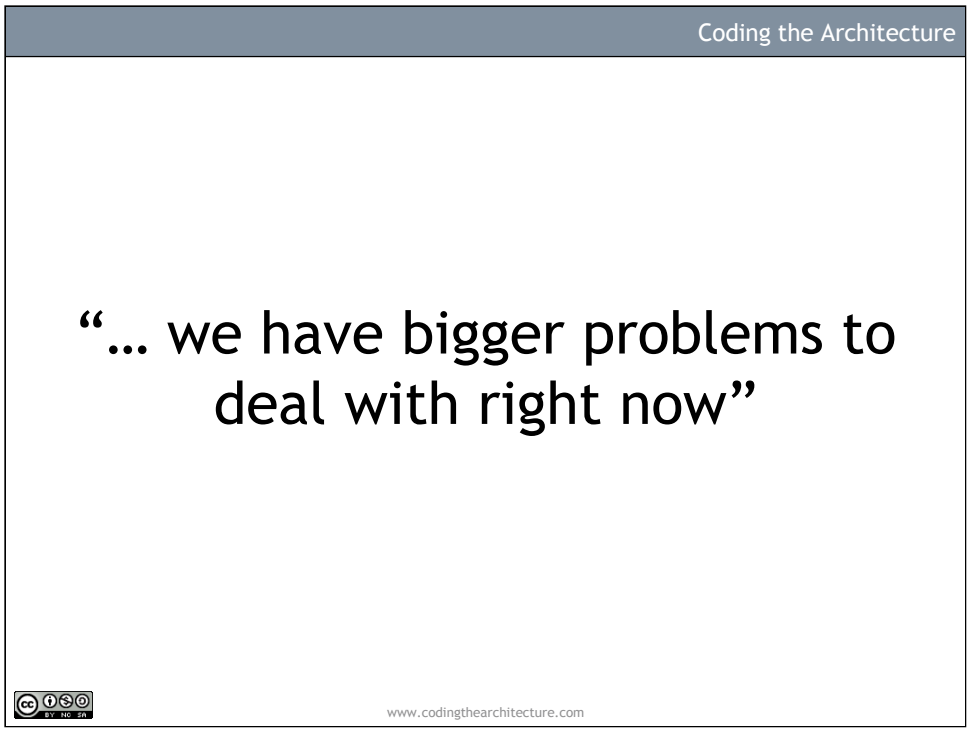


Getting stuck into implementation before giving things careful consideration is often accompanied by this phrase.

Sounds sensible – why paint yourself into a corner now?

But it's not a justification or mitigation, it's simply a statement of purpose.

So what do you or someone else mean by it…?



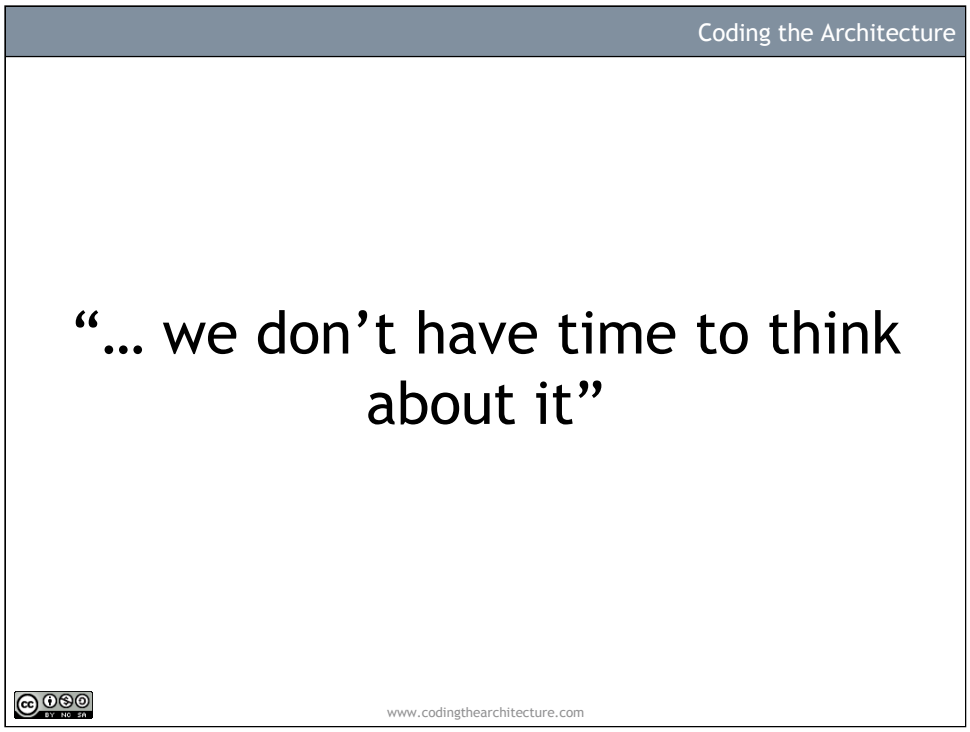
Basically means either:

- it's not that important;
- there're more serious things to deal with

If it's the former then you might not even consider the matter of architectural significance, just part of the implementation.

