

# Software Development Lifecycles

UW CSE 403 Software Engineering

# Today's Outline

---

- Software development lifecycles (SDLC)
  - What they are
  - Why are they needed
  - Recurring themes
  - Popular models and their tradeoffs
    - Traditional
    - Agile

# Software Engineering is ...

---

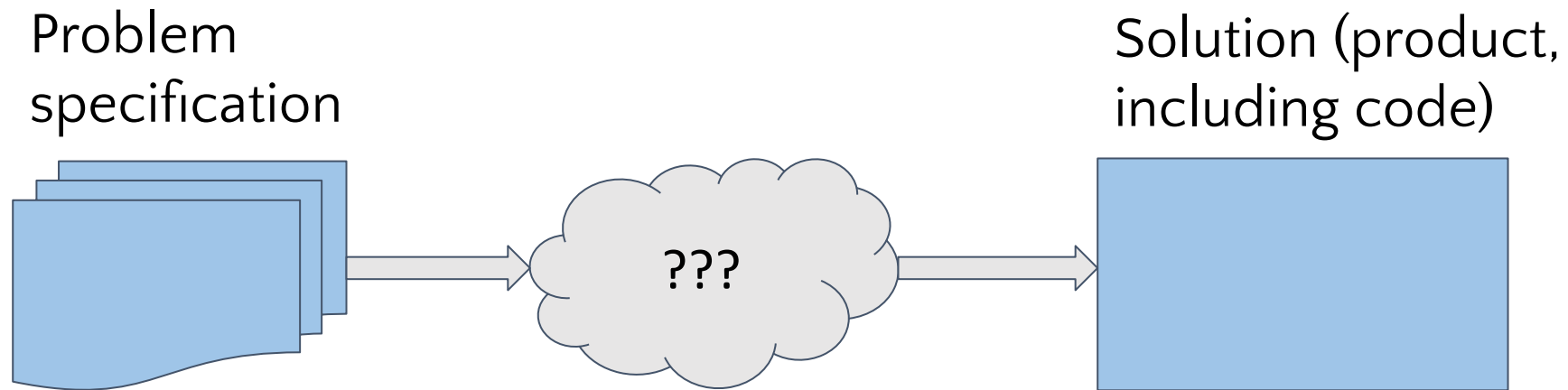
“An **engineering discipline** concerned with all aspects of **software production** from the early stages of system specification [requirements] through to maintaining [evolving] the system after it has gone into use.” — Ian Sommerville

Software Engineering tasks include:

- Requirements engineering
- Specification writing and documentation
- Architecture and design
- Programming
- Testing and debugging
- Deploying, operating, evaluating, refactoring and evolving
- Planning, teamwork and communication

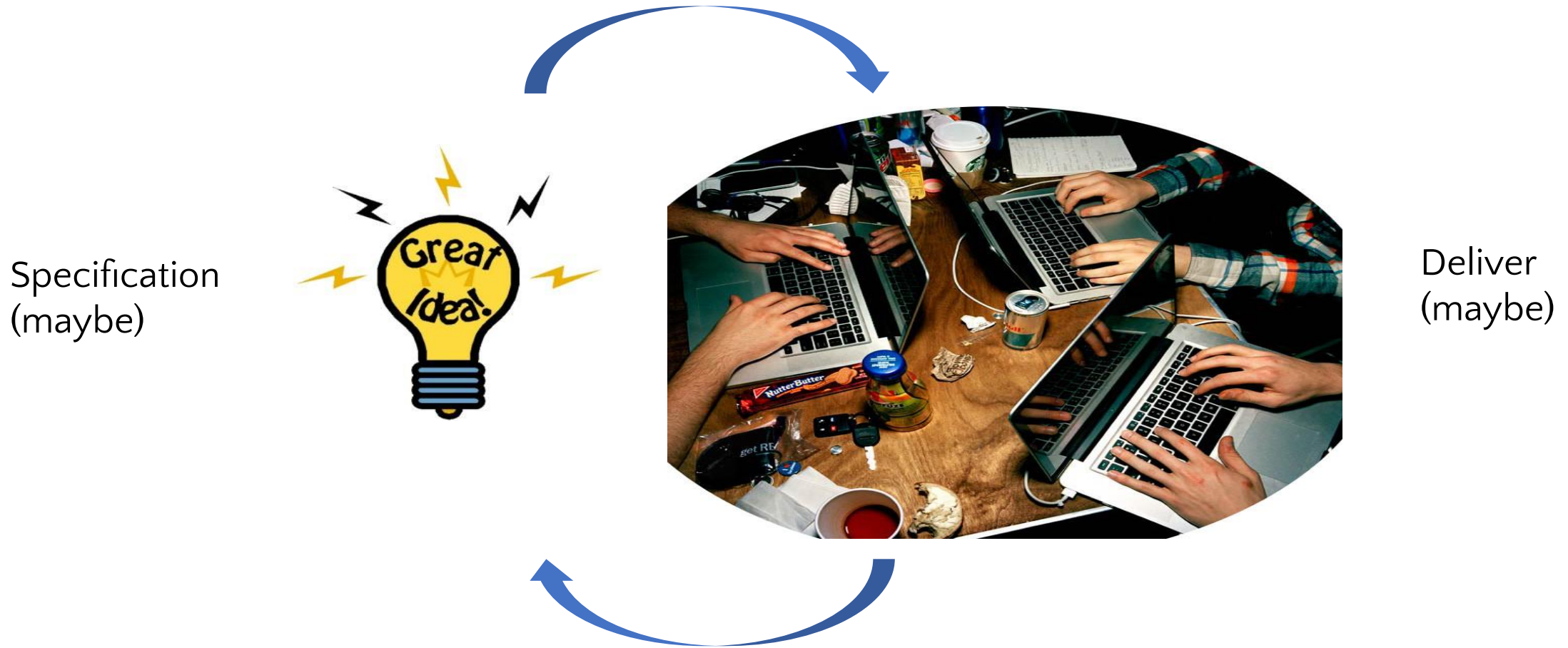
# The software development challenge

---



# One solution: Code and fix

---



# SDLC: Code and fix

---

## Pros:

- Little or no overhead – just dive in and develop, and see progress quickly
- Applicable *sometimes* for small projects, short-lived prototypes, and/or small teams

## Cons:

- <Over to you>

# SDLC: Code and fix

---

## Pros:

- Little or no overhead – just dive in and develop, and see progress quickly
- Applicable *sometimes* for small projects, short-lived prototypes, and/or small teams

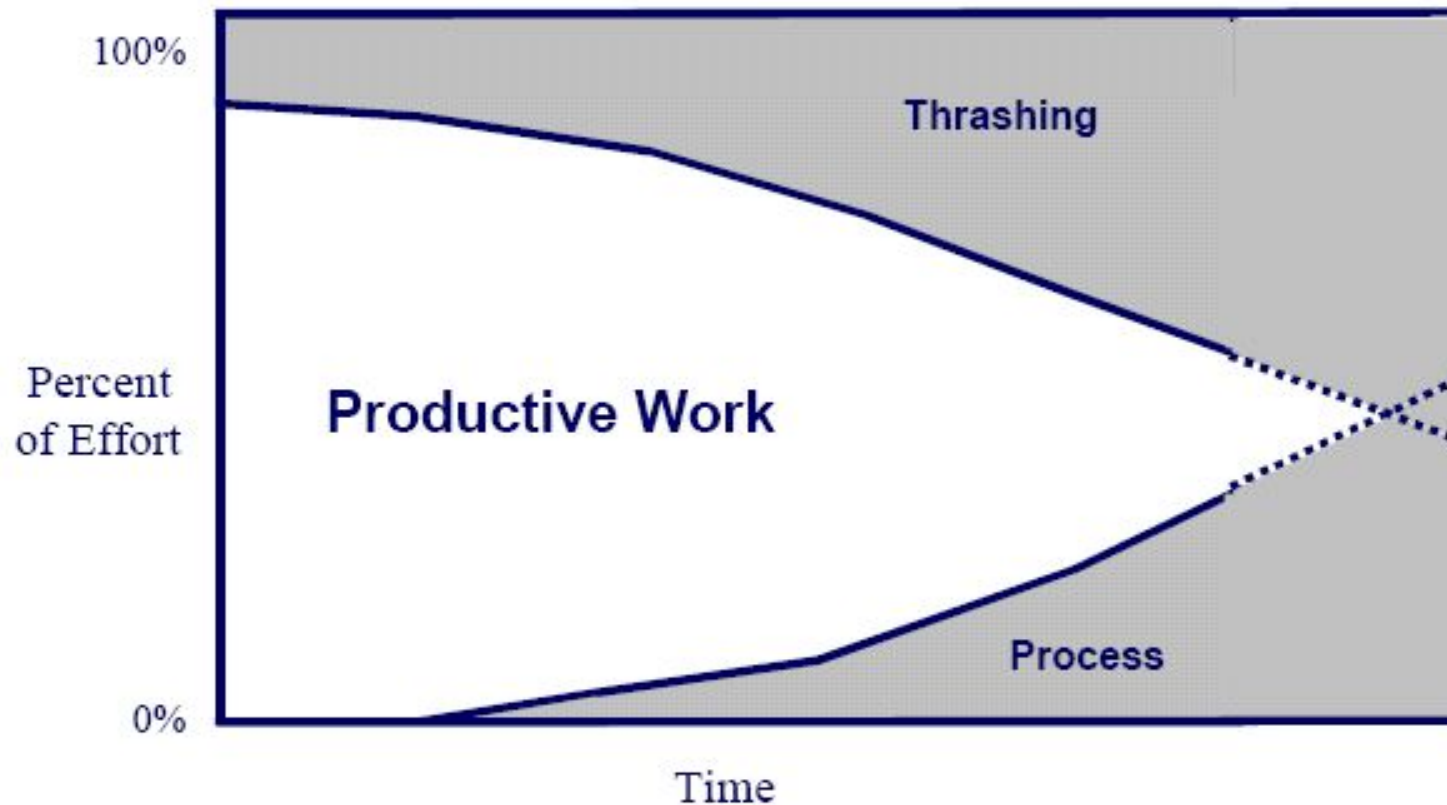
## Cons:

- **No way to assess progress, quality or risks**
- **Challenging to manage multiple developers – how synchronize your work**
- Harder to accommodate changes without a major design overhaul
- Unclear delivery of features (scope), timing, and support

