Coding the Architecture London User Group

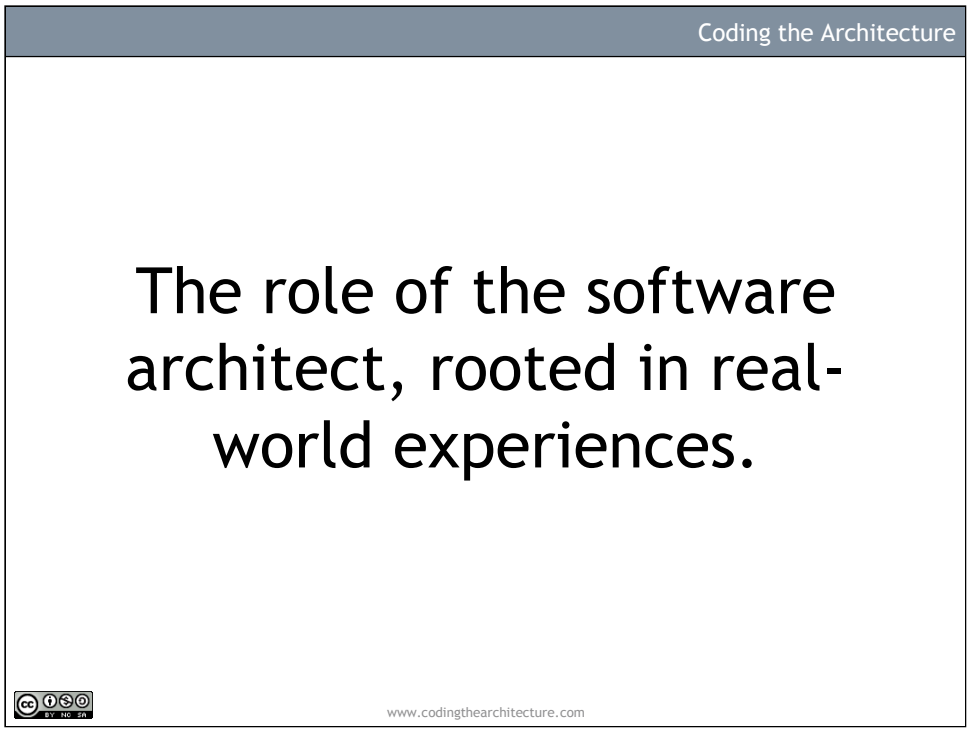
Tuesday 4th September 2007

codingthearchitecture.com

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 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.

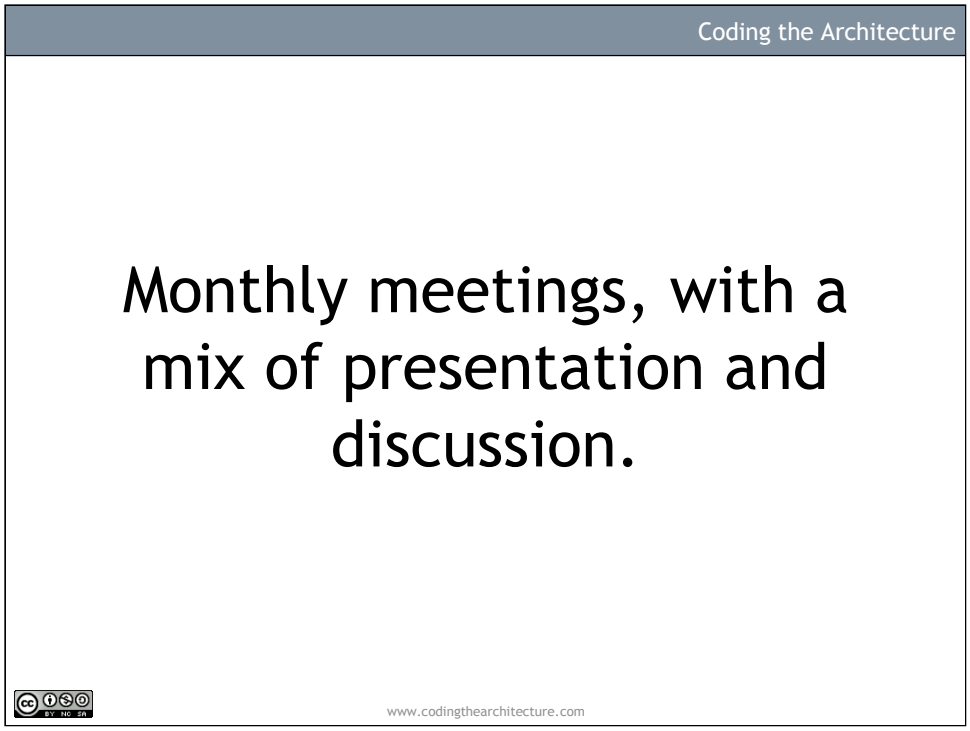
This isn't academic content – it's us sharing our experiences, good and bad!

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.