If it's the latter then these more serious matters should probably be holding up the start of implementation.

I suspect the "more serious matter" is really…

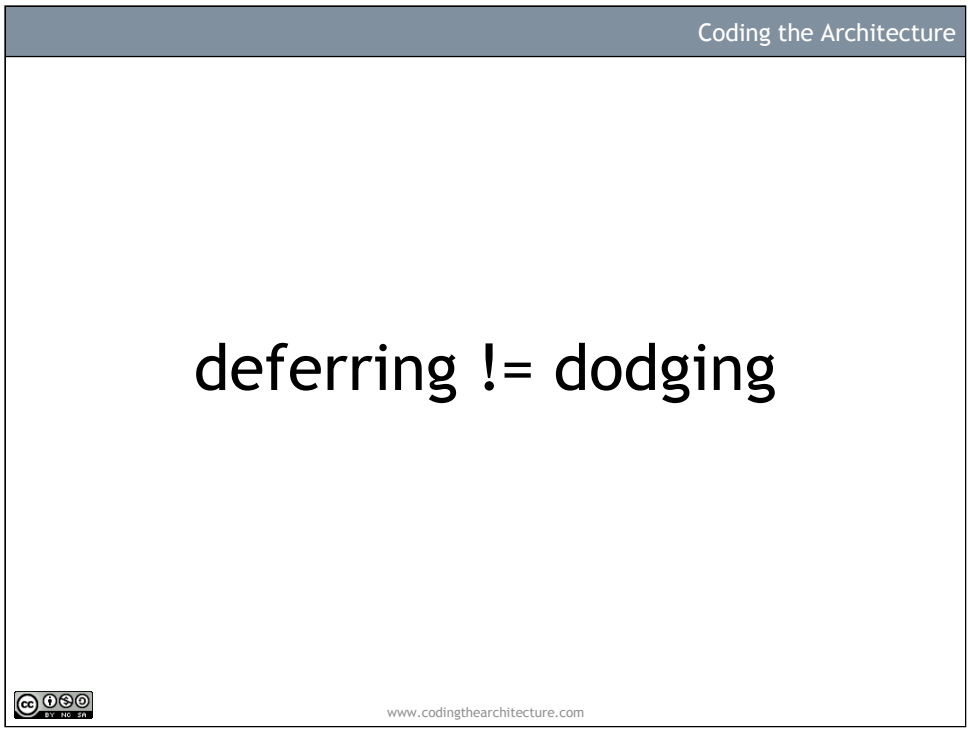


…a lack of time.

Basically means : we’d best get cracking

alarm bells

If you don’t have time to think about the architectural decisions then you could well be into the realms of a tactical solution. After all, if you can’t validate the architecture due to a lack of time then you need to be explicit about what you could be losing out on.