# Code and fix productivity over time

---

Project with little attention on SDLC process

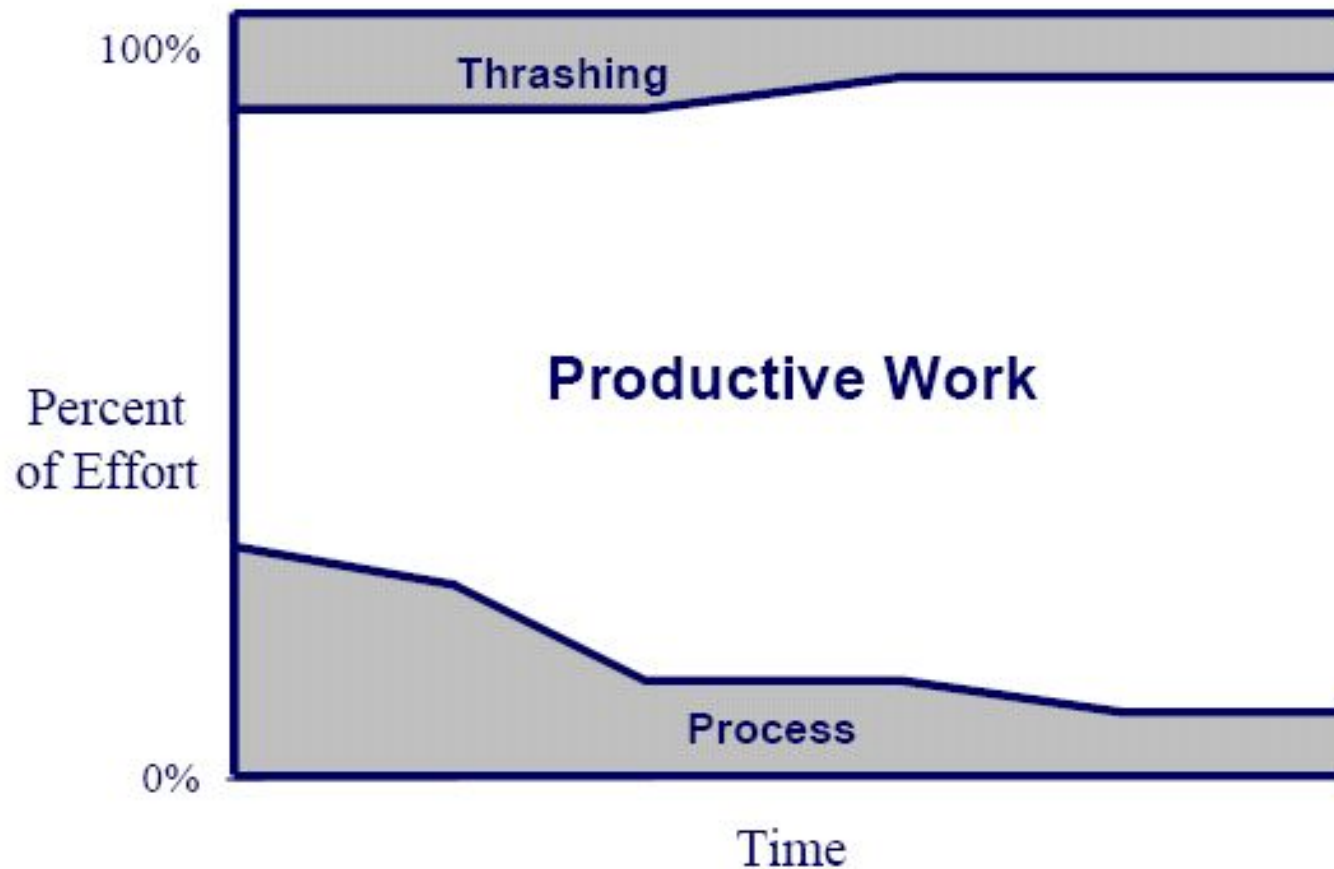


**Thrashing:** doing work but not making progress towards the goal

# SDLC process: productivity over time

---

Project with early attention to SDLC process



# Is a more structured SDLC necessary?

---

It establishes an order – provides a model – of software project events.

- It forces us to think of the “big picture” and follow steps so that we reach it without glaring deficiencies.
- Without it we may make decisions that are individually on target but collectively misdirected.
- It allows us to organize and coordinate our work as a team.
- It allows us to track progress and risks, and adjust as necessary.

# Recurring themes in SDLCs

---

A SDLC defines how to produce software through a series of stages.

## Common stages

- Requirements
- Architecture
- Design
- Implementation
- Testing/Verification
- Delivery/Release
- Maintenance

## Goals of each stage

- Define a clear set of actions to perform
- Produce tangible (trackable) items
- Allow for work revision
- Plan actions to perform in the next stage

**Key question:** how to **combine** the stages and in what **order**

# Today's Outline

---

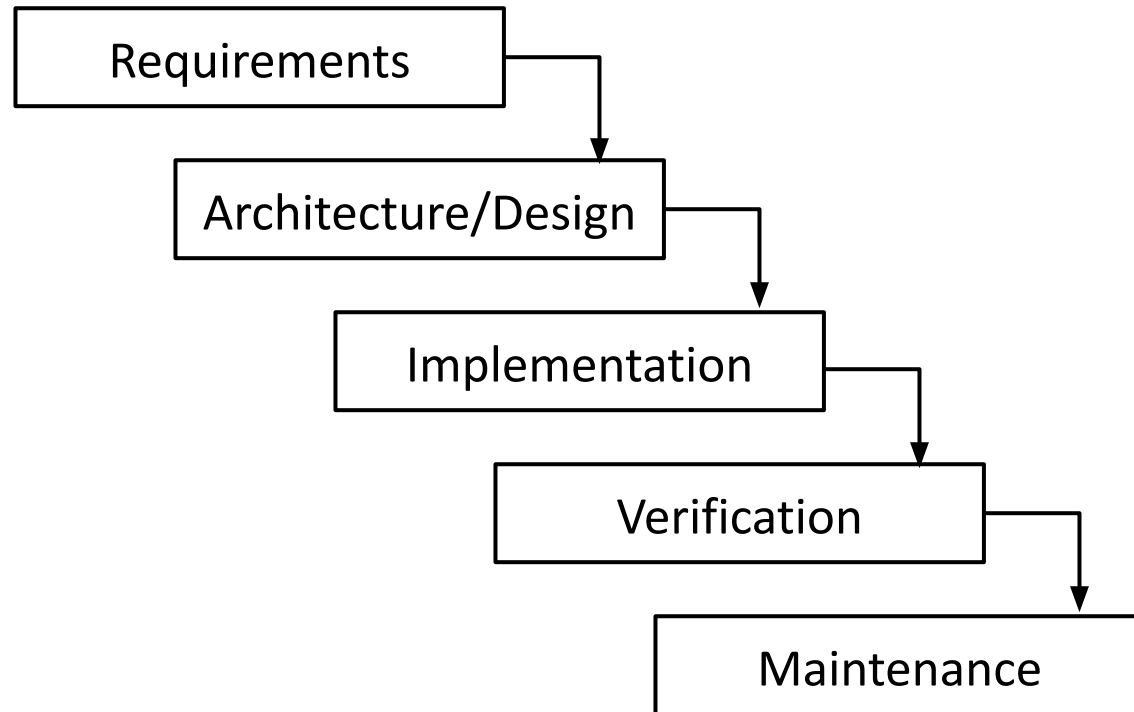
- Software development lifecycles (SDLC)
  - What they are
  - Why are they needed
  - Recurring themes
  - **Popular models and their tradeoffs**
    - Waterfall model
    - Prototyping
    - Spiral model
    - Staged delivery
    - Agile (XP, Scrum)



All have the same goal – deliver high quality software, on time, meeting the customer's needs

# SDLC: Waterfall model

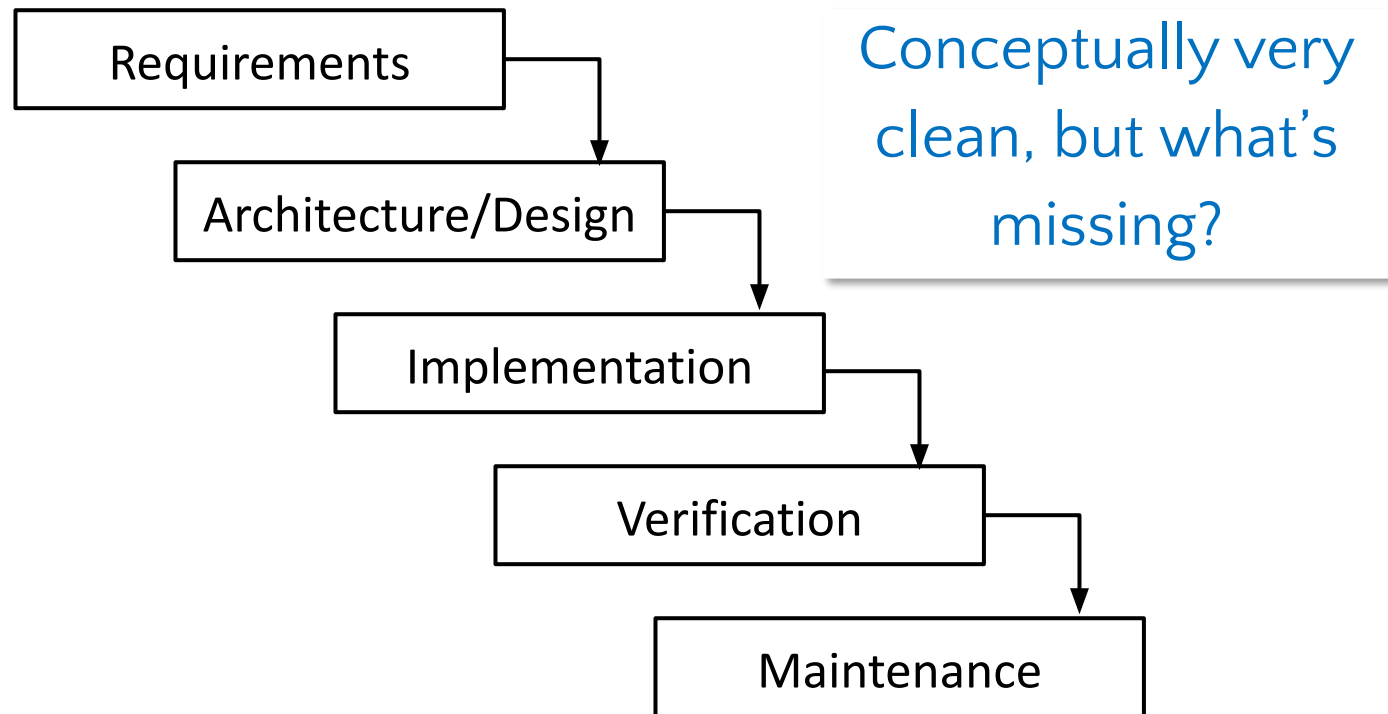
---



- Top-down approach
- Sequential, non-overlapping activities and steps
- Each step is signed off on and then frozen
- Most steps result in a final document