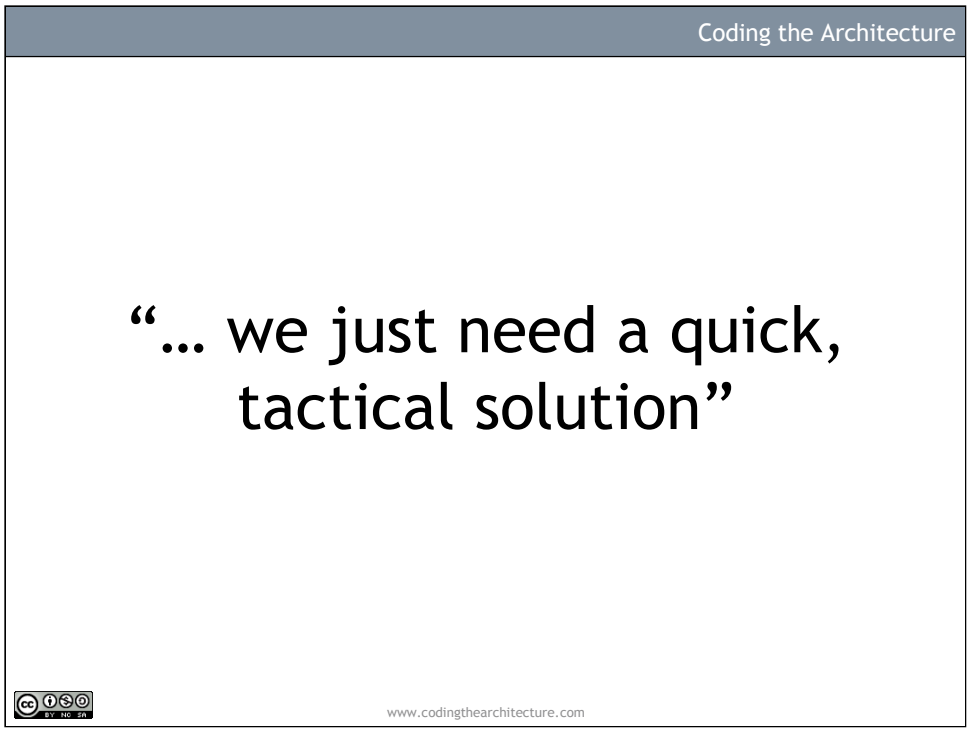
The Implications of Architecting a Tactical Solution

Simon Brown
Tuesday 4th September 2007

We're going to explore what a "tactical solution" is and how this affects the way that you approach the architecture and design.

I don't have all of the answers; I can only share my experiences.

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

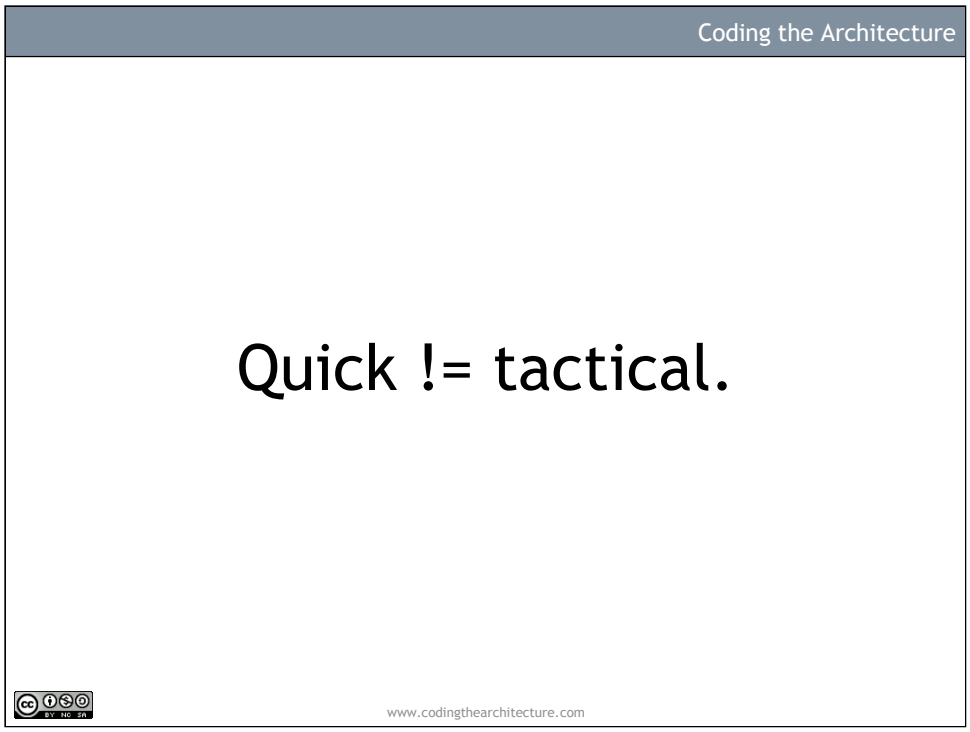


Famous words that will strike fear into a software development team!

In my experience, there's no such thing as a tactical solution? What this really means is…

"… we need something built as quickly as possible and, although we think it will have a limited lifespan, it will more than likely remain in use for some time into the future"