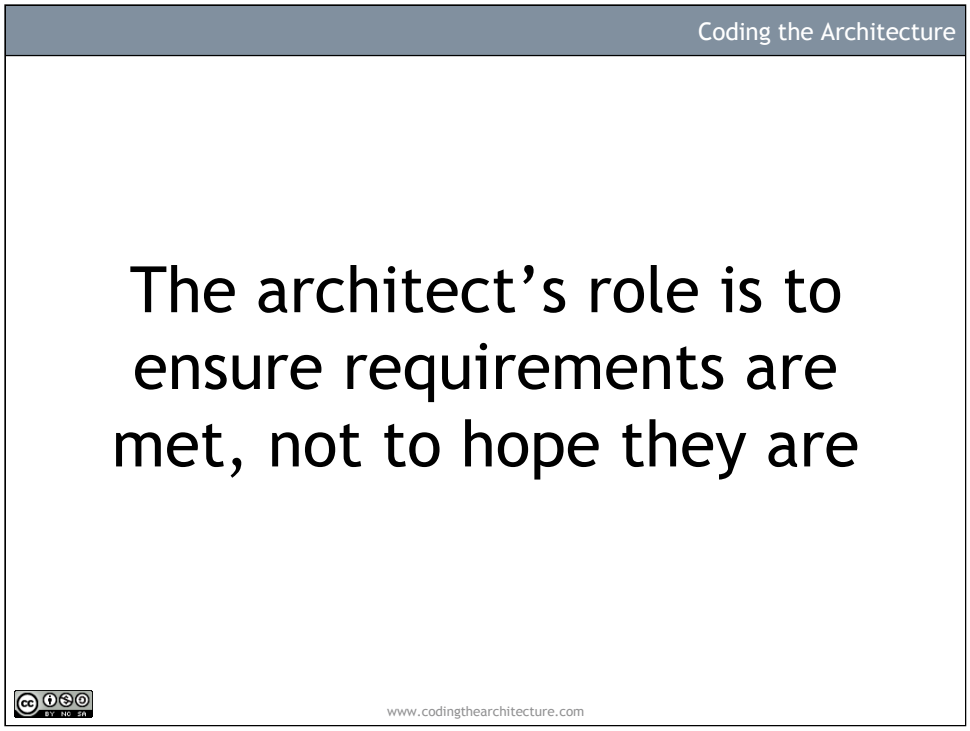


Deferring decisions to the last responsible moment is all the rage, particularly with agile methodologies.

Indeed, noting that not all architectural issues have to be resolved in order for implementation to begin is what permits a multi-skilled team to continue developing sprint after sprint without the need to stop and start. And this is when architectural decisions and implementation mix.

But the decisions still need to be made and they need to be made from a position of technical authority – otherwise what's the point in architecting?

In my experience a lot of technical decisions are deferred to the point where they make themselves through lack of time and options or sheer weight of surrounding implementation.



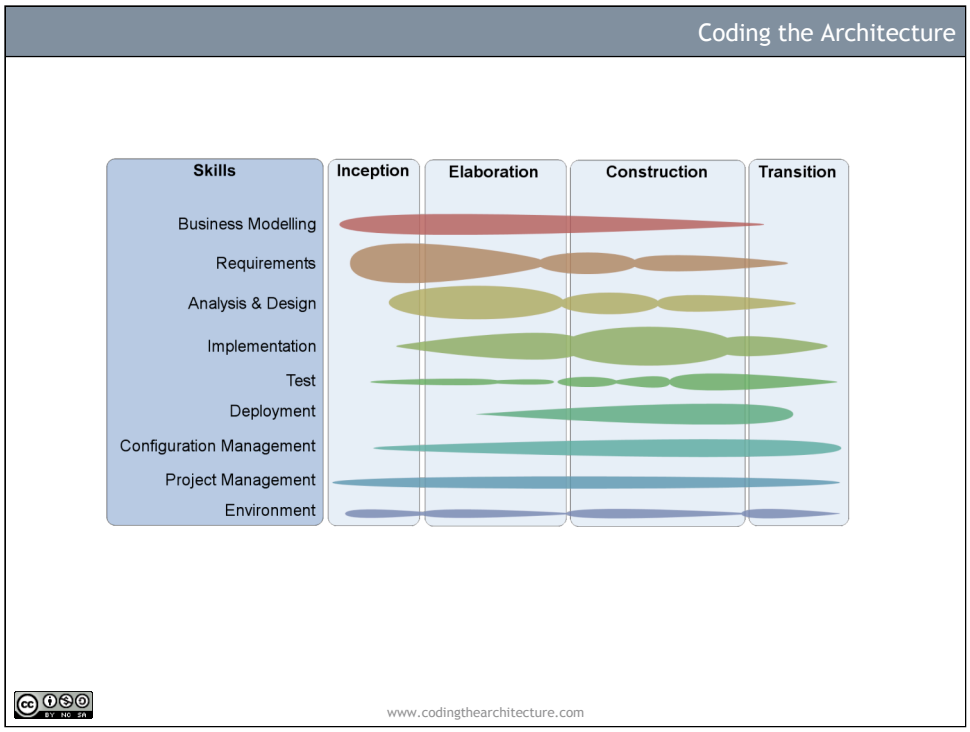
Every system has an architecture, but it is the architect's responsibility to conceive it, validate it and enforce it – not just to document it (post-hoc).

If all you ever do is implement blindly and fix problems when they crop up are you simply doing the job of a lead developer, albeit an experienced one?

When is the last responsible moment?

The key to deferring decisions to the point where implementation is being carried out is the ability to gauge when those decisions need to be made.

