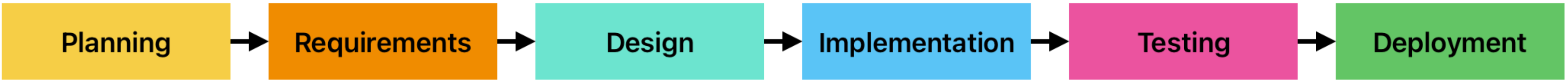
Agile Development

CS 346 Application Development

How to build software

People often think that building software is like building anything else, e.g., a car, or a refrigerator.

At first glance, this *seems* reasonable: software is something that you manufacture. Your project includes determining requirements, designing and building something. You might envision a process that looks something like this:



Planning

Requirements

Design

Implementation

Testing

Deployment

How to build software

Planning - “What are our goals?”, “What is the budget?”, “Who is working on it?”

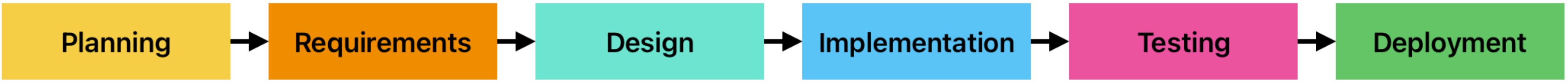
Requirements - “Who are our users?”, “What problem are we solving?”

Design - “What technical constraints exist?”, “What might it look like?”

Implementation - “How do we build it efficiently?”

Testing - “Does it meet specifications?”

Deployment - “How do we sell it and maintain it properly?”

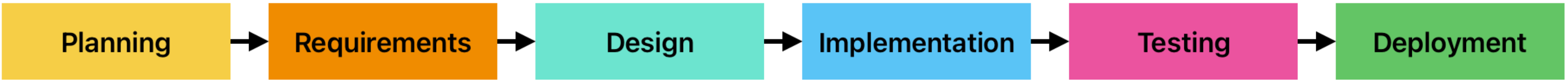


Process models

We use the term **process model** to describe this structure of activities.

> “A process model defines the complete set of activities that are required to specify, design, develop, test and deploy a product, and describes how they fit together.“

A **software process model** is a process model adapted to describe how we might build software systems. We also refer to a software process model as the **Software Development Lifecycle (SDLC)**.



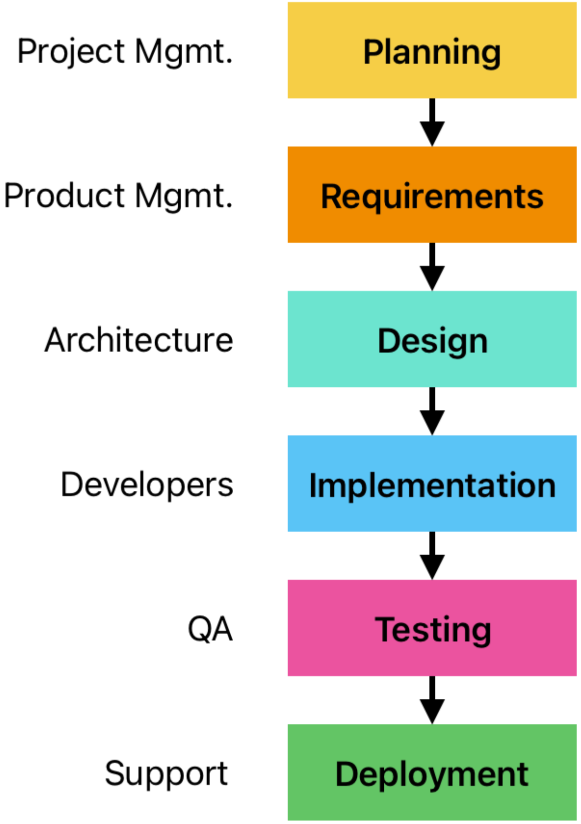
Planning → Requirements → Design → Implementation → Testing → Deployment

SDLC: Waterfall

In a 1970 paper, Winston Royce described a process model that envisions software production as a series of cascading steps.