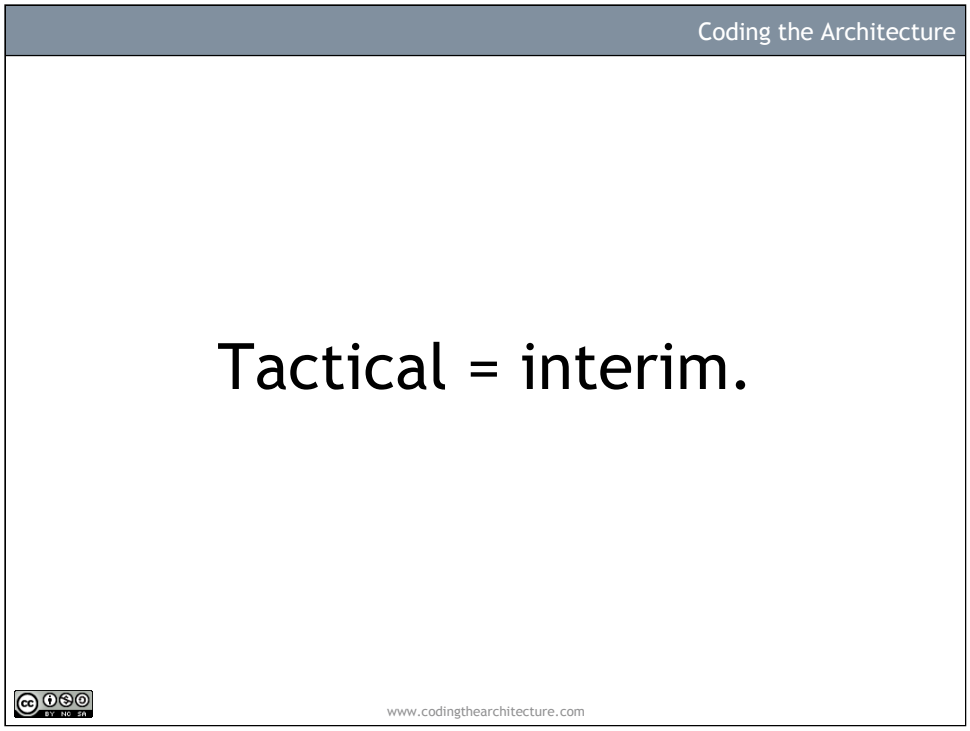
Basically means : we want it now and it's going to be running for a long time!



For me, the real stumbling block is that quick doesn't necessary mean tactical.

Iterative, incremental and agile methods let you build something quickly, but you probably wouldn't describe the resulting solutions as tactical.

No, tactical is about much more than the speed of delivery.



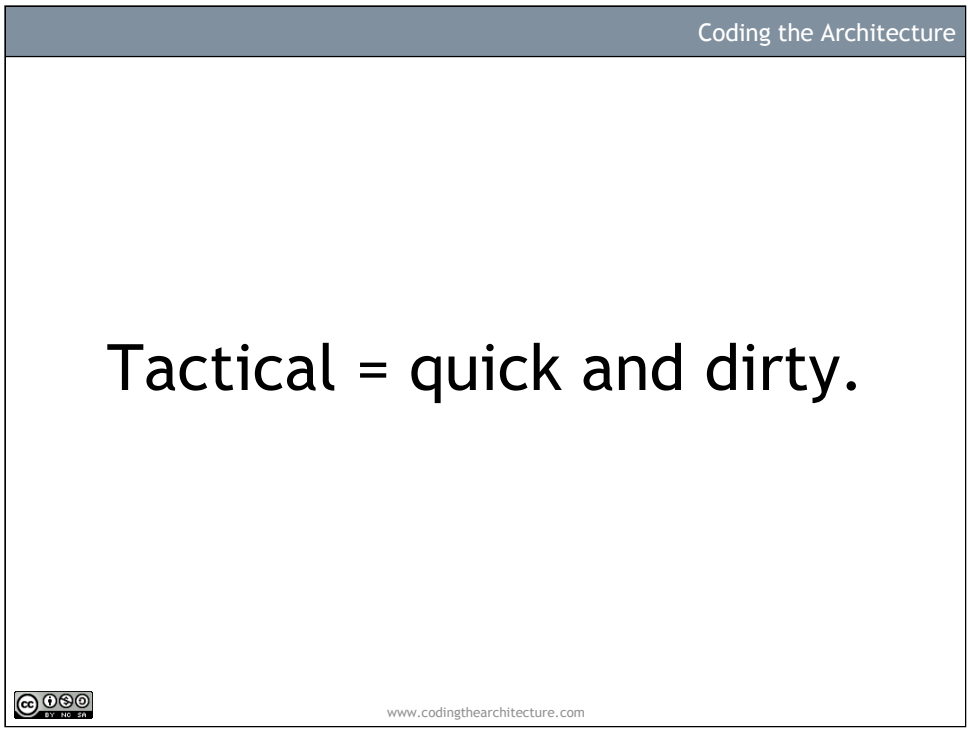
Firstly, I like to think of tactical solutions as being a stopgap before something bigger, better and more strategic comes along.

It could also be a stopgap before a change in business process.

It could also be a stopgap to help with a particular technical problem (e.g. a temporary feed between two systems while the a “proper” integration solution is being developed on either side).

Tactical solutions might be part of a larger strategic roadmap, yet not be strategic themselves.

Typically though, tactical solutions are unforeseen and not planned for.



When somebody asks for a tactical solution, they usually aren't looking for something elegant.