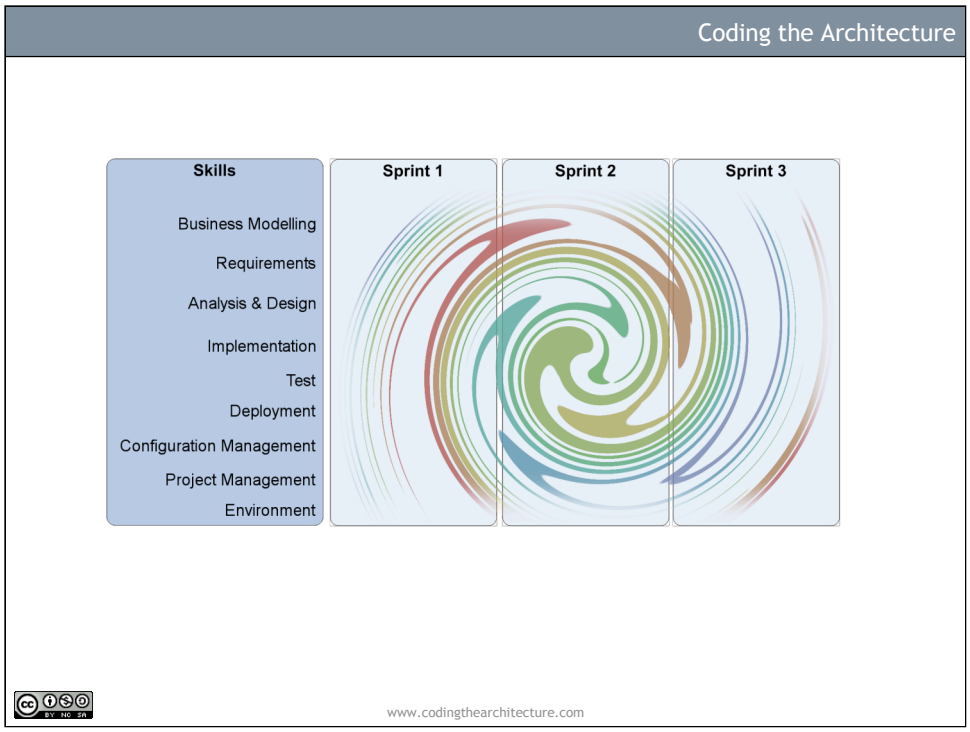
This is driven by various aspects of the project…



…the key to which is accepting that even agile techniques need structure to them:

- you still need to think about things before you commit
- you still have dependents

While iterative RUP is not necessarily ideal for everything, a lot of “agile” projects I’ve worked on have looked a bit more like…

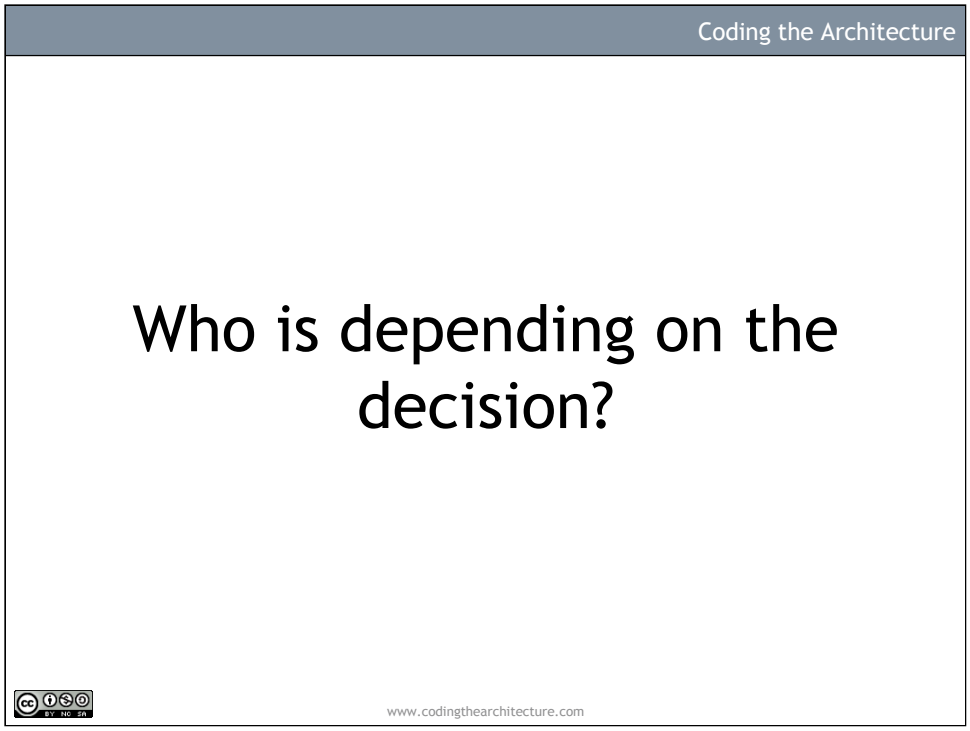


…a free-for-all.