# SDLC: Waterfall model

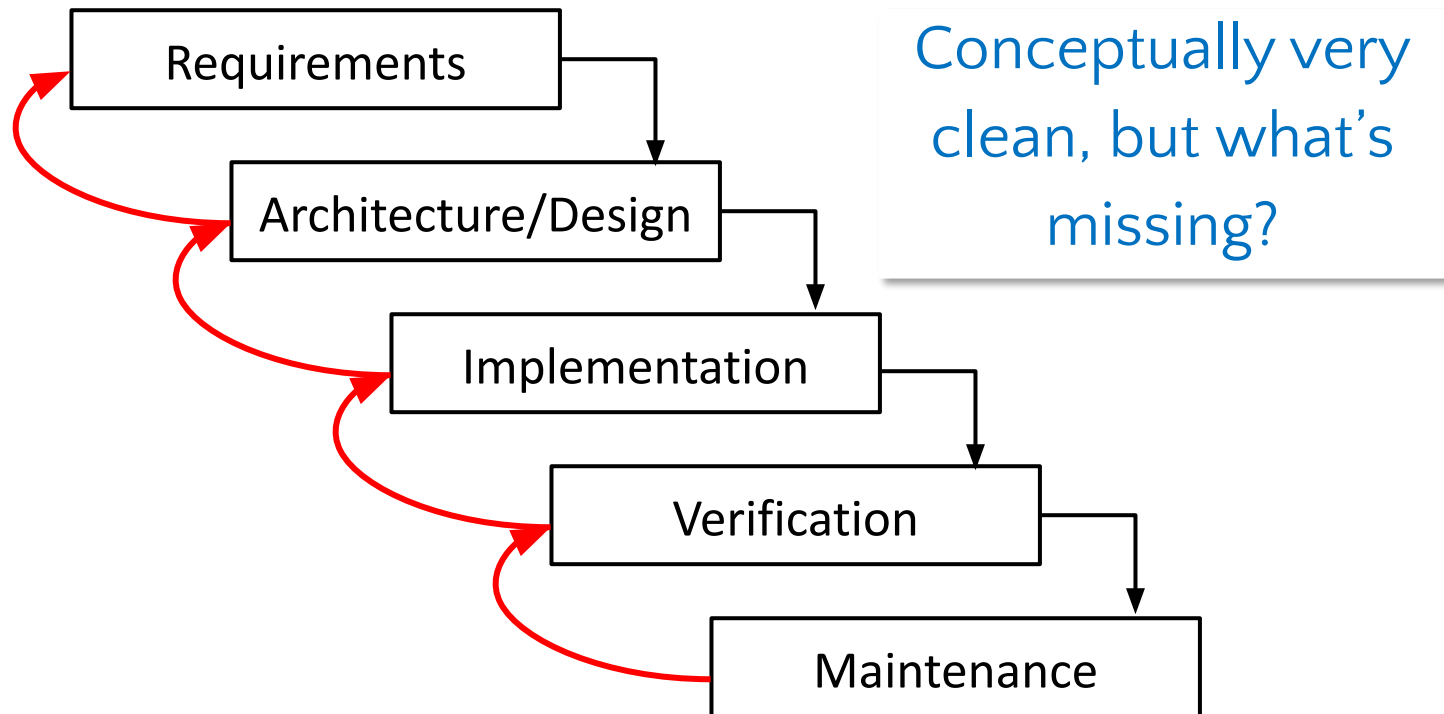
---



- Top-down approach
- Sequential, non-overlapping activities and steps
- Each step is signed off on and then frozen
- Most steps result in a final document

# SDLC: Waterfall model

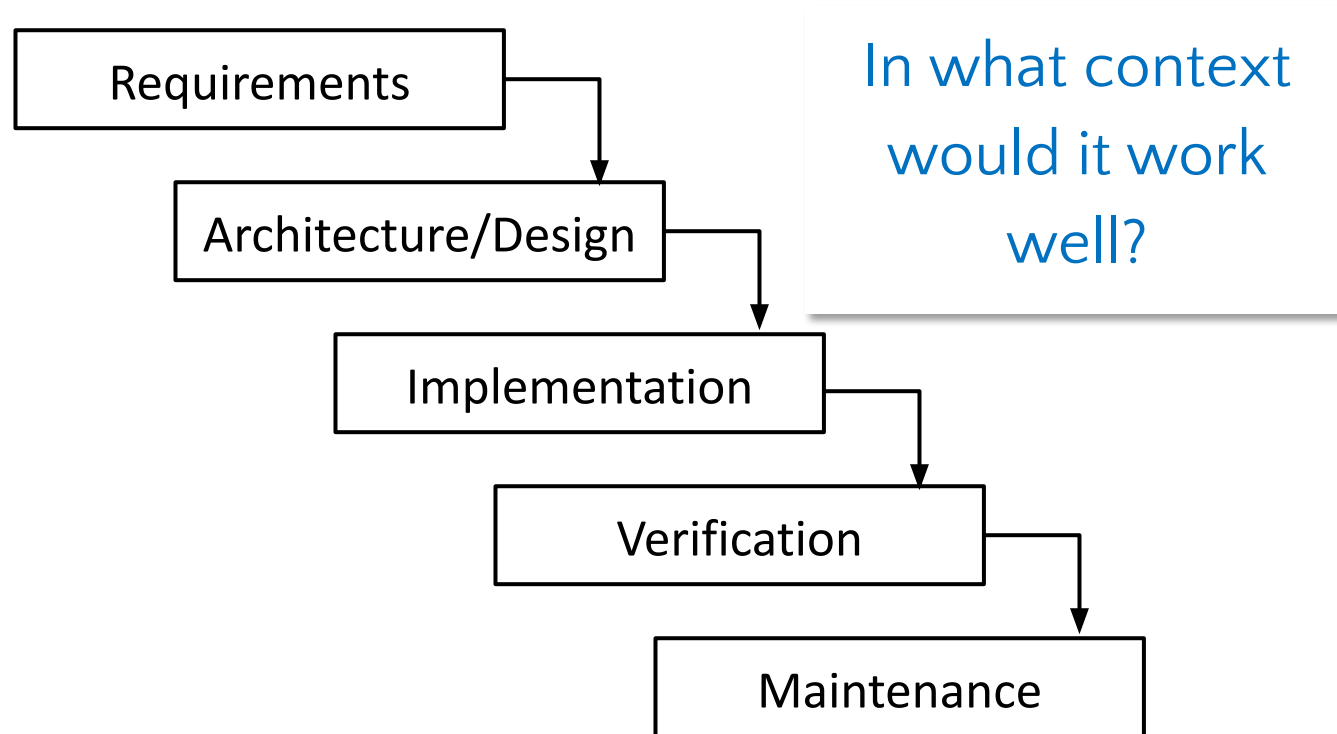
---



- Top-down approach
- Sequential, non-overlapping activities and steps
- Each step is signed off on and then frozen
- Most steps result in a final document
- Backsteps to correct mistakes

# SDLC: Waterfall model

---



- Top-down approach
- Sequential, non-overlapping activities and steps
- Each step is signed off on and then frozen
- Most steps result in a final document

# Honeywell's Flight Management System Selected By Airbus

Honeywell's solution will address the avionics needs of the Airbus A320, A330 and A350 aircraft fleet

Ahjay Rai

May 19, 2022



Their SDLC is waterfall-like due to the upfront and regulated requirements

**U.S. FOOD & DRUG  
ADMINISTRATION**

Home / Medical Devices / Device Advice: Comprehensive Regulatory Assistance / Overview of Device Regulation

## Overview of Device Regulation

[Share](#) [Tweet](#) [LinkedIn](#) [Email](#) [Print](#)

### Overview of Device Regulation

A History of

### Introduction

FDA's Center for Devices and Radiological Health (CDRH) is responsible for regulating firms who manufacture, repackage, relabel, and/or import medical products in the United States. In addition, CDRH regulates radiation-emitting products (medical and non-medical) such as lasers, x-ray equipment, microwave ovens and color televisions.

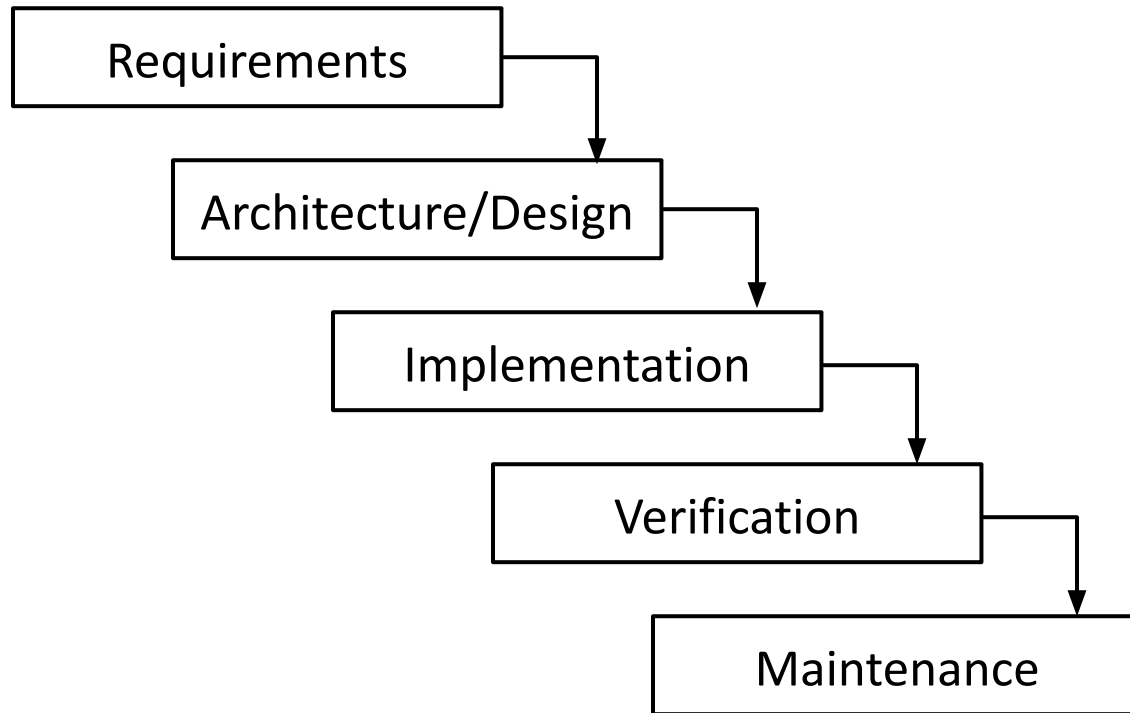
[Electronic Products](#)