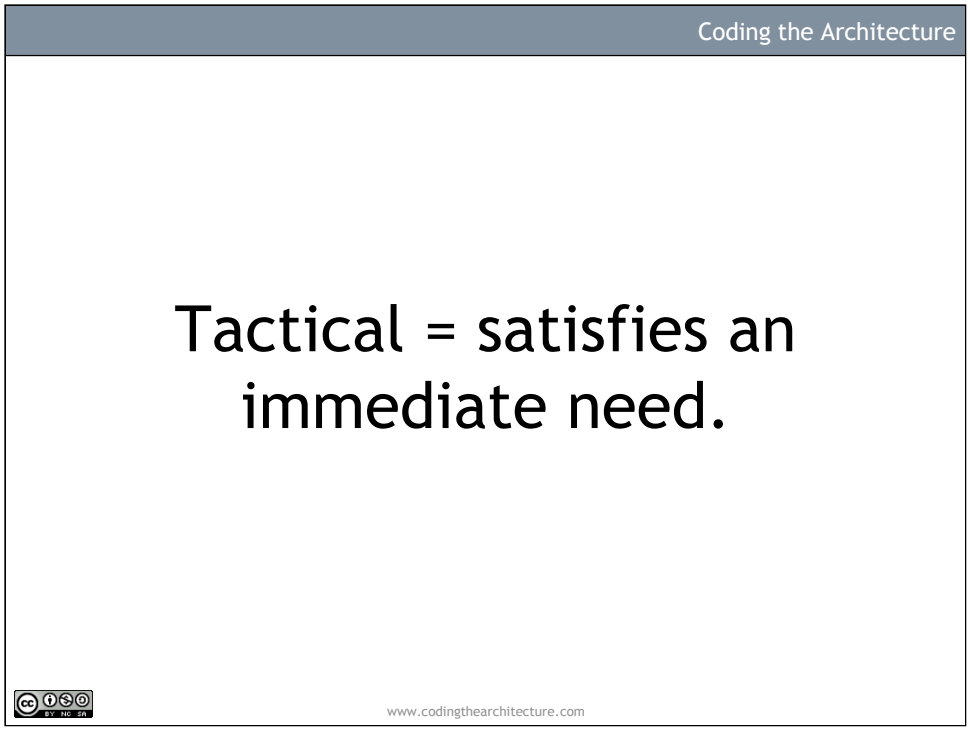
Functional yes, but pretty, no.

"Quick and dirty" is rather subjective.

A Java webapp with no layering (data access in the JSPs) may appear quick and dirty to some people, but not others.

Likewise, the same Java webapp with basic architectural layering may appear quick and dirty to people living in an SOA world, where architectural principles state that reuse should always be maximised.

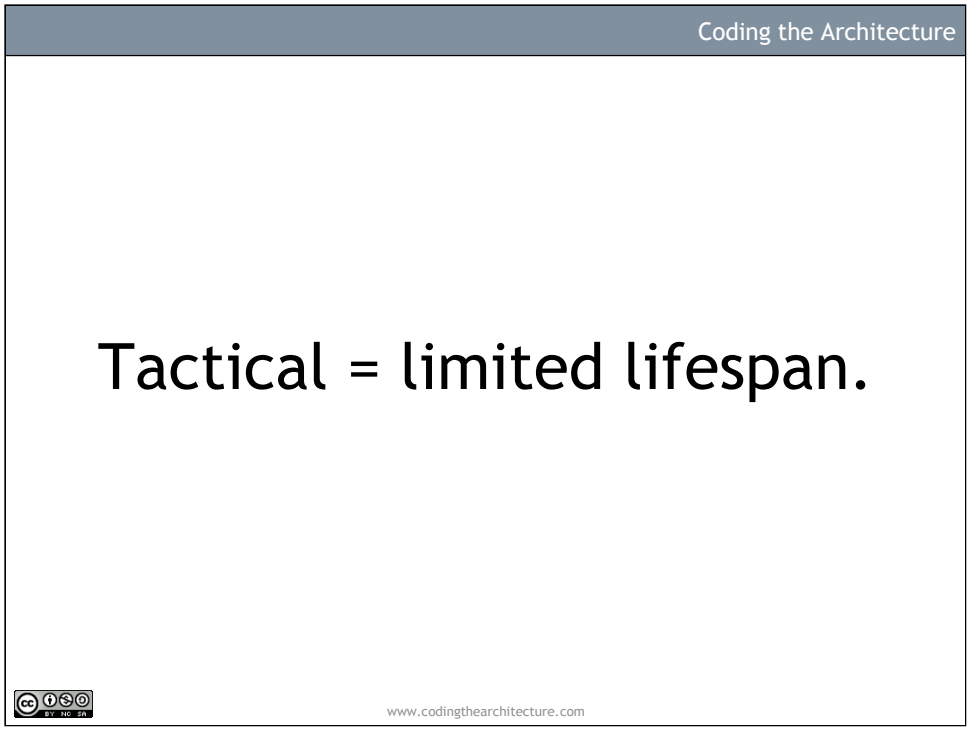
What would you define as "quick and dirty"?



Tactical solutions are generally required because somebody has an immediate need and can't wait.

A tactical solution might solve an urgent technical or business problem.

Once the problem has been solved, it might never come back.



Tactical solutions are often said to have a limited lifespan.

“It’ll only be running for 3 months”.

“It’ll be thrown away when the new strategic solution is delivered”.

“We’re planning to rewrite all of this anyway”.

Whether tactical solutions really do have a limited lifespan is questionable.

Supporting tactical solutions can be hard.

How many people here have worked on a tactical solution that exceeded it’s expected lifespan?