Self-sustaining development teams are good for development. But there is always a boundary where this team finishes and a formal interface spoils the party:

- hosting companies want contracts
- quality assurance teams need documentation
- support teams want handover
- stakeholders want to know how at risk their investment is

So if you're deferring an architectural decision you should know why and what the consequences will be…

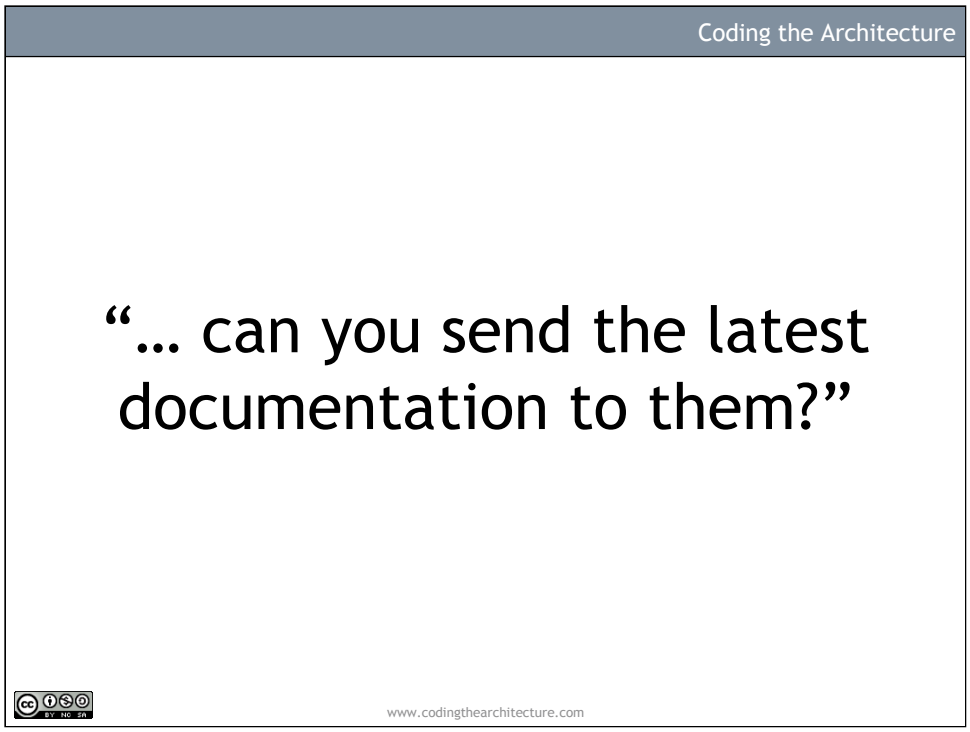


There are numerous dependents of an architectural decision:

- development team
- testers
- commissioning test, production environments
- publicity

In particular, anyone engaged on a fixed-price, fixed-time or fixed-anything basis, notably outsourced teams, will need to know the broad architecture in order to price up and commit to their portion of the work. And if you change your mind on a fixed-price contract you could find the consequences far greater than just a bit of refactoring.