He dubbed this this **Waterfall Model**, and described challenges associated with it:

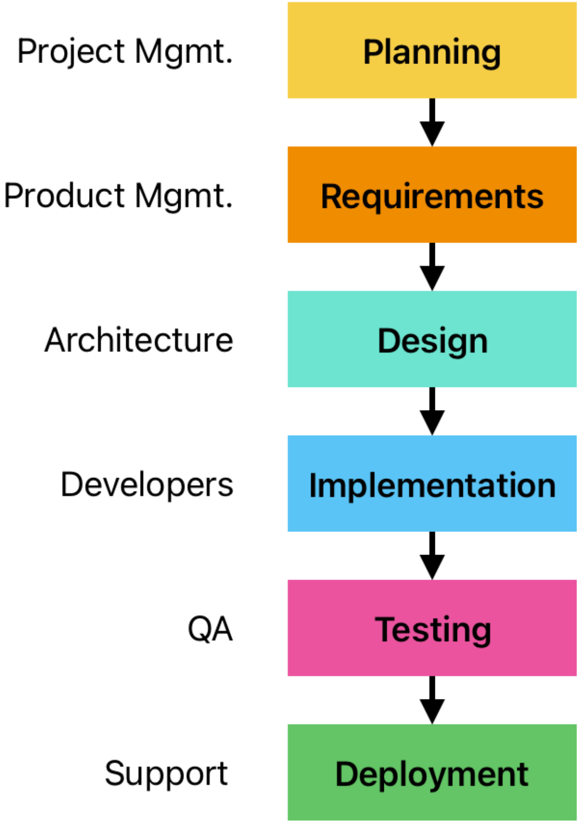
- Discrete steps that prescribe a specific order, with gatekeeping between them.
- Inability to collaborate between groups that “own” each step.
- Discouraged from “revisiting” earlier decisions.



Challenges

Why did Waterfall fail in practice?

- Customer priorities can change over a project, and requirements may need to be revisited.
- Your understanding of a problem will increase over time; you will uncover new data during design and implementation phases.
- Many of these activities should not be separated! e.g., testing & development.
- Building silos discourages collaboration. Cross functional teams are more effective.