Cont  
of:  
09/04  
Regu  
Prod  
Medi  
Radi  
Prod



# Waterfall: pros and cons

---

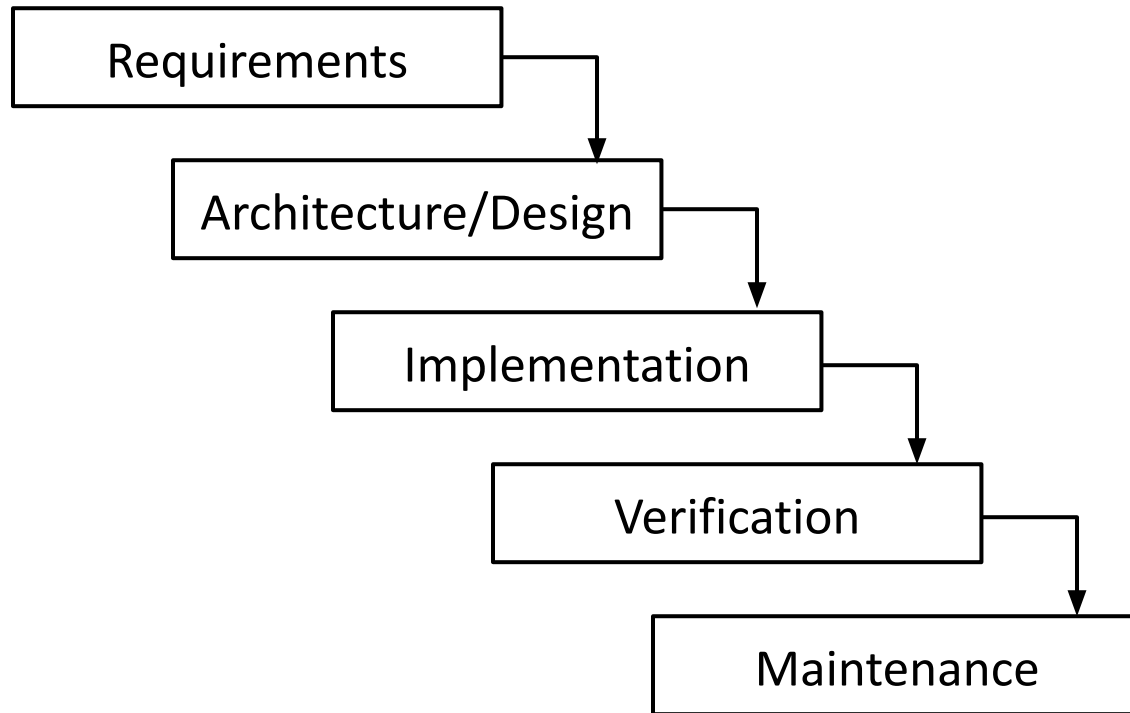


Pros:

Cons:

# Waterfall: pros and cons

---



## Pros:

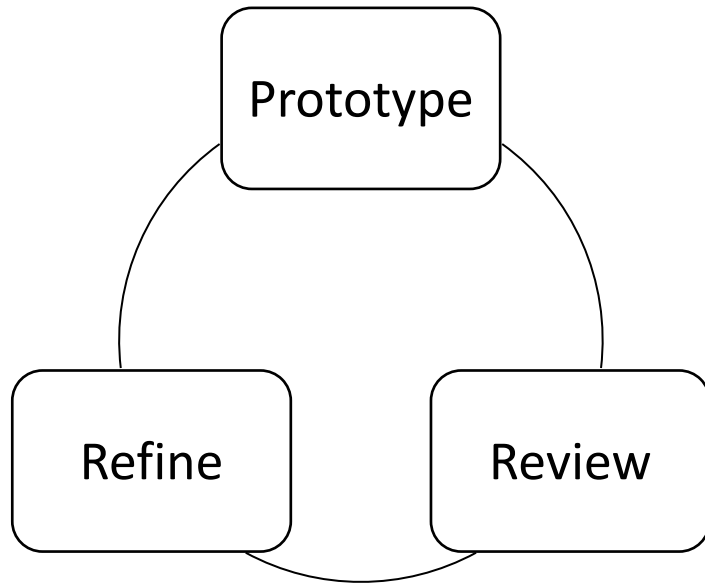
- Simple to understand
- Promotes common dialogue
- Highly regulated deliverables

## Cons:

- Hard to do all the planning upfront
- Inflexible – changes are expensive
- Test and integration come late – fixes are expensive
- Final product may not match the customer's needs

# SDLC: Prototyping

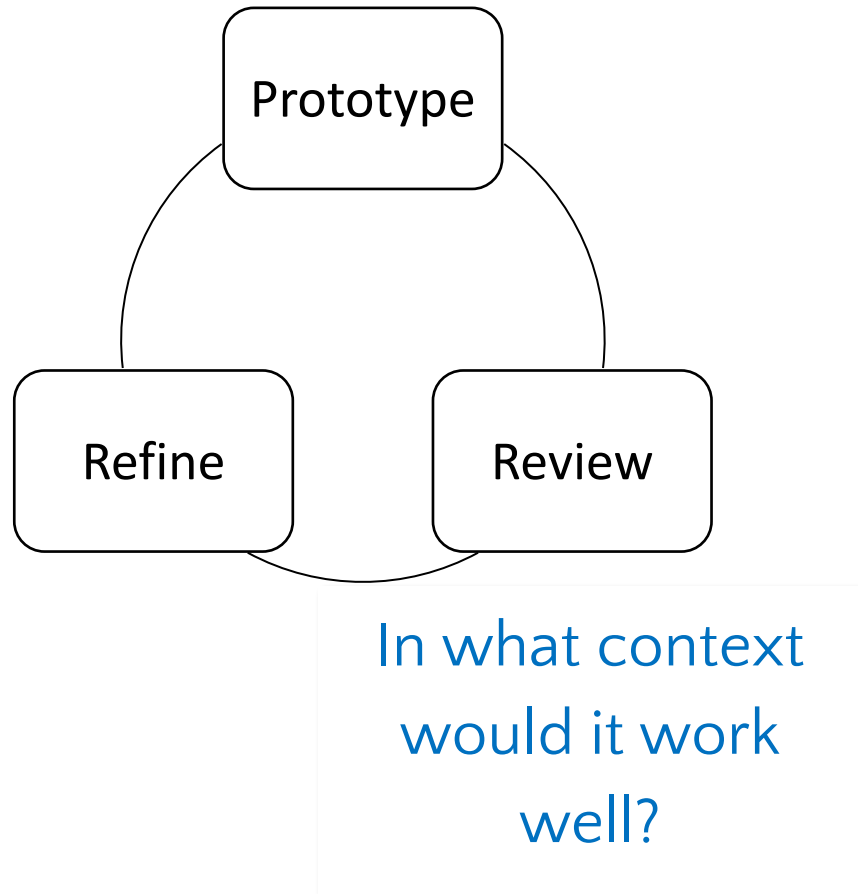
---



- Problem domain or requirements not well defined or understood
- Create small implementations of requirements that are least understood
- Requirements are “explored” before the product is fully developed
- Developers (and customers) gain experience when developing the product
- Prototype can evolve to the real product or can serve to be a learning tool only

