Software Architecture is ...

the highest level concept of a system, ...
its structure of significant components
interacting through interfaces

— IEEE standard

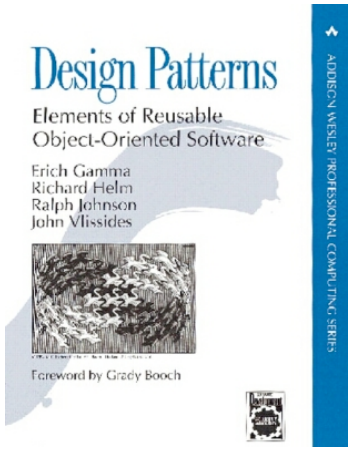
Ralph Johnson: Bogus. Different concepts for customers and developers. Some developers know only a single piece.

(This is the same Ralph Johnson from the Gang of Four that wrote Design Patterns, a University of Illinois CS faculty member. He is a very practical guy.)

Is this an example of "... software engineering of **large corporations** ..."? What about small, one- or two-person projects — does this definition work?

Still: someone's understanding, and decision about what is significant.

The GoF Book



Toward handbooks for software engineers.

Design Patterns, the “Gang of Four book” or GoF, is the **de facto** standard for OO design.

Pattern-Oriented Software Architecture, the PoSA book, is a part of a series aimed at large distributed systems.

Software Architecture is ...

~~*the highest level concept of a system*~~

the set of design decisions that must be made early in a project

— common wisdom

Does programming language count as architecture? By this definition, many would have to say “yes”. (Not me...)

We want to make some decisions — well — early in a project. But most often we don’t do a very good job.

Software Architecture is ...

~~*the highest level concept of a system*~~

~~*decisions that must be made early*~~

what the software architect works on.

Does programming language count as architecture? By this definition, many would have to say “yes”. (Not me...)

We want to make some decisions — well — early in a project. But most often we don’t do a very good job.

The software architect is ...

the person who works on the software architecture.

... another recursive decision.

Can this really be helpful? Perhaps...

Software Architecture is ...

... whatever the expert developers think is important.

The important stuff.

This changes from domain to domain, application to application, developer to developer, and even for an individual programmer over time.

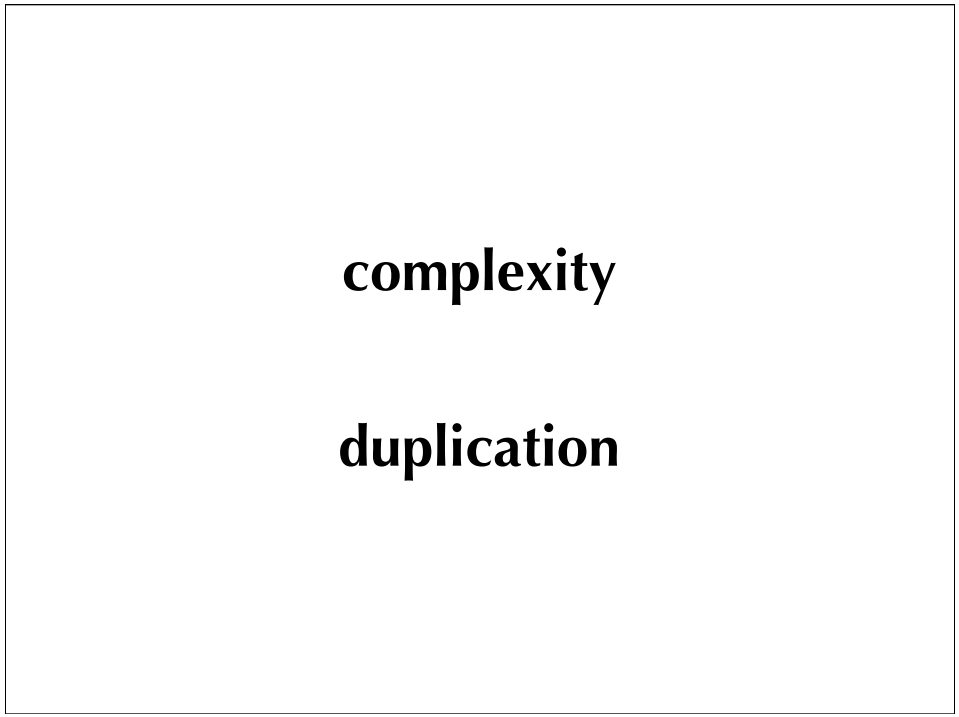
What is important enough to try to get right early?

Things that are **hard to change**!

Software Architecture is ...

... the parts of the system that the expert developers think will be hard to change later.

That's the important stuff, because when we get it wrong we have to live with it forever or battle to change it.



These are what make software hard to change, not any physical limitations.

The traditional approaches attack complexity by trying to **impose structure** up-front.

The agile approaches attack complexity by trying to make as many decisions as possible **reversible**.

different views on a system

logical view

structural view

code view

physical view

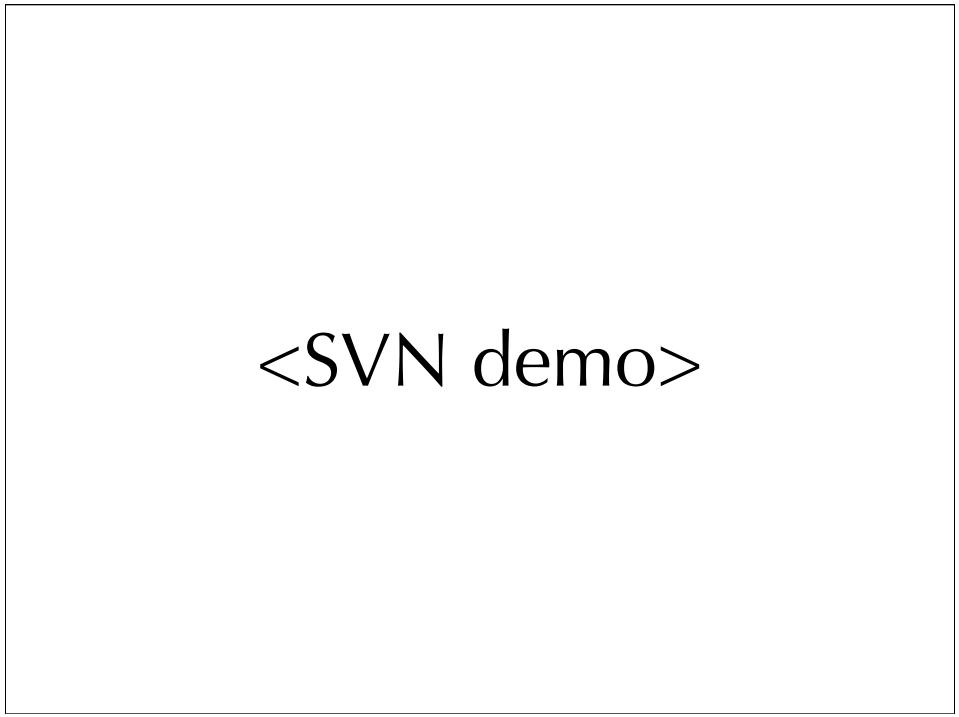
user view

concurrency view

data view

Data modeling has driven a lot of Software Engineering. Our previous 810:172 instructor was an expert in and proponent of data modeling as the backbone of big systems.

Object-oriented analysis and design talk about objects, which are in one sense data at the code level. But OO approaches are not about data modeling. They are about **behavior**, with data views as a subsidiary.



Thanks to Allyn, Nick, and Danny for sharing their expertise — and students' point of view!