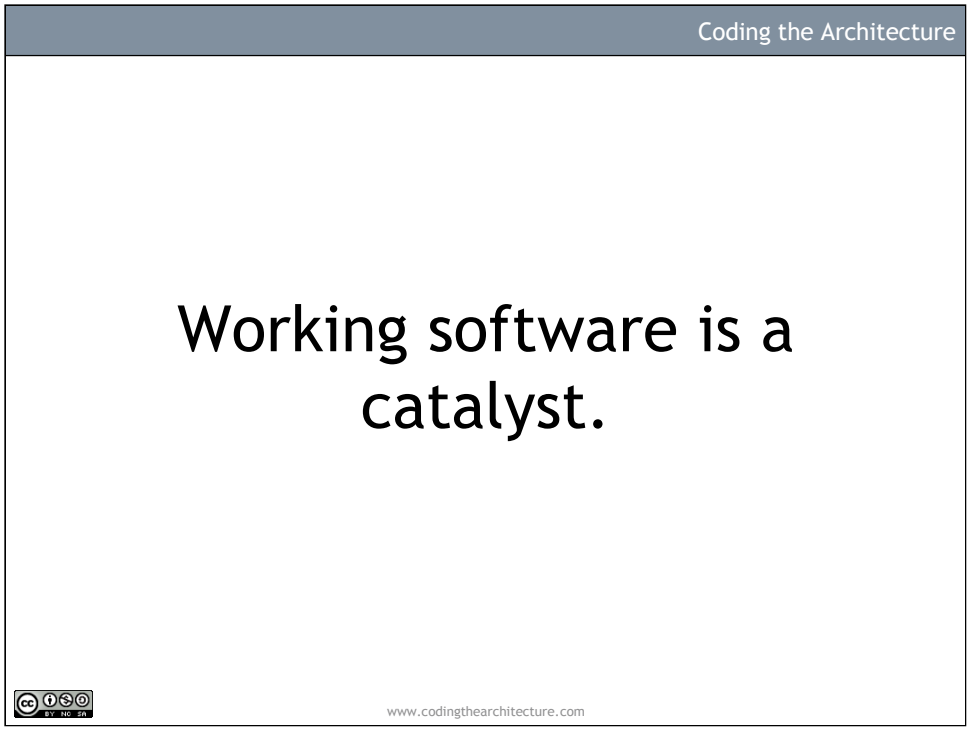
Is there such a thing as a tactical solution?

I'm not really sure that there is such a thing as a tactical solution.

One tactical solution we built was running for 1-2 years (original estimated lifespan was 3-6 months).

We had to build something quick, in aggressive timescales, but it certainly didn't end up being a tactical solution.

We'll come on to this at the end when I talk about what happens when you get it wrong.



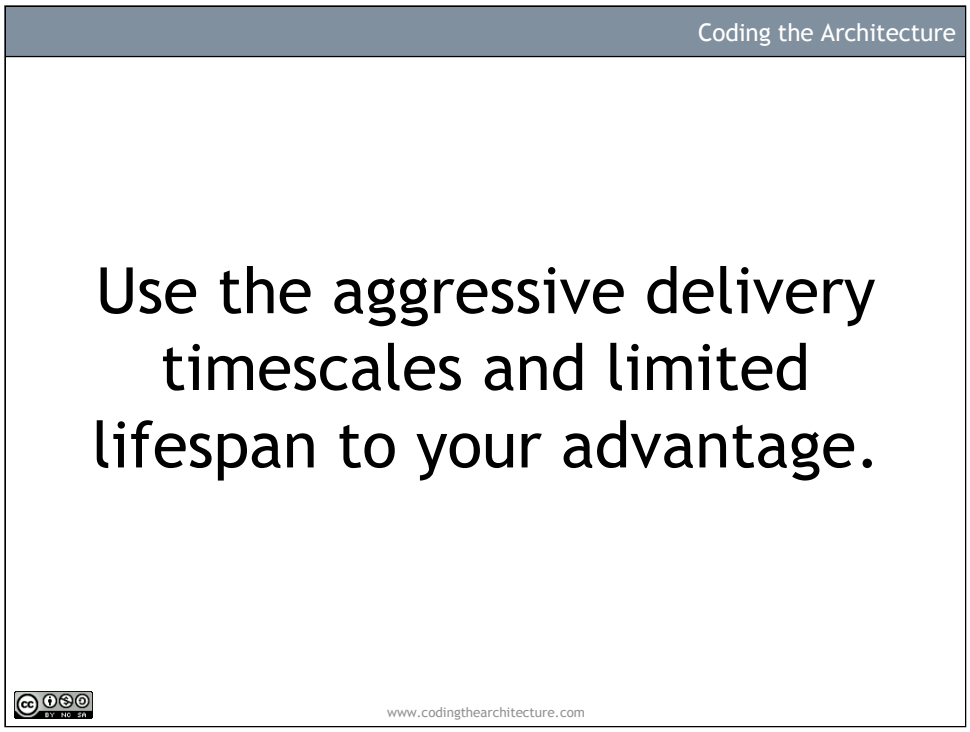
One of the reasons that agile techniques work well is because working software is a catalyst for new ideas.

On paper, a tactical solution looks exactly that – tactical. But in real-life, it looks so much different.

“Oooh, wouldn’t it be great if the system also did X?”

“The system looks really good and I know that we can change the config via the database directly, but we are going to need a full web-based administration facility”.

Anybody ever shown mock-ups to a user where the response has been, “that’s excellent, when can it go live?”.



Architecting a tactical solution is tricky.

We usually have to balance everything and ensure the non-functionals are met; now we have aggressive delivery timescales and a limited lifespan thrown into the mix.

But it's not all bad news - you can use these to your advantage and let them influence your architecture and design decisions.

Don’t agonise over technology decisions.

You probably don’t have time.

Do you have existing experience?

Does the team have existing experience?

If not, are you sure you want to carry additional risk of learning something new?

If you really are under aggressive timescales, use this as a reason to use technology that might not normally be used approved.

Are you confident?

Don’t agonise over design decisions.

Again, you probably don’t have time.

