Software Engineering

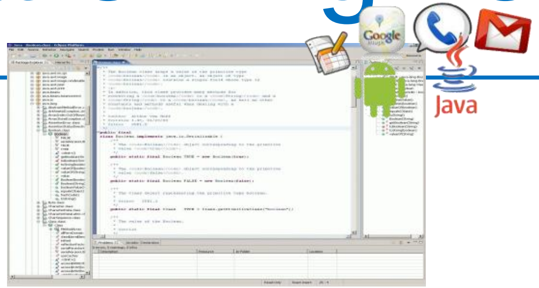
Course introduction

Today

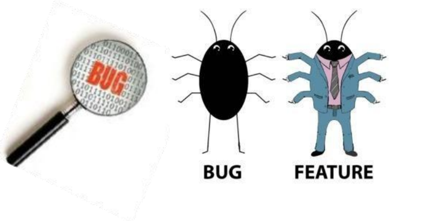
- What is Software Engineering
- Course overview and expectations

What is Software Engineerin

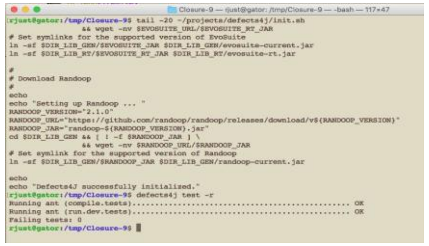
- Developing in an IDE
  and software ecosystem?



- Debugging and maintaining a software system?



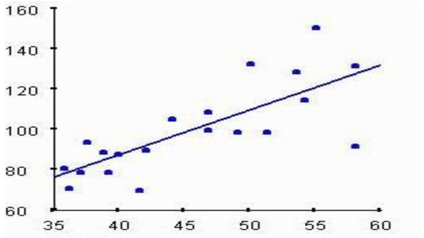
- Deploying and running
  a software system?



```
Closure-9 — rjust@gator: /tmp/Closure-9 — -bash — 117×47
rjust@gator:/tmp/Closure-9$ tail -20 ~/projects/defects4j/init.sh
                && wget -nv $EVOSUITE_URL/$EVOSUITE_RT_JAR
# Set symlinks for the supported version of EvoSuite
ln -sf $DIR_LIB_GEN/$EVOSUITE_JAR $DIR_LIB_GEN/evosuite-current.jar
ln -sf $DIR_LIB_RT/$EVOSUITE_RT_JAR $DIR_LIB_RT/evosuite-rt.jar

#
# Download Randoop
#
echo
echo "Setting up Randoop ... "
RANDOOP_VERSION="2.1.0"
RANDOOP_URL="https://github.com/randoop/randoop/releases/download/v${RANDOOP_VERSION}"
RANDOOP_JAR="randoop-${RANDOOP_VERSION}.jar"
cd $DIR_LIB_GEN && [ ! -f $RANDOOP_JAR ] \
                && wget -nv $RANDOOP_URL/$RANDOOP_JAR
# Set symlink for the supported version of Randoop
ln -sf $DIR_LIB_GEN/$RANDOOP_JAR $DIR_LIB_GEN/randoop-current.jar

echo
echo "Defects4J successfully initialized."
rjust@gator:/tmp/Closure-9$ defects4j test -r
Running ant (compile.tests)............................................ OK
Running ant (run.dev.tests)............................................ OK
Failing tests: 0
rjust@gator:/tmp/Closure-9$ 
```

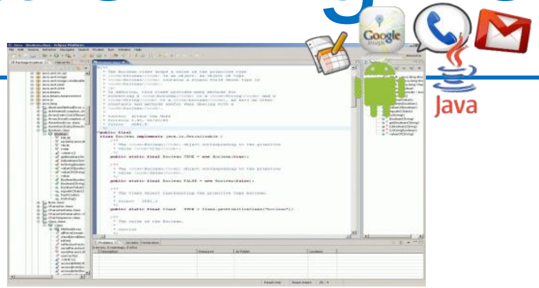


- Empirically evaluating a software system?

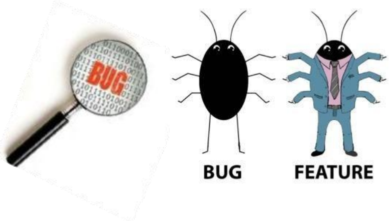
- Writing (design) docs?

What is Software Engineerin

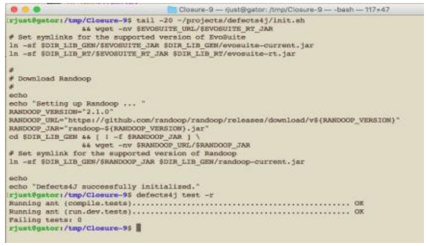
- Developing in an IDE
  and software ecosystem?