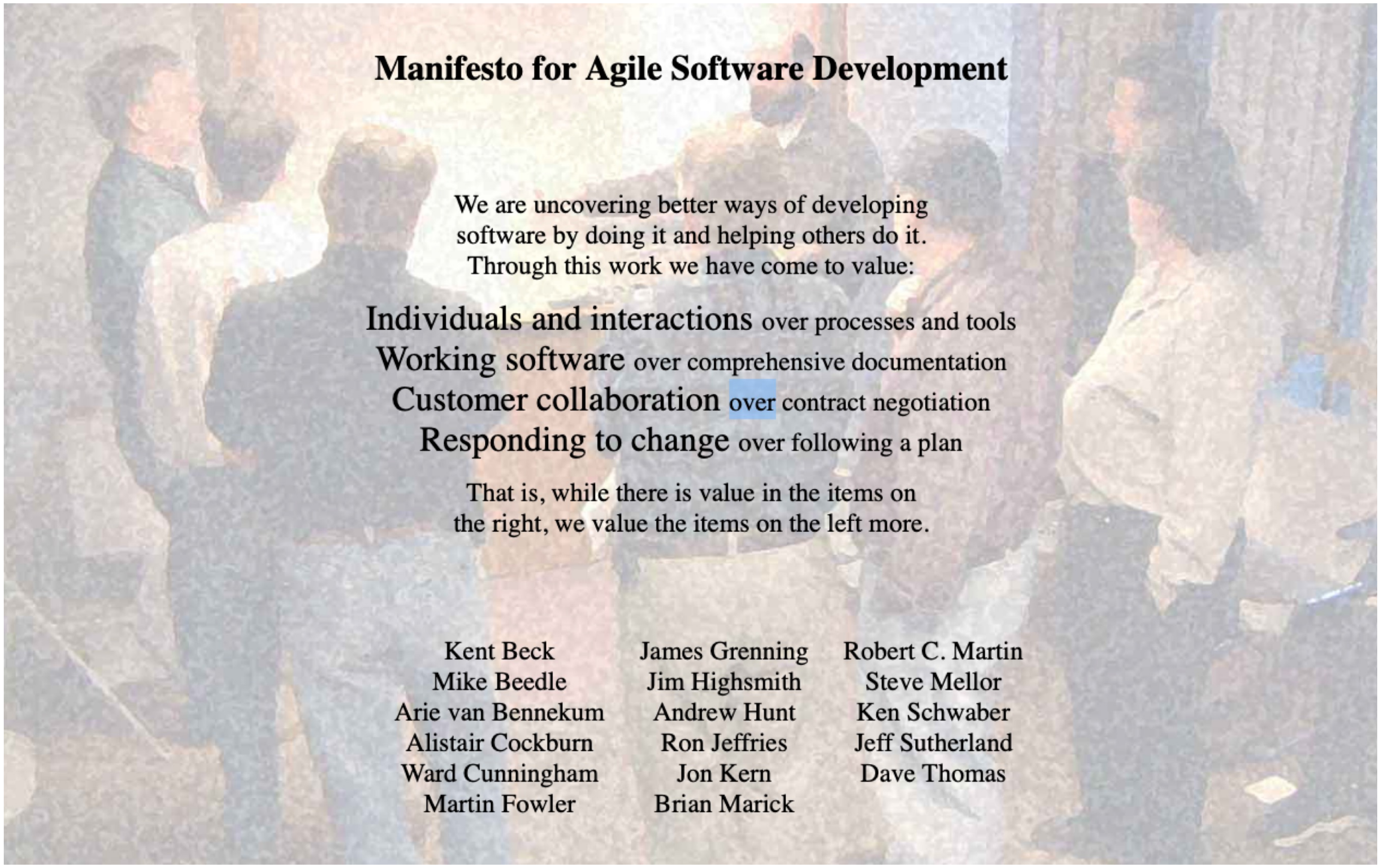


New Process Models

By the mid-1990s, there was a widespread recognition that this way of building software just didn’t work:

- Developers were frustrated by rigid processes/changing requirements.
- Business owners were frustrated by the inability to make changes to projects once they were past the requirements phase.
- Projects were being delivered late and/or over-budget.

Alternate models included: Extreme Programming (XP), Scrum, Lean, Rational Unified Process (RUP), Crystal Clear and many others.



Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

https://agilemanifesto.org/

The Agile Manifesto (2001)

Individuals and interactions (over processes and tools): Emphasis on communication with the user and other stakeholders.

Working software (over comprehensive documentation): Deliver small working iterations of functionality, get feedback and revise based on feedback. You will NOT get it right the first time.

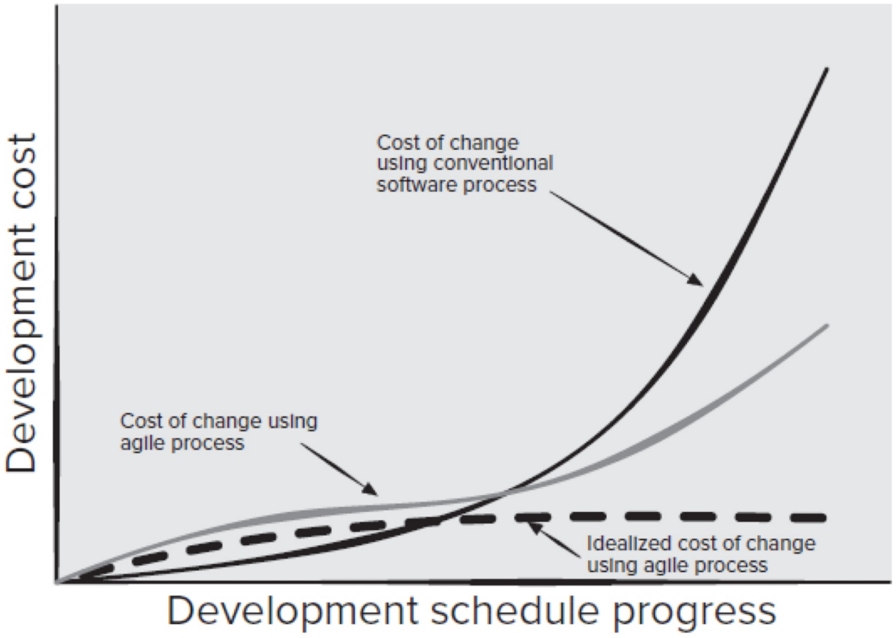
Customer collaboration (over contract negotiation): Software is a collaboration between you and your stakeholders. Plan on meeting and reviewing progress frequently. This allows you to be responsive and correct your course early.

Responding to change (over following a plan): Software systems live past the point where you think you're finished. Customer requirements will change as the business changes.

The importance of iterative development

The cost of change increases nonlinearly as a project progresses.

- Cost includes time, effort and money.
- The later you recognize a problem, or introduce a new requirement, the costlier it will be.
- Iterative approaches encourage you to make required changes earlier in the process, when the cost of doing so is lower.



What does iterative development look like?

Getting feedback at every stage of development.

- Identify incorrect requirements earlier, so that you don't waste time designing something that isn't needed.
- Identify poor designs earlier, before you waste time refining and polishing and implementing the wrong design.
- Identify failing tests earlier, so that you can correct them through design changes and not just hacking together a fix.