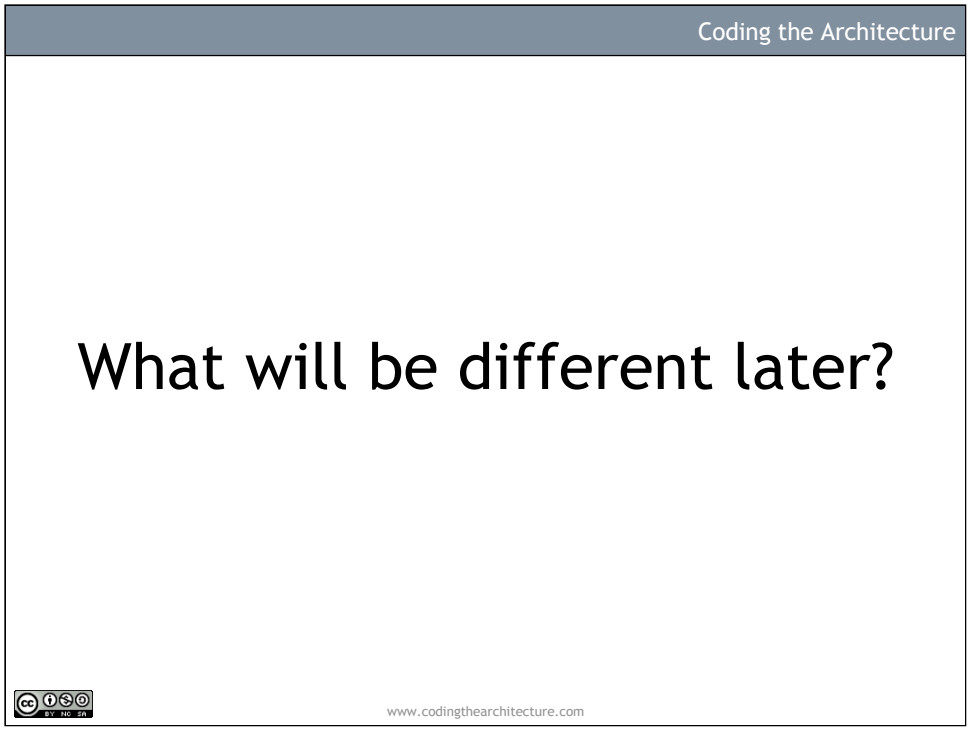
For example, one of the offshore development companies I work with always start at risk. They submit a proposal for work against an assumed architecture. When the project starts they document the architecture and start coding simultaneously. When the architecture document arrives for review I often make changes and they sometimes get bounced back by the ODC on the basis that it would change the nature of the work that they are contracted to do. While the solution is probably to get the architect to do the architecture up front (duh!) it demonstrates what would happen if a decision is changed during implementation.



This question seems to crop up at the end of just about every project I've worked on. In some cases an export of the project Wiki will suffice but that rarely seems to satisfy everyone.

So why do most my projects end up with me creating some documentation to send someone? It's probably because I've forgotten about them…

- configuration/build manager
- production support

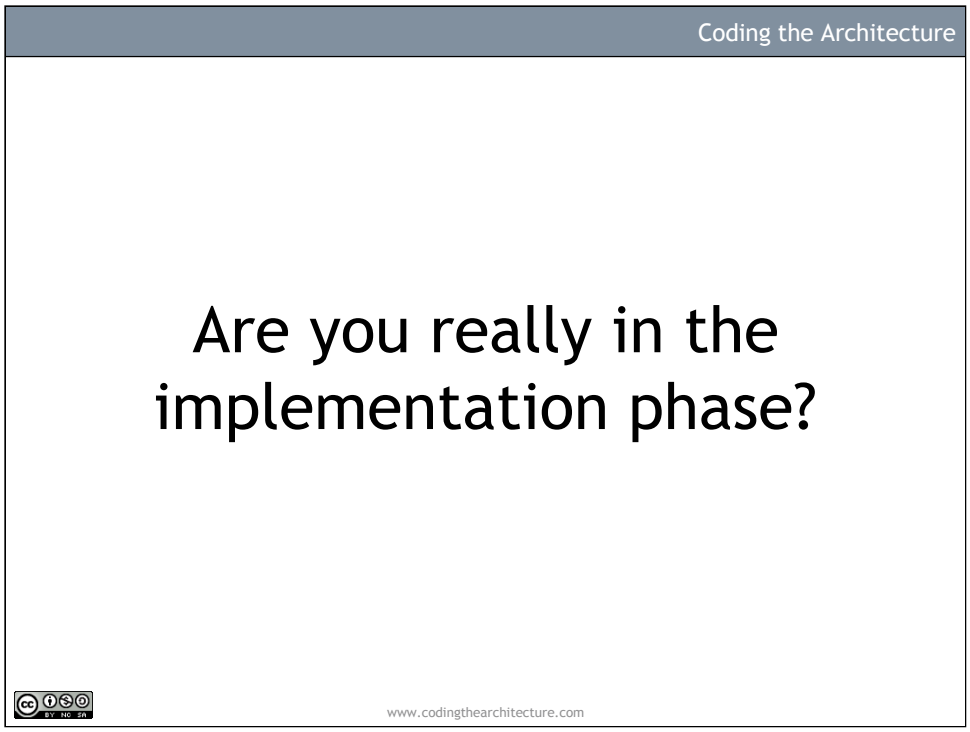


Unlike leaving revision until the night before an exam, there may be good reason for deferring an architectural decision:

- functional and non-functional requirements have a habit of turning up during development
- you'll have learned a lot from the existing implementation
- priorities tend to change
- if things change you won't have wasted any effort

There are also numerous reasons for not deferring:

- there'll be less time!
- someone else may have painted you into a corner (third-parties also working on the project)
- you may have painted yourself into a corner (what will all that coding you've been doing)
- by committing early you might actually prevent change (particularly arbitrary change, eg bun fights with third-parties)