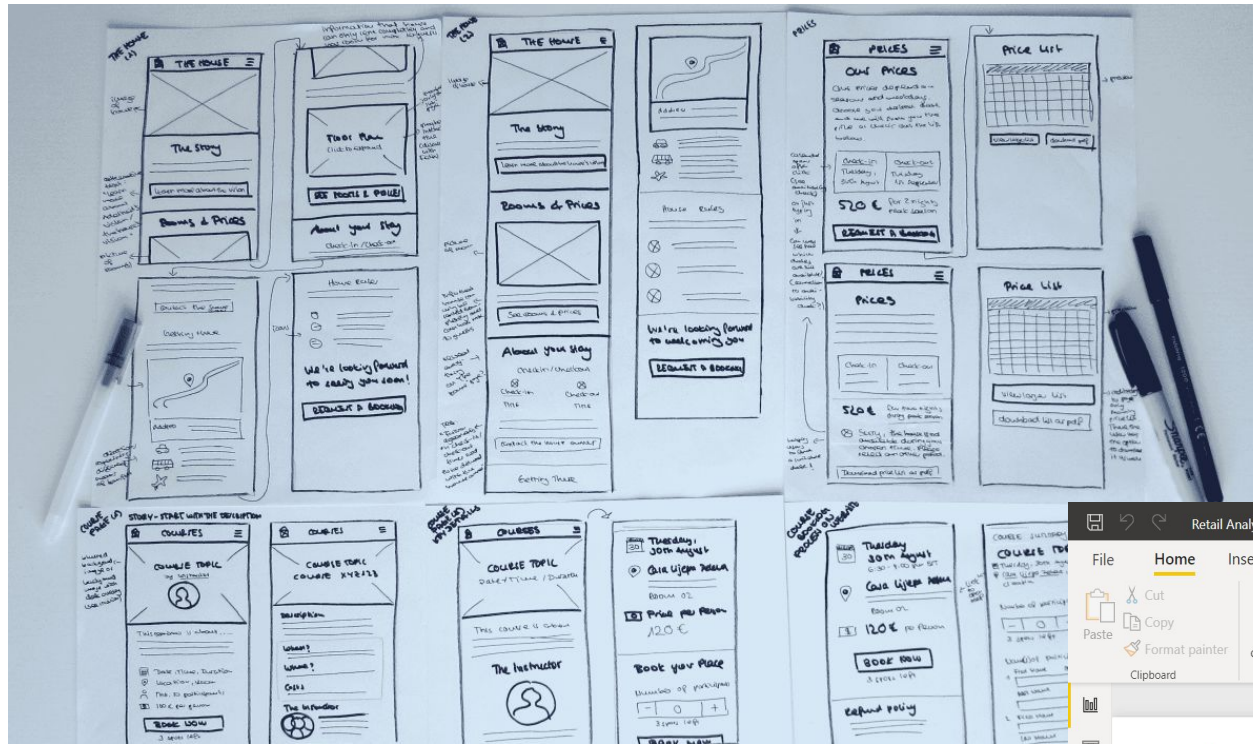
# SDLC: Prototyping

---

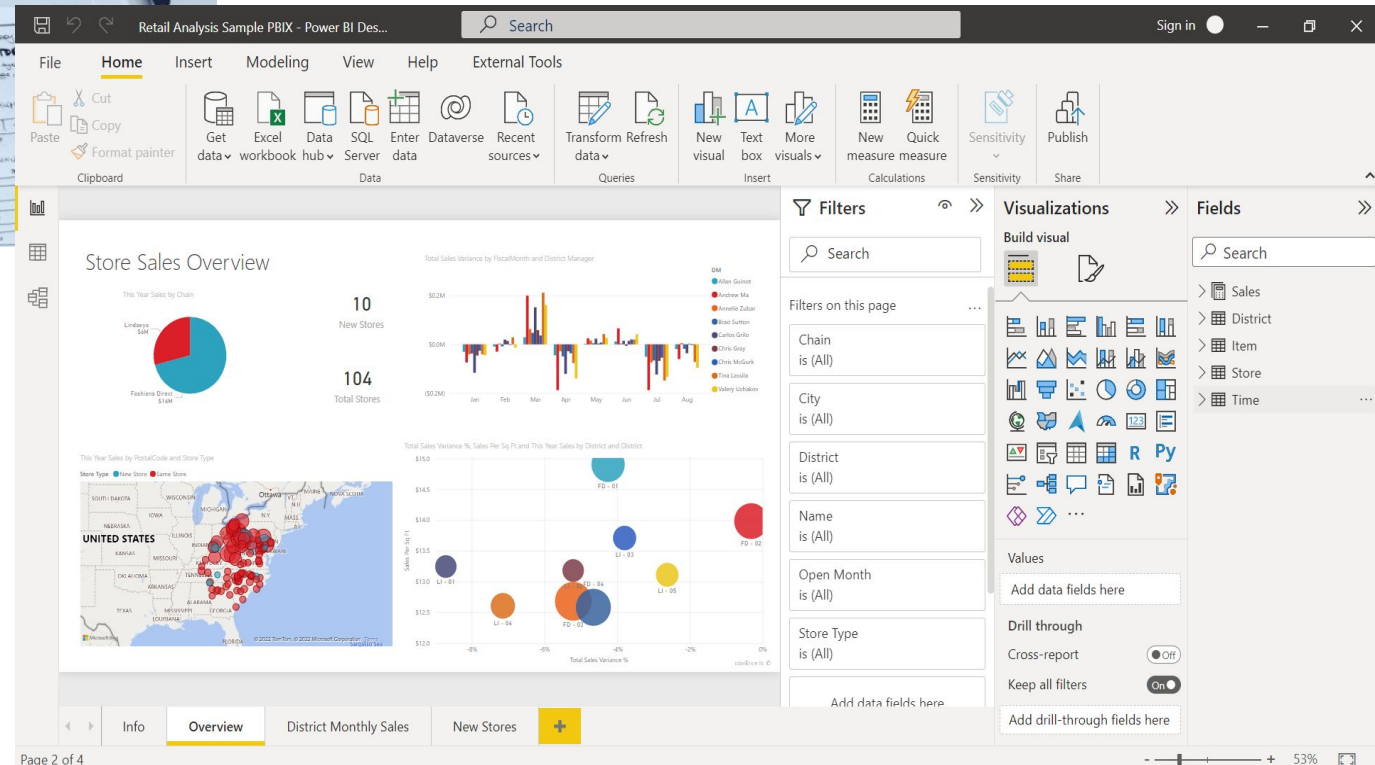


- Problem domain or requirements not well defined or understood
- Create small implementations of requirements that are least understood
- Requirements are “explored” before the product is fully developed
- Developers (and customers) gain experience when developing the product
- Prototype can evolve to the real product or can serve to be a learning tool only

# UI prototyping is popular



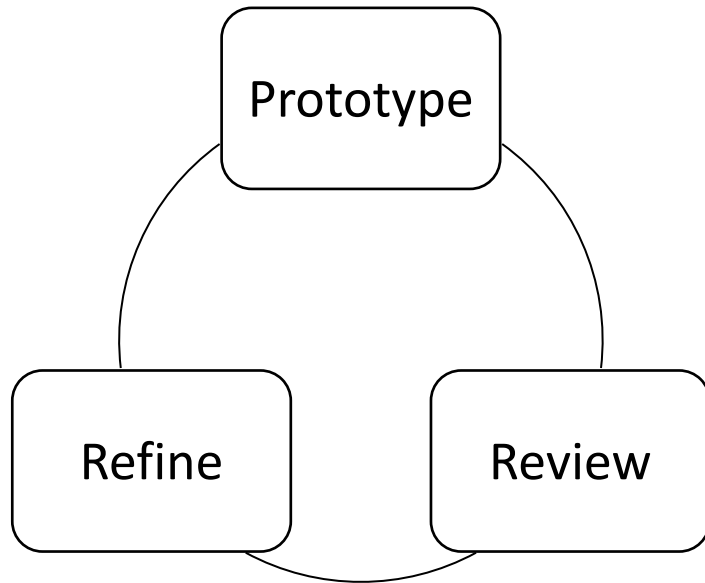
<https://internetdevels.com/blog/what-is-website-prototype-how-build-website-prototype>



<https://learn.microsoft.com/en-us/power-bi/fundamentals/desktop-what-is-desktop>

# Prototyping: pros and cons

---

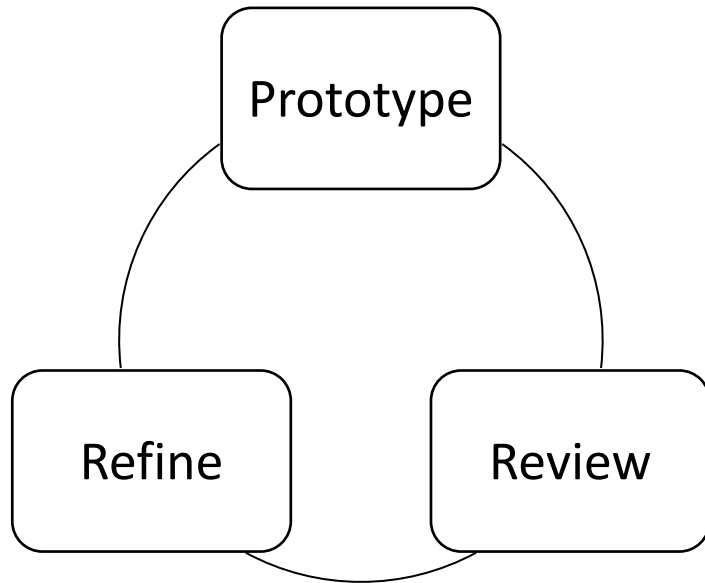


Pros:

Cons:

# Prototyping: pros and cons

---



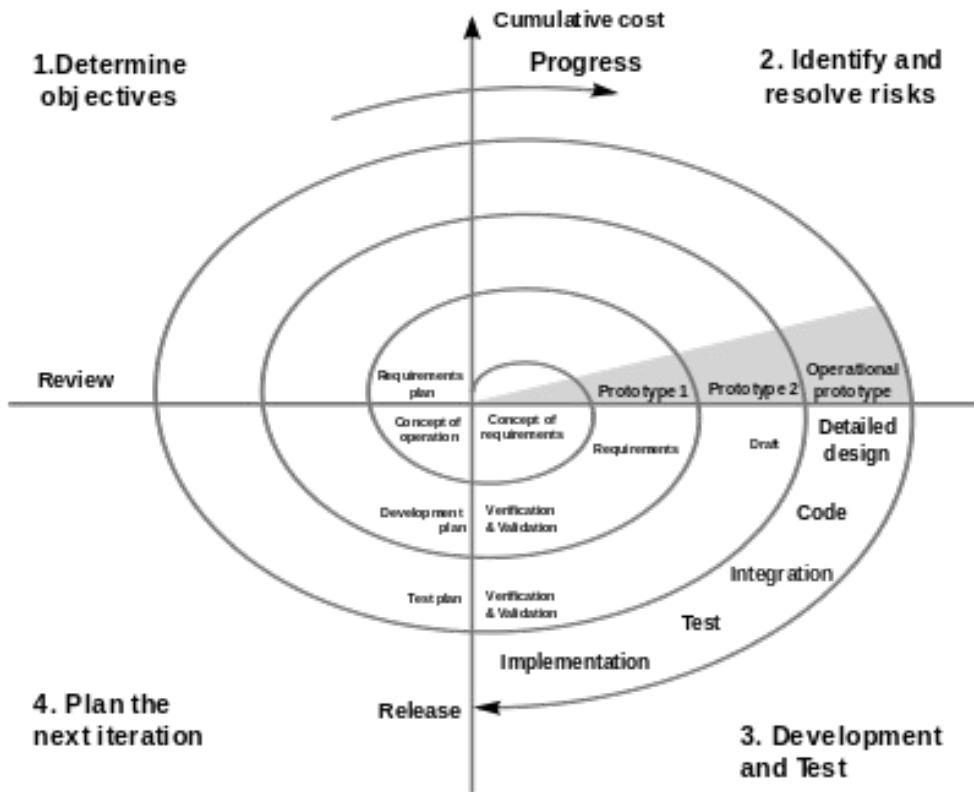
## Pros:

- Client involvement and early feedback
- Improves requirements and specifications
- Reduces risk of developing the “wrong” product

## Cons:

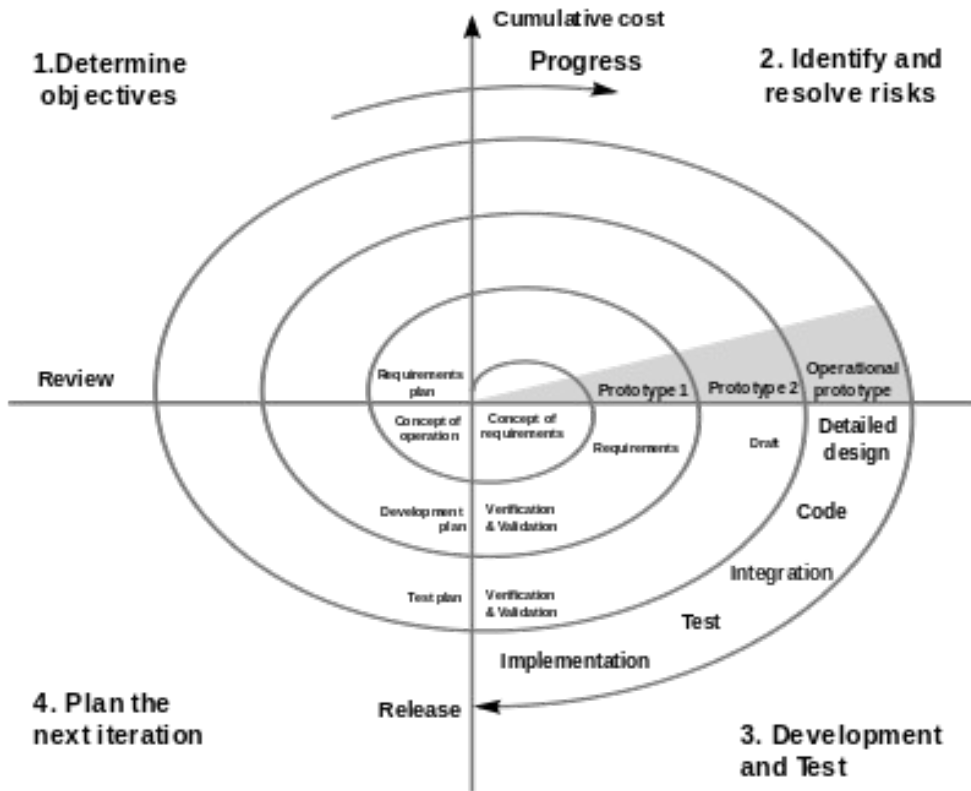
- Time/cost for developing may be high
- Hard to commit what will be delivered and when
- May end up evolving a poor choice (limit thinking holistically)