- Debugging and maintaining a software system?



- Deploying and running
  a software system?

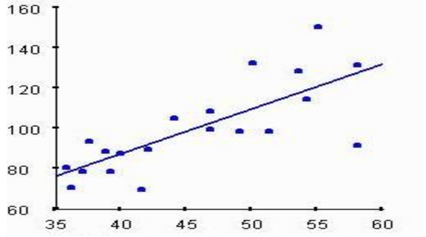


```
Closure-9 — rjust@gator: /tmp/Closure-9 — -bash — 117×47
rjust@gator:/tmp/Closure-9$ tail -20 ~/projects/defects4j/init.sh
                && wget -nv $EVOSUITE_URL/$EVOSUITE_RT_JAR
# Set symlinks for the supported version of EvoSuite
ln -sf $DIR_LIB_GEN/$EVOSUITE_JAR $DIR_LIB_GEN/evosuite-current.jar
ln -sf $DIR_LIB_RT/$EVOSUITE_RT_JAR $DIR_LIB_RT/evosuite-rt.jar

#
# Download Randoop
#
echo
echo "Setting up Randoop ... "
RANDOOP_VERSION="2.1.0"
RANDOOP_URL="https://github.com/randoop/randoop/releases/download/v${RANDOOP_VERSION}"
RANDOOP_JAR="randoop-${RANDOOP_VERSION}.jar"
cd $DIR_LIB_GEN && [ ! -f $RANDOOP_JAR ] \
                && wget -nv $RANDOOP_URL/$RANDOOP_JAR
# Set symlink for the supported version of Randoop
ln -sf $DIR_LIB_GEN/$RANDOOP_JAR $DIR_LIB_GEN/randoop-current.jar

echo
echo "Defects4J successfully initialized."
rjust@gator:/tmp/Closure-9$ defects4j test -r
Running ant (compile.tests)............................................... OK
Running ant (run.dev.tests)............................................... OK
Failing tests: 0
rjust@gator:/tmp/Closure-9$ 
```

- Empirically evaluating a software system?



- Writing (design) docs?

All of the above and much more!

Software Engineering is more than writing code

Software Engineering is the complete process of

specifying,

requirements engineering, specifications, documentation

designing, software architecture and design, UI

developing, programming (just one of many important tasks)

analyzing,

testing, debugging, linting, verification, performance engineering

deploying,

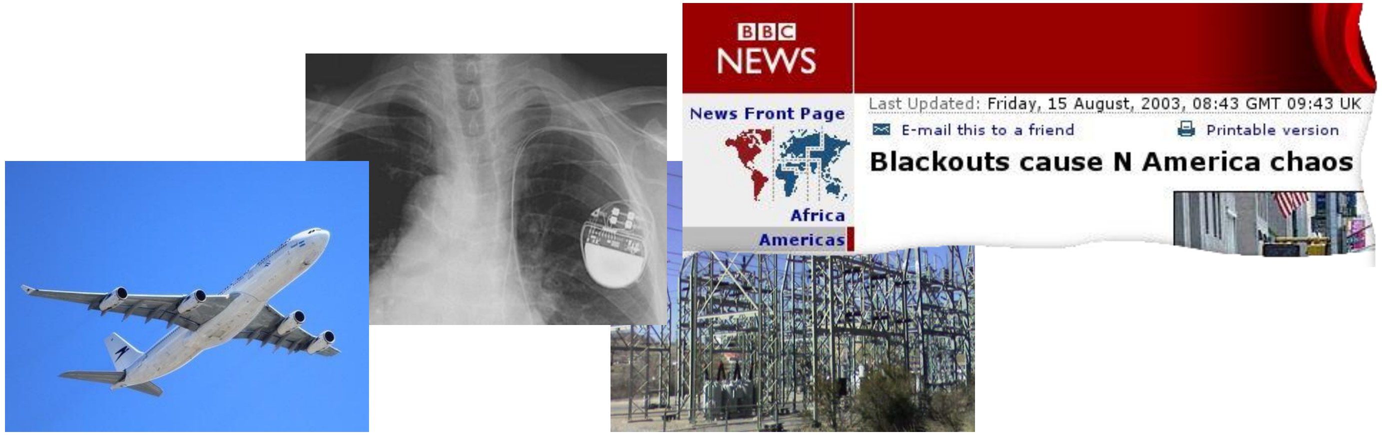
DevOps, CI, packaging, operation, remote diagnostics, documentation, websites

& maintaining refactoring, extensions, adaptation, issue tracking

a software system. nearly every system contains software

Why is Software Engineering important?

Software is everywhere!