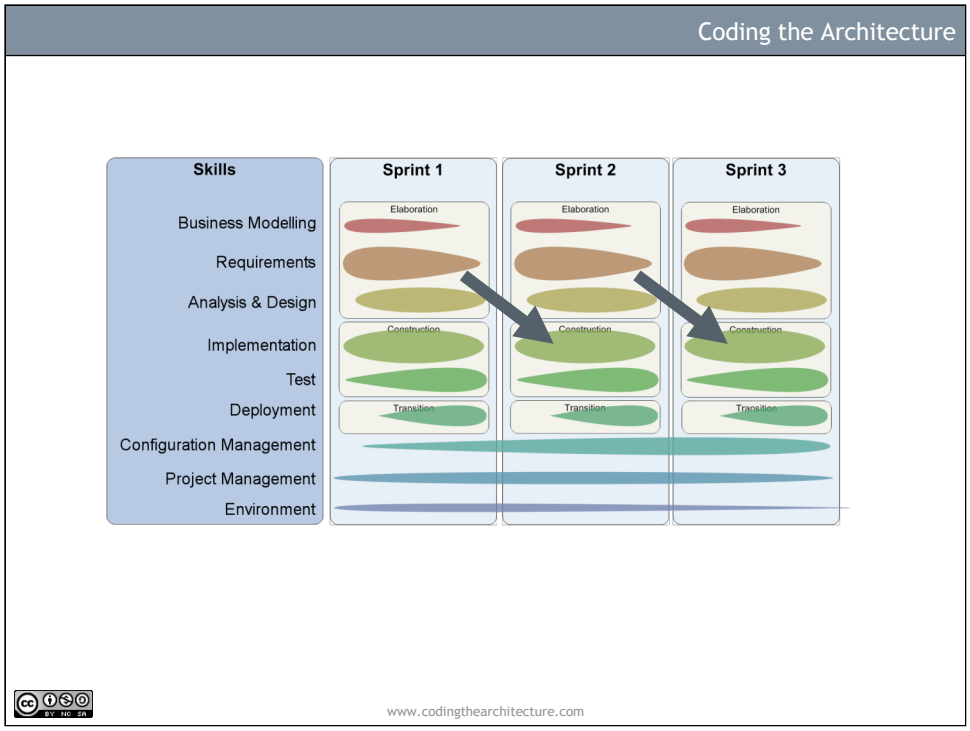


Despite sometimes being heavily involved in coding, I don't think the decisions I make come from within an implementation phase. They probably come from the development of a spike architecture as part of elaboration.

If you can do a spike architecture I'd strongly recommend it – and I'd recommend it's done during elaboration…

Making architectural decisions in front of the development team can be a good learning experience and provide useful insight but it can also be confusing; it weakens your technical authority and blurs the architectural vision for the system.

By imposing a little structure on the agile process you can get elaboration and implementation to coexist …



Ideally you should be doing the elaboration for the *following* sprint in order to separate the decision making commitments from the implementation commitments.

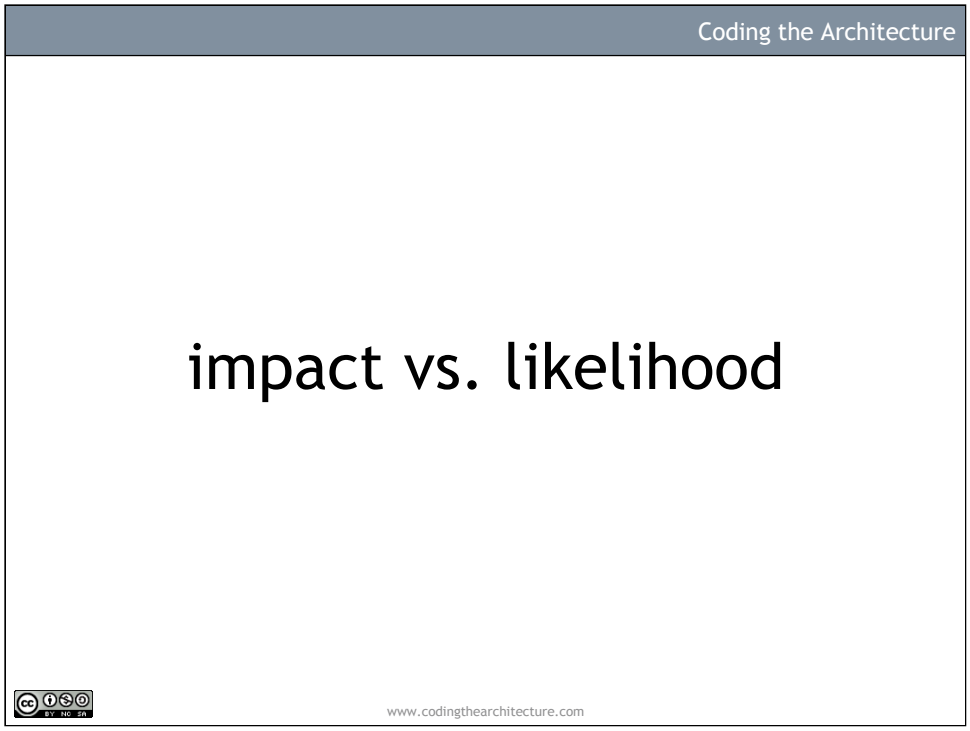
* Agile RUP diagrams are usually a bit more structured than this, having elaboration sprints – this is simplistic so as to make the point about the two tasks being run concurrently *