# SDLC: Spiral Model



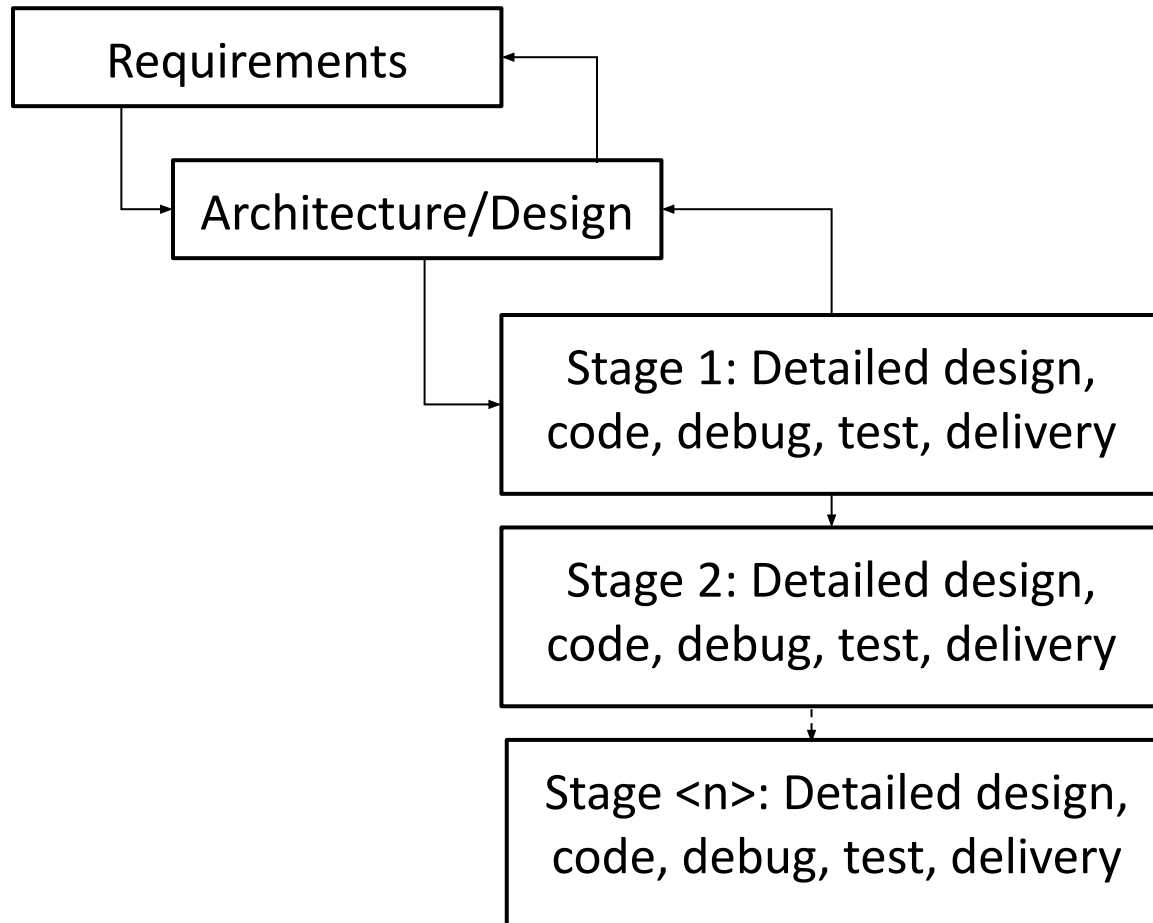
- Incremental/iterative model
- Iterations called spirals
- Repeat these activities:
  - Determine objectives (reqs)
  - Risk analysis
  - Develop and test
  - Plan
- Phased reduction of risks (address high risks early)

# SDLC: Spiral Model



- Interesting to us as it's a **precursor to agile models**
- Software development is based on **iteration**, using “**risk reduction**” as the criterion to prioritize activities at each iteration

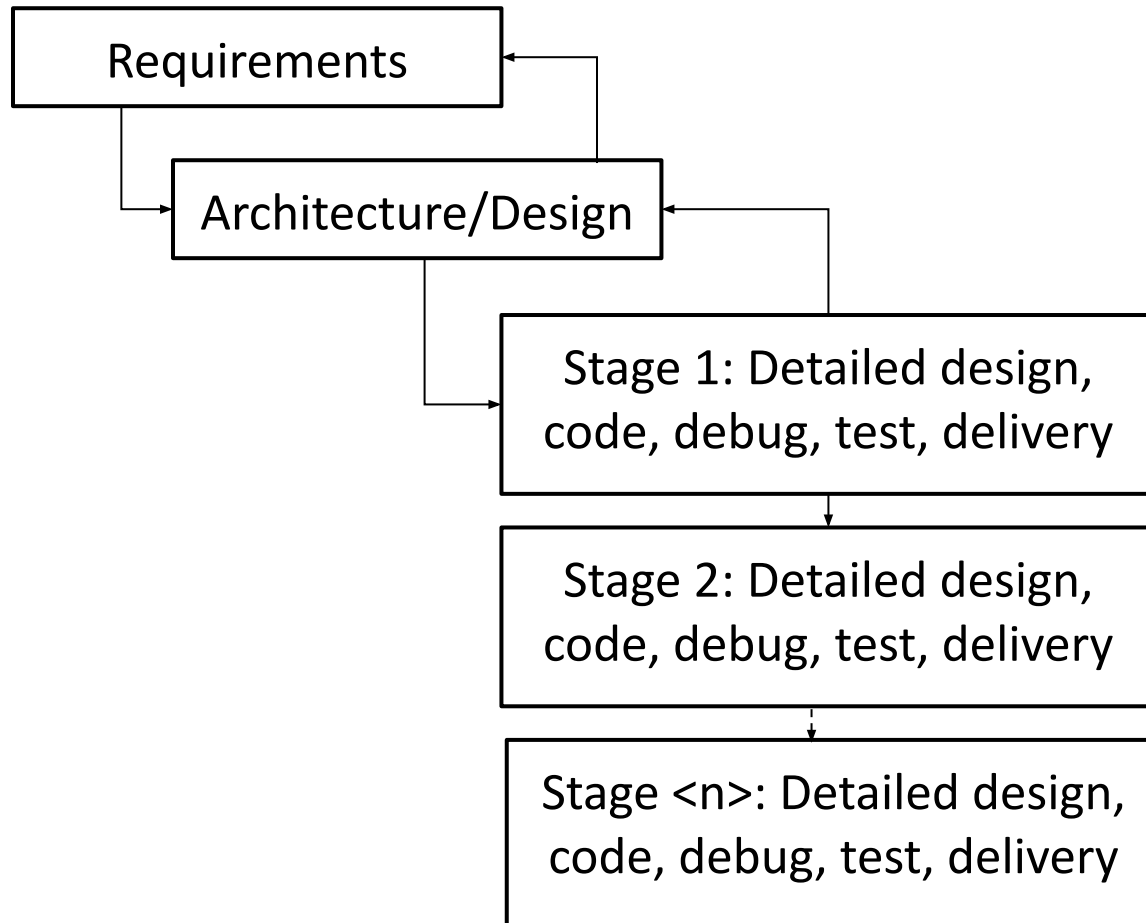
# Staged Delivery: one of many variants 🧠



- Waterfall-like planning upfront then spiral/agile-like short release cycles

# Staged Delivery: pros and cons

---

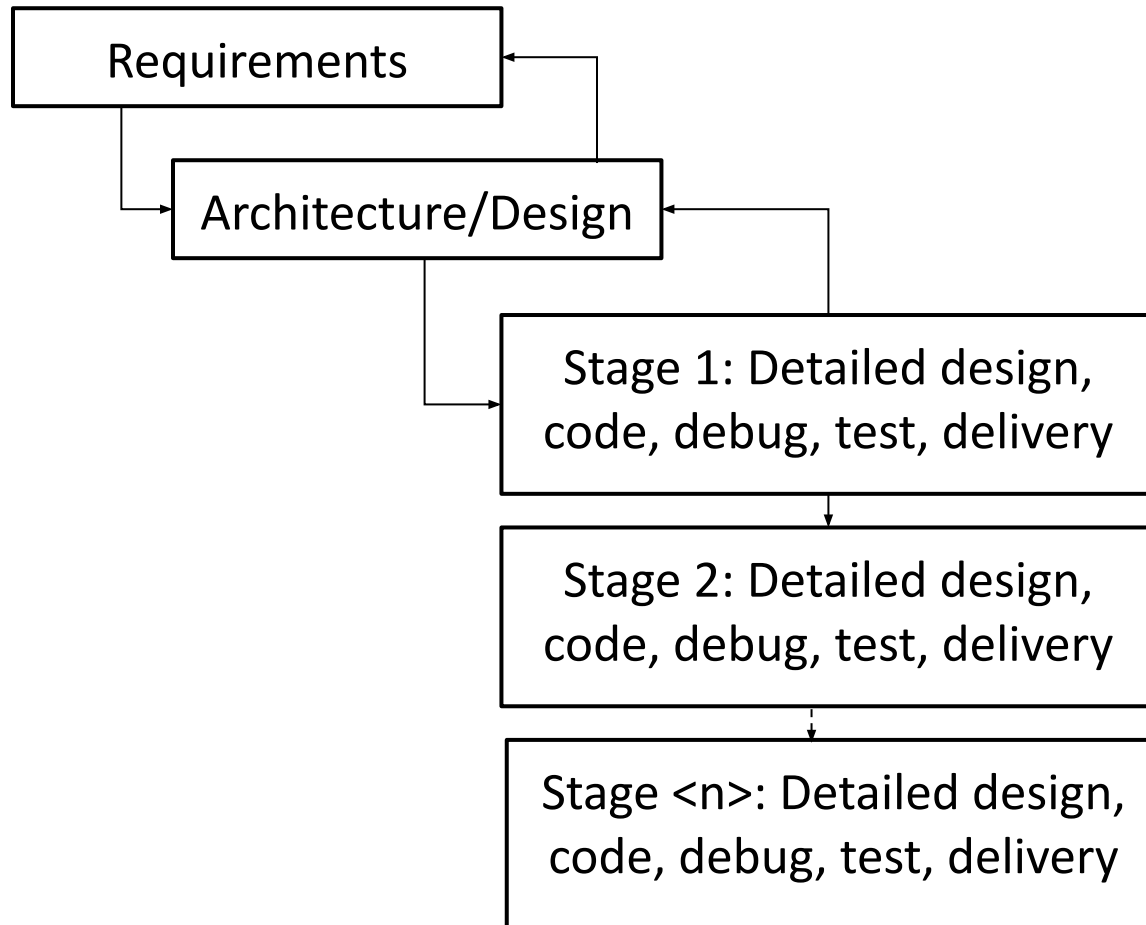


- Pros:

- Cons:

# Staged Delivery: pros and cons

---



- Pros:

- Can ship at the end of any release cycle
- Intermediate deliveries show progress, satisfy customers, and lead to feedback
- Problems are visible early

- Cons:

- Requires tight coordination
- Product must be decomposable
- Extra releases cause overhead

# Today's Outline

---

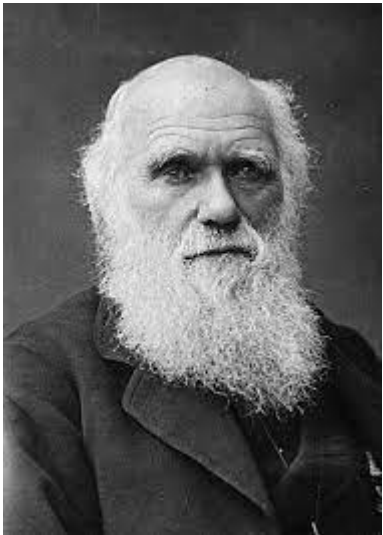
- Software development lifecycles (SDLC)
    - What they are
    - Why they are needed
    - Recurring themes
    - **Popular models and their tradeoffs**
      - Waterfall model
      - Prototyping
      - Spiral model
      - Staged delivery
      - Agile (XP, Scrum)
- 
- Traditional models

# Agile models

---

What is Agile all about?