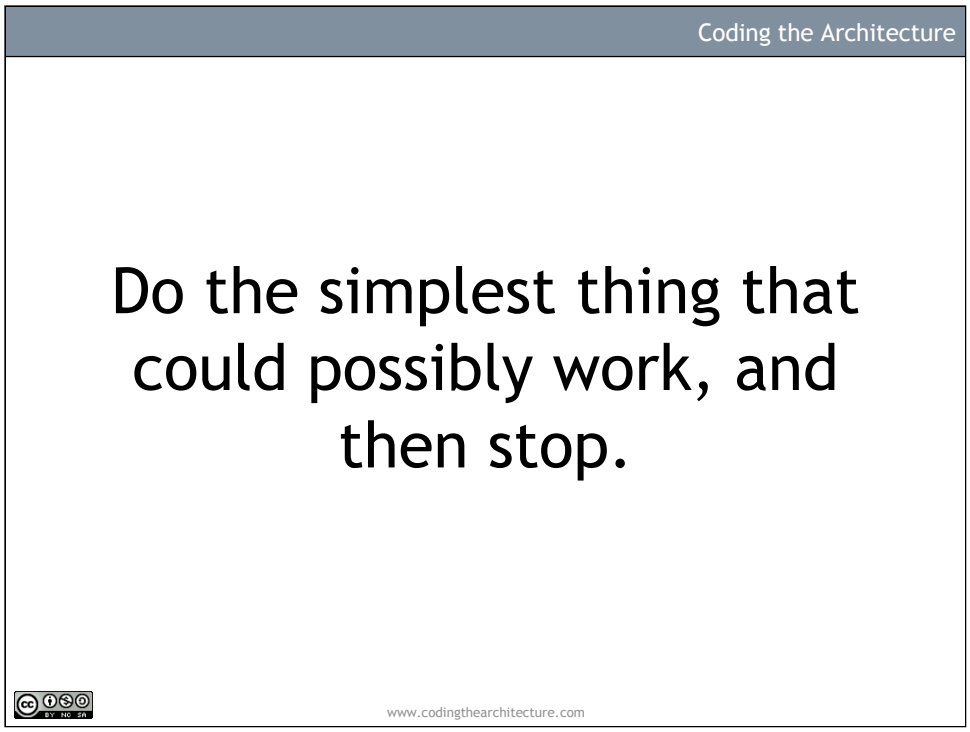
Is it fit for purpose?

Will it meet the functional and non-functional requirements?

It is easy to build and test?

Does previous experience provide you with confidence that the design will work?

Will the rest of the team understand it?



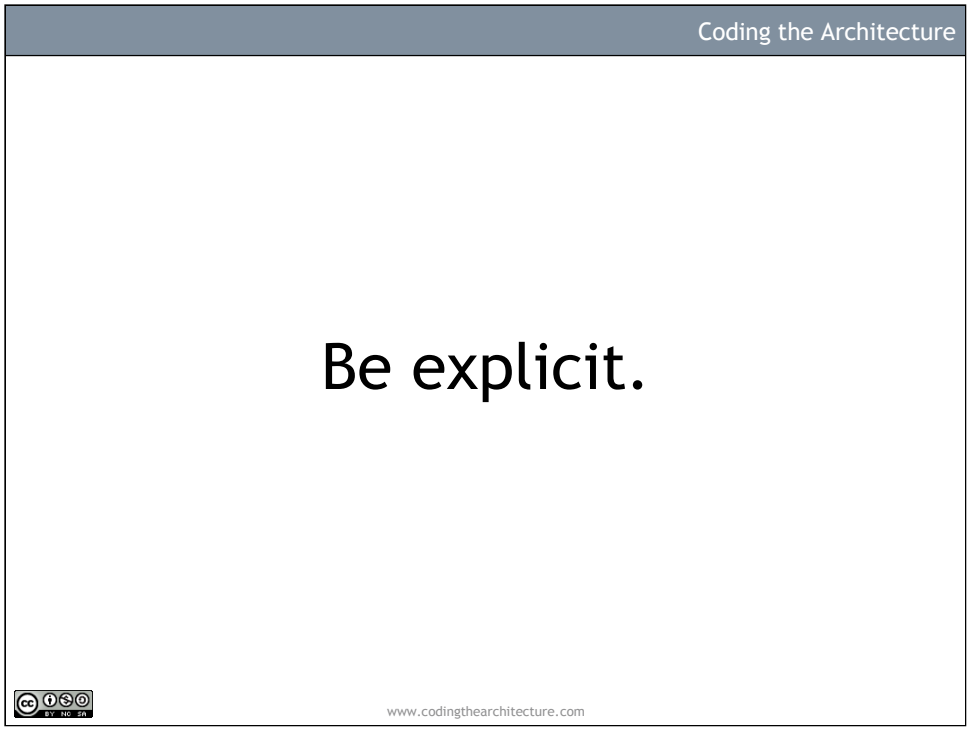
An agile principle that is very applicable to designing a tactical solution. We're not talking elegance, we're talking simplicity.

Take a "back to basics" approach.

Make everything as easy as possible : to design, to build, to test, to deploy, to support.

Don't use a complicated technology stack that you aren't familiar with.