Working software is a catalyst … and a constraint

Agile techniques are excellent at putting working software in front of stakeholders early.

But if your architecture is wrong it's harder to change it:

- public interfaces
- client technologies
- data migration



There is a difference between architecture decisions and other design decisions:

Put simply, by definition, the cost of change of architectural decisions is greater.

"Architecture represents the *significant* design decisions that shape a system, where significance is measured by cost of change." – Grady Booch

Do you want to minimise the cost of change by getting the decision right early or minimise the likelihood of change by leaving it later?

A project I'm working on is currently on the brink of implementation, with no discernible system architecture having been defined.

So how do we choose which problems to spike now and which ones to leave later?

Cost of change:

- technology / product selection
- synchronous / asynchronous interfaces

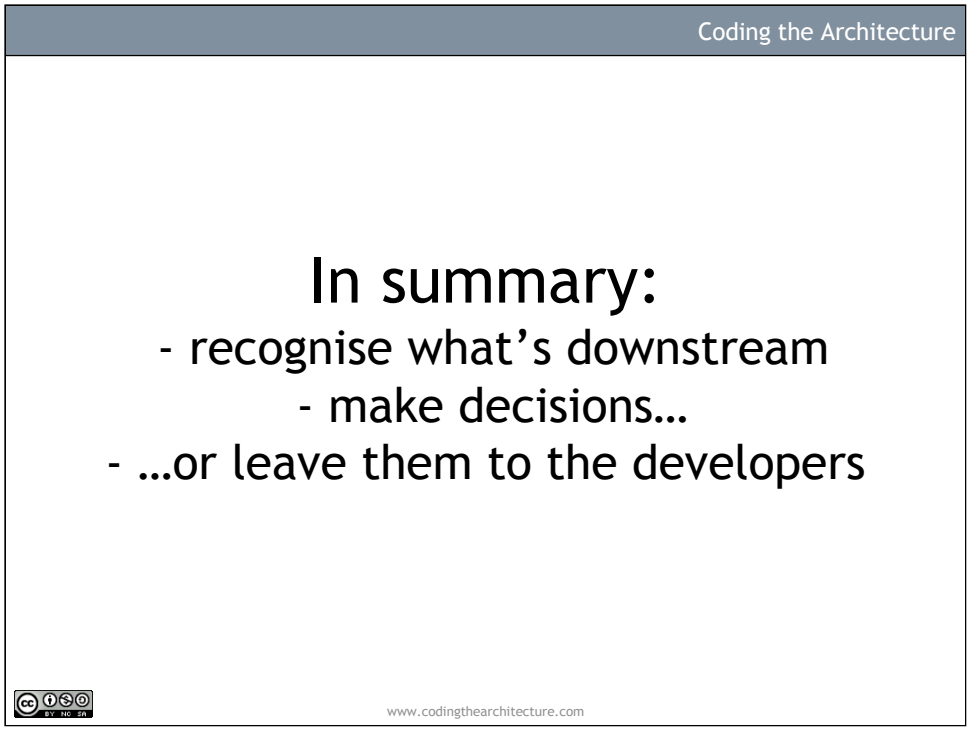
Likelihood of change:

- data structures
- interface specifications

We're doing a mix of both but in the past I've tended to leave data structures and interface specifications until implementation when the detail emerges.

On my current project, however, we need to stub these out for third-party teams to implement against.

This is working solely because the development team is capable of being actively involved in making the architectural decisions and can turn change around quickly.



It's not just the code that wants to know what the architecture is so "the last responsible moment" may well be sooner than you think.

Don't hide behind the methodology or lack of pressure – do your job and define the architecture :

- Spike
- Prototype
- Proof of concept