Premise: the world is uncertain, and we must be flexible and responsive to changes



*There is nothing permanent except change. -Heraclitus  
(Greek philosopher)*

*It is not the strongest or the most intelligent who will  
survive but those who can best manage change. -Charles  
Darwin (English naturalist)*



# Agile Manifesto

---



Agile Manifesto (<http://agilemanifesto.org/>):

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

*While there is value in the items on the right,  
we value the items on the left more.*

# The Agile Manifesto (12 points)

---

Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.

Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.

**Deliver** working software **frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must **work together daily** throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

**Working software** is the primary **measure of progress**.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to **maintain a constant pace indefinitely**.

Continuous attention to **technical excellence and good design** enhances agility.

**Simplicity**—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from **self-organizing teams**.

At regular intervals, the team reflects on how to become more effective, then tunes and **adjusts its behavior** accordingly.

# Agile models

---

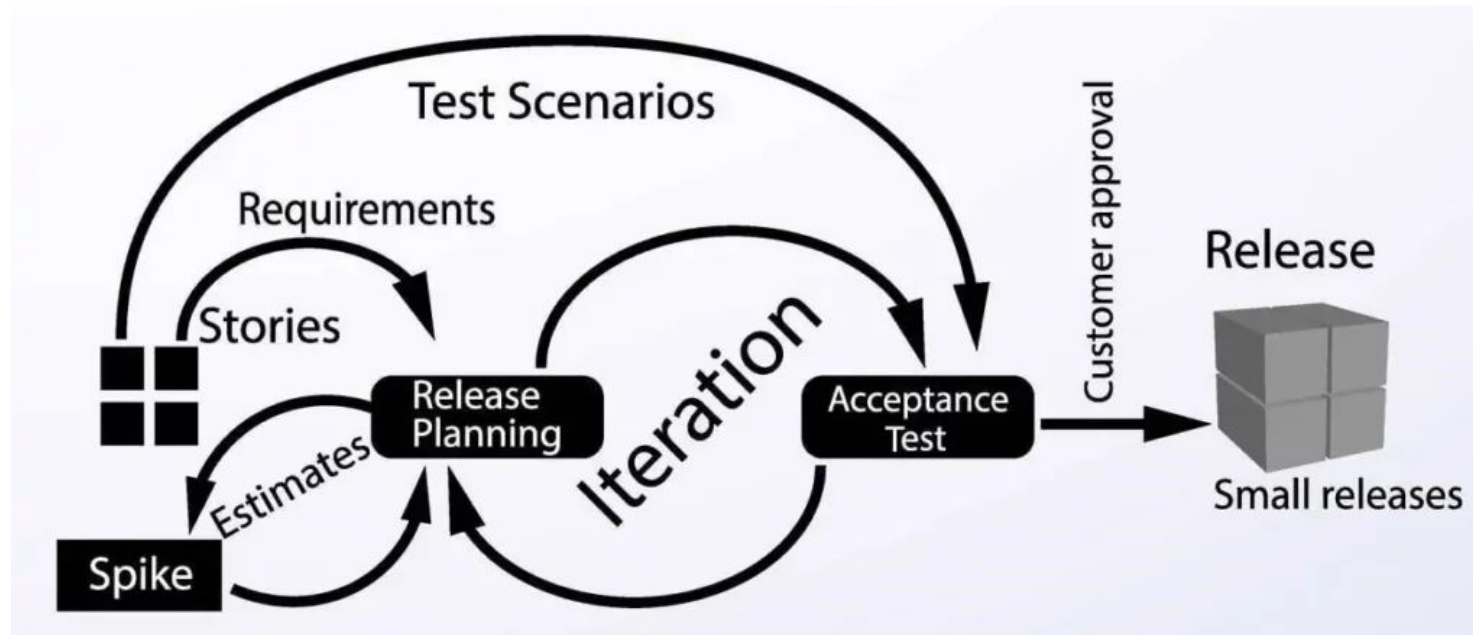
“Agile software development” is a general term for frameworks and practices outlined in the Agile Manifesto

## Agile models

- Aim to deliver a high-quality product to the customer as fast as possible
- Focus on simplicity, continuous testing, integration
- Incremental and frequent delivery of working software
- Continuous customer involvement
- Expect requirements to change

<http://agilemanifesto.org/principles.html>

# Agile SDLC: Extreme Programming (XP)



<https://www.nimblework.com/agile/extreme-programming-xp/>

- XP emphasizes how engineers should work – good practices taken to an extreme
- Examples:
  - Continuous testing and integration
  - 10-minute build
  - Constant discussions with customers
  - Full flexibility to change requirements anytime
  - Pair programming
  - Test-driven development

# Extreme Programming (XP): 12 practices

---

## Fine-scale feedback

- Pair programming
- Planning game
- Test-driven development
- Whole team

## Continuous process

- Continuous integration
- Refactoring or design improvement
- Small releases

## Shared understanding

- Coding standards
- Collective code ownership
- Simple design
- System metaphor

## Programmer welfare

- Sustainable pace

# XP Practice: Pair Programming

---

Pair programming – All production software is developed by two people sitting at the same machine.

Pairs and roles (driver/navigator) are frequently changed.

Provides for continuous code development, collaboration, and review.



Thoughts?

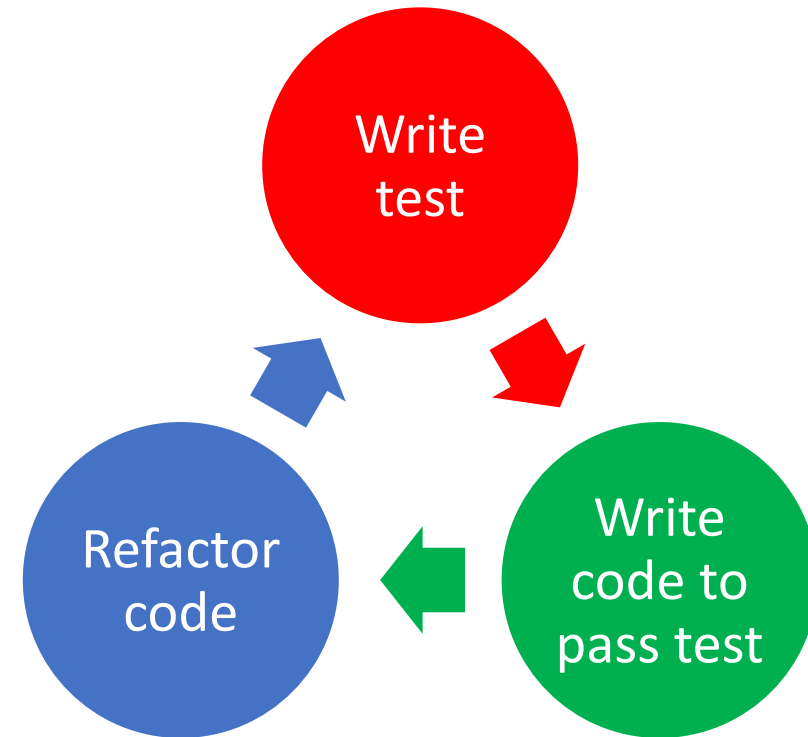
# XP Practice: Test-driven development

---

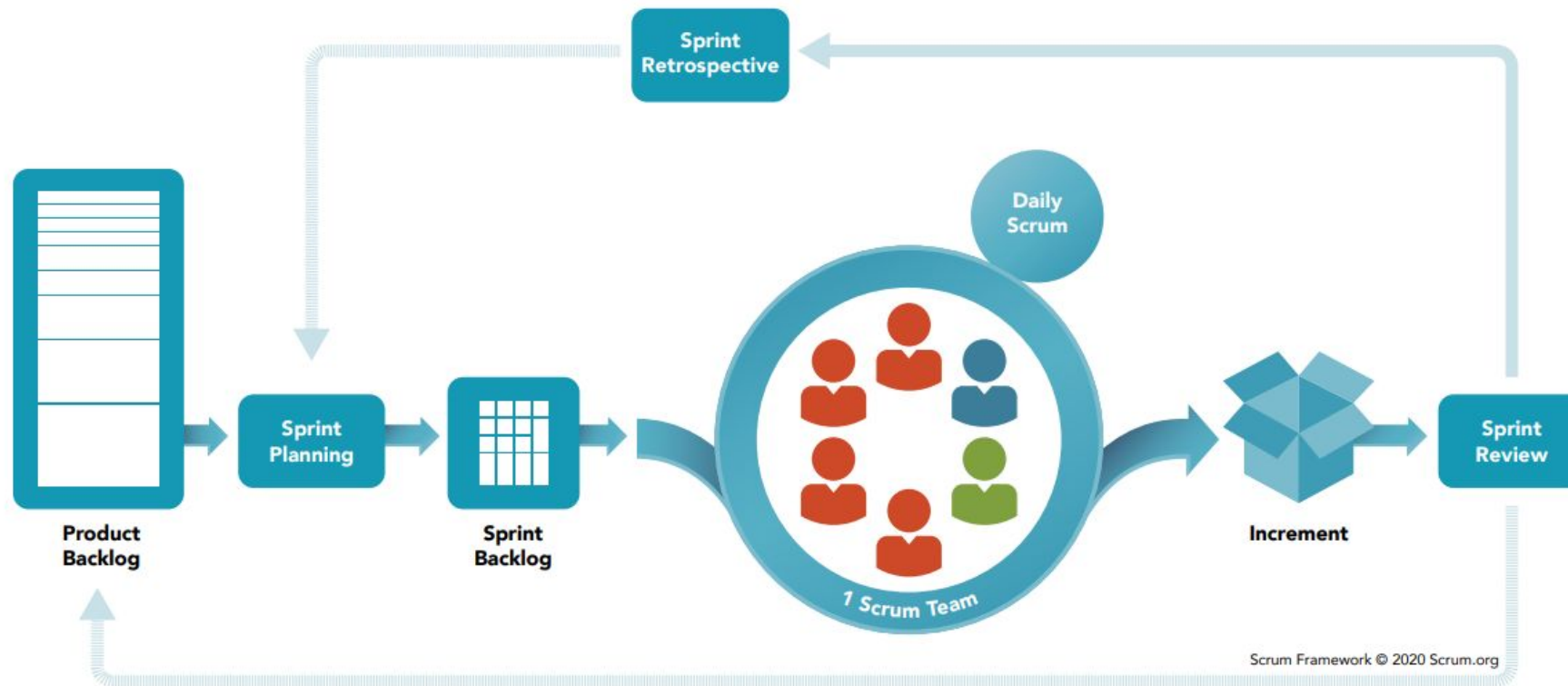
Start with requirements  
Write tests before code  
Develop code to make the tests pass

Tests run early and often

Thoughts?