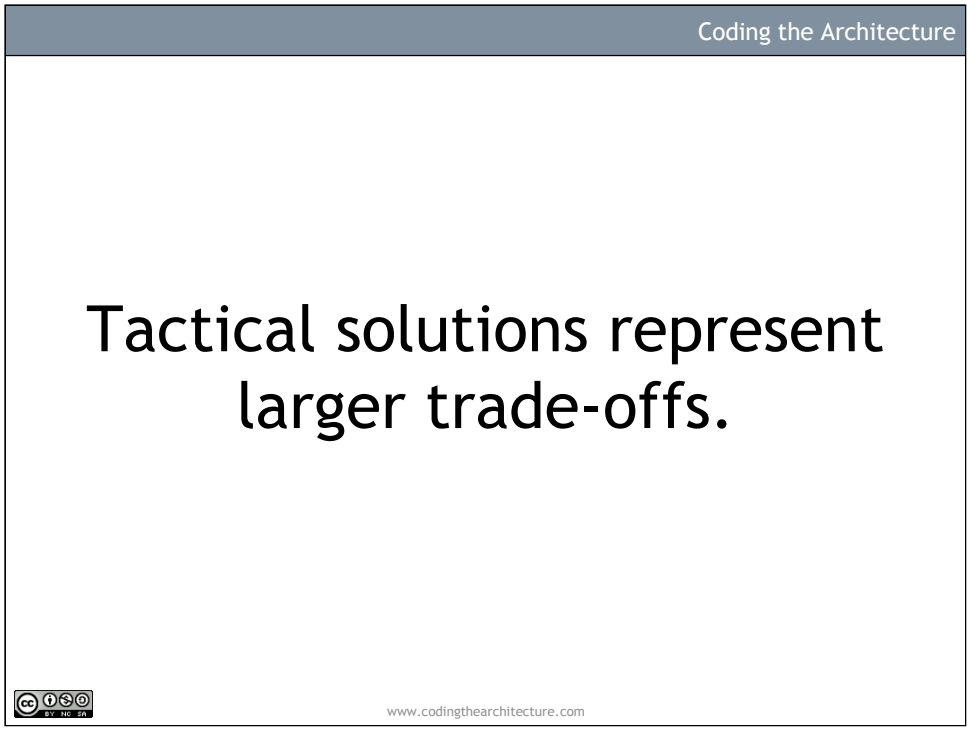
Likewise, don't use a complicated design that you aren't confident will work.



Be explicit about what the system will and won't do.

Aside from any contractual requirements, being open and explicit makes everybody make informed decisions about what they are getting.

As usual, make sure any assumptions you are working to are explicit.

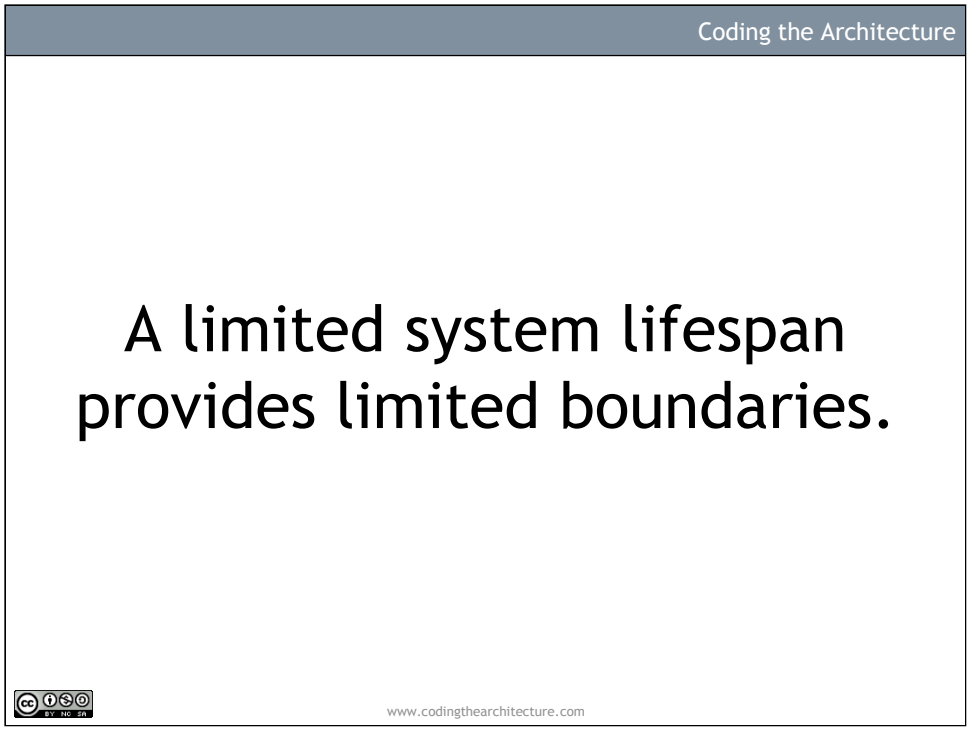


You might trade-off extreme performance and scalability for a simple object/data model that is easy to work with.

Understand the implications of the trade-offs you are making, make sure everybody else understands them and, again, be explicit.

Say what the system will and won't do.

State the trade-offs and their limitations to allow others to make informed decisions.



A limited system lifespan typically provides boundaries for the key NFRs, so take advantage of this.