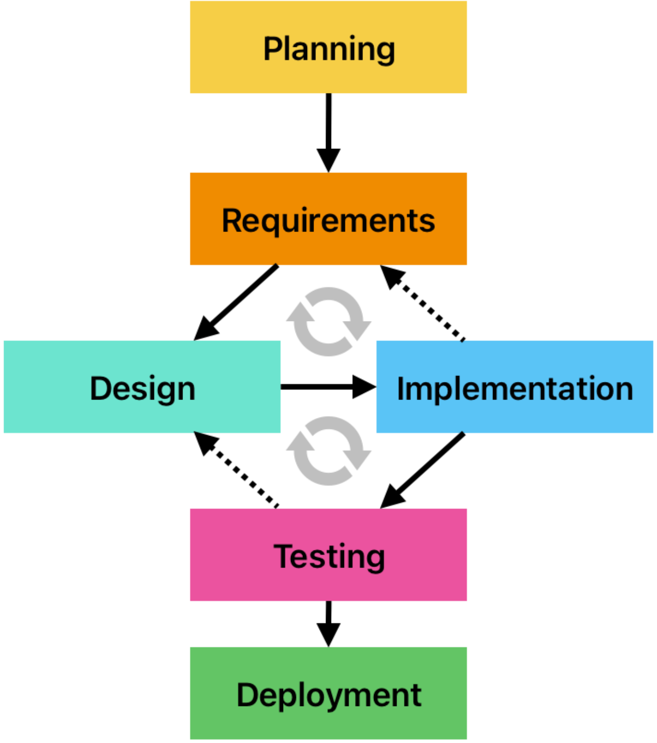
Focus on delivering one feature at-a-time and getting immediate feedback.

- Feedback from customer, development team, stakeholders.

Agile SDLC

This diagram represents the path for a *single feature*. We iterate over each of our features independently.

- Solid lines represent the “happy path” where your requirements, design and implementation all work as expected.
- Dotted lines suggest that you can loop-back if something isn’t working or needs to change e.g., issues with implementation may result in requirements changes; inability to test may force you to rethink an earlier design decision.

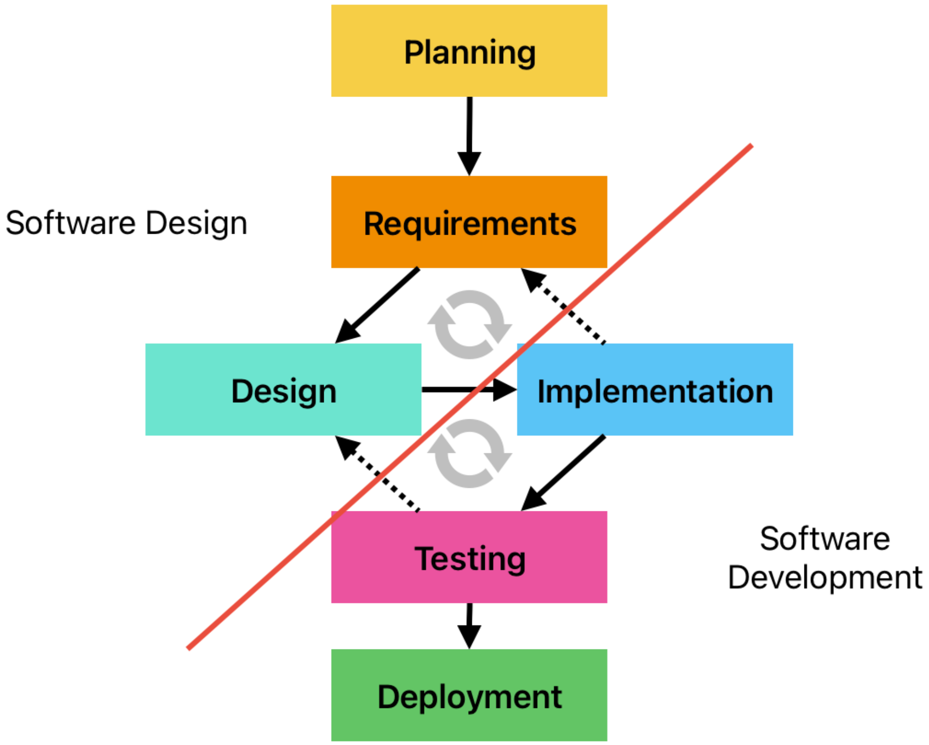


What’s next?

We’ll discuss activities and practices that are relevant to the steps in our SDLC.

We’ll break these into two parts:

- Software Design
  - Design Thinking lecture
- Software Development
  - Every other lecture…


Planning
Software Design
Requirements
Design
Implementation
Testing
Software Development
Deployment