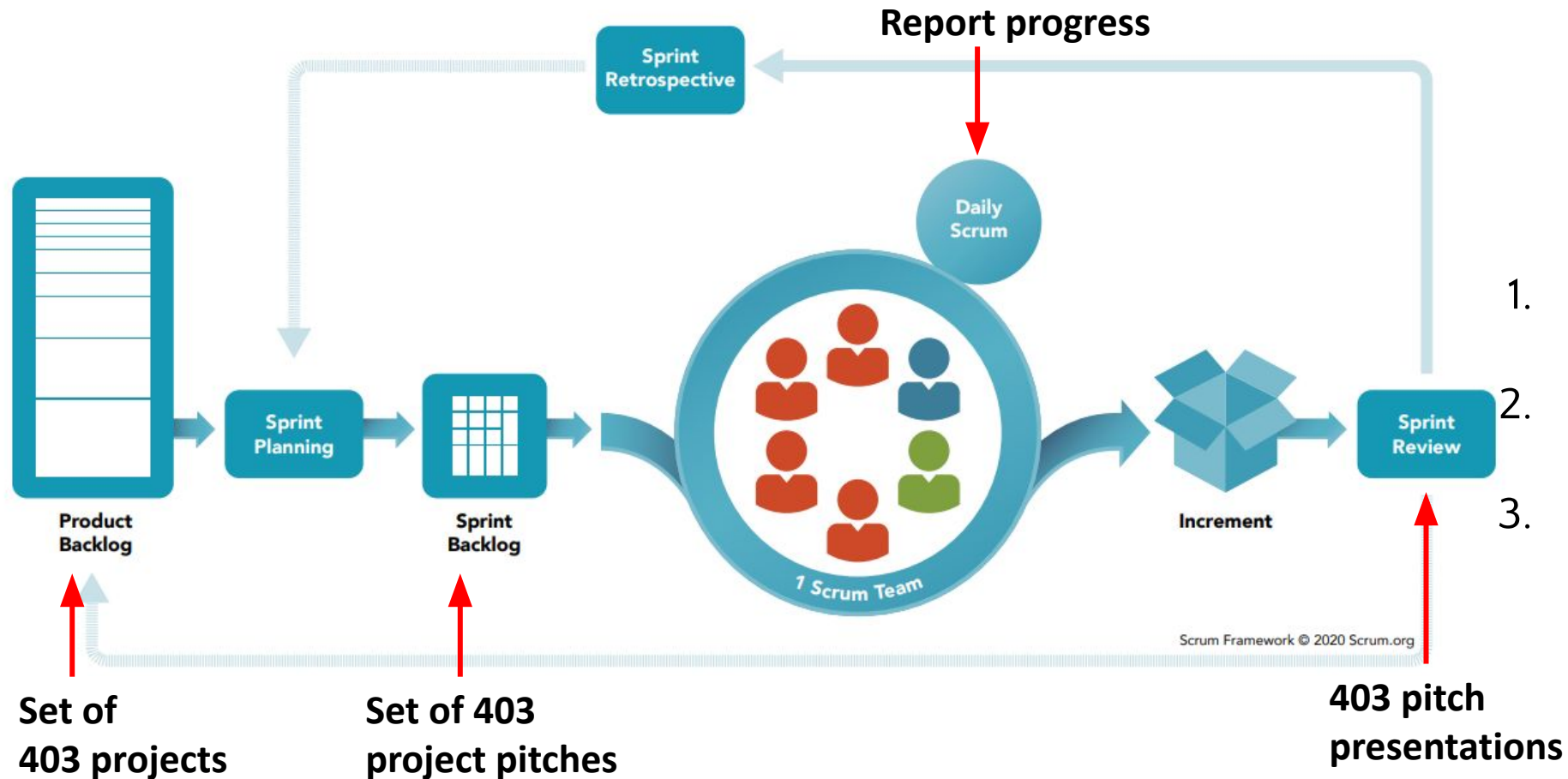


# Agile SDLC: Scrum



- Many analogies with XP
- Scrum focuses on management and productivity
- XP addresses software quality and engineering techniques

# Shall we try a daily standup?



**Daily Standup**  
Answer 3 questions

1. What did I accomplish yesterday?
2. What am I planning for today?
3. Am I blocked on anything?

# Agile Summary

---

## Pros

- Flexibility (changes are expected)
- Focus on quality (continuous testing)
- Focus on communication – with customers – with team

## Cons

- Requires experienced management and skilled developers  
(e.g., responsible, proactive, communicate well)
- Prioritizing requirements can be difficult when there are multiple stakeholders
- Needs customer to be flexible in delivery (what / when)
- Works best for small teams and small to medium-sized projects

# What SDLC would you pick and why?

---

<http://tinyurl.com/cse403-sdlc>



- A control system for anti-lock braking in a car
- A hospital accounting system that replaces an existing one
- An interactive system that allows airline passengers to quickly find replacement flights
- New innovative but tbd features for a social media app
- Your 403 class project

# What SDLC would you pick and why?

---



- A control system for anti-lock braking in a car
- A hospital accounting system that replaces an existing one
- An interactive system that allows airline passengers to quickly find replacement flights
- New innovative but tbd features for a social media app
- Your 403 class project

# Why are there so many SDLC models?!

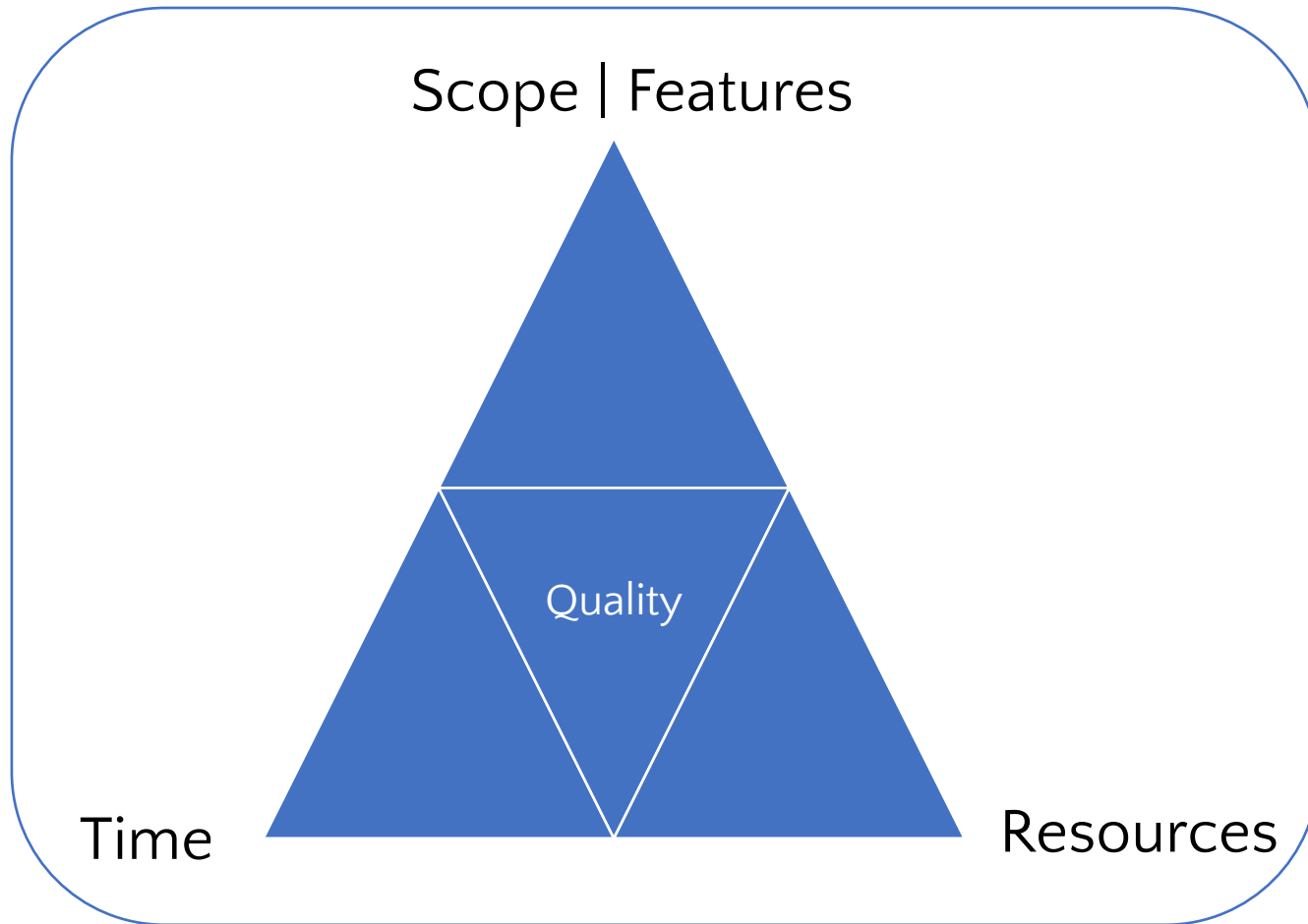
---

Choices are good 😊!

- The choice depends on the project context and requirements
- All models have the same goals: manage risks and produce high quality software
- All models involve the same general activities and stages (e.g., specification, design, implementation, and testing) and can be tailored
- Recent models involve customer feedback and the ability to adapt to changing requirements

# Triangle – project management tool

---



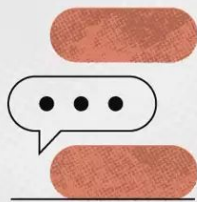
- Software projects must balance what's delivered, when, and with what resources
- When there are changes to one axis, at least one other has to adapt
- These are also good considerations when choosing a SDLC model or adapting to a changing environment

# Elevator Pitch

---

An elevator pitch is a brief, persuasive speech that you use to spark interest in a product, project or idea, or in yourself. An elevator pitch is short, about the time you spend in an elevator, hence the name.

## A foolproof elevator pitch template



01  
Introduce  
yourself



02  
Present  
the problem



03  
Present  
your solution



04  
Share your value  
proposition



05  
Add a call  
to action

You have 2  
minutes for your  
project pitch to  
the class - this is a  
good example of  
how it could flow

# Press Release

---

Write a mock product press release describing your product

Includes

Problem trying to solve

Value proposition

How differs from competitors

Release timing and teaser of future beyond release

Quotes from well known users showing their delight

Excellent way to paint the vision and get buyin