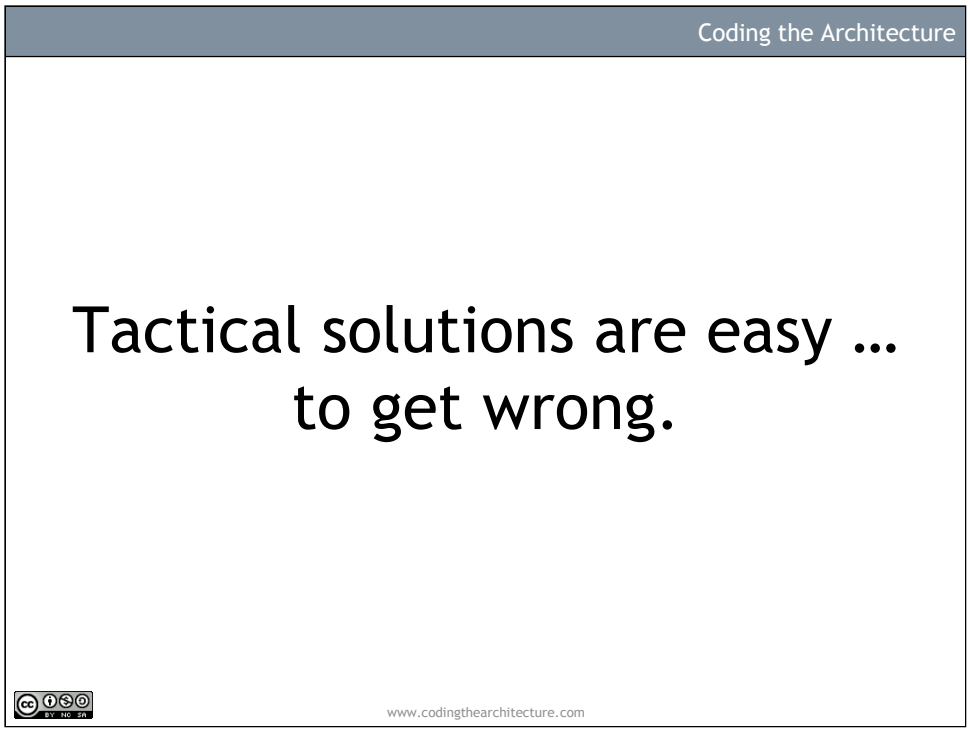
Work out those boundaries and architect the system to meet them (plus a bit more for safety).

If the system will only be live for 3 months, it should be easy to work out how many users/data/etc the system will need to cope with.

Normal systems usually allow for 10x growth, or growth over an X year period.

Take advantage of the fact that tactical solutions have a limited growth.

Do you really need a true horizontally scalable system given that the system will only be in use for 6 months and will support 20 concurrent users?



Don't think tactical solutions are easy to design/build - just like any other solution, they are easy to get wrong.

I've learnt the hard way.

One of my projects was widely acknowledged to be a stopgap prior to something bigger and better happening.

Was hard to agree non-functional boundaries (particularly around performance and scalability) and I didn't make a point of getting to the bottom of these.

A few months later, we were called back because the system wasn't performing

The business rules in our system were data driven and shared with another system.

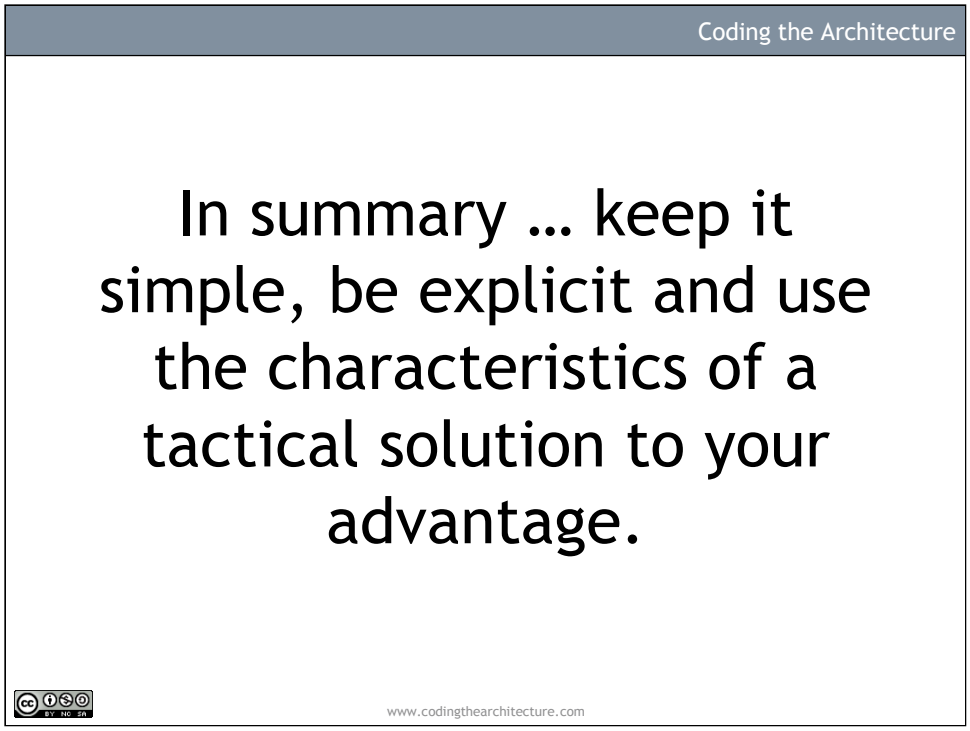
Our system had never been tested against the new data.

Don’t ignore the non-functional requirements because “it’s only a tactical solution”.

Targets should have been made explicit rather than being ignored because it was "a tactical solution“.

We didn't explicitly say that a change in the data defining the business rules may change the non-functional characteristics of the system.

We didn't give this much thought given the aggressive timescales and limited lifespan of the system.



Tactical solutions aren't that much different to regular solutions, but they do provide some benefits.

Keep it simple and be explicit.