Last Updated: Friday, 15 August, 2003, 08:43 GMT 09:43 UK

Blackouts cause N America chaos

c|net
Search CNET
Reviews
News
CNN
U.S. Edition
News Video TV Opinions More...
U.S. World Politics Tech Health Entertainment Living Travel
LaTeX
R Git
DW
BREAKING
CROWDSTRIKE
facebook

Apple finally fixes 'gotofail' OS X

released a system software
e known as the "gotofail"

The 'Heartbleed' security flaw that affects most of the Internet

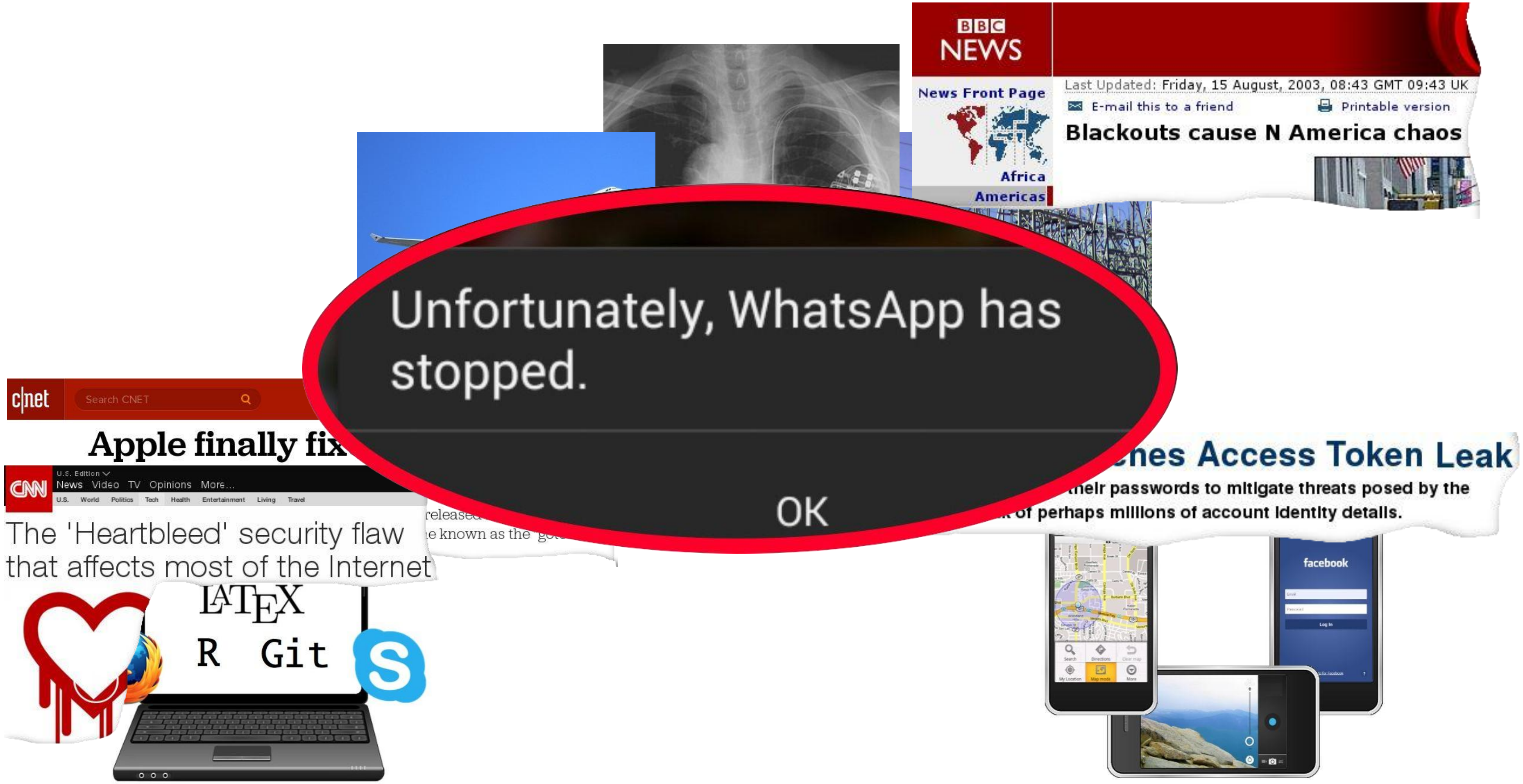
'The largest IT outage in history'

Facebook Patches Access Token Leak

Users should change their passwords to mitigate threats posed by the accidental leak of perhaps millions of account identity details.

Why is Software Engineering important?

Software is everywhere!



BBC NEWS

Last Updated: Friday, 15 August, 2003, 08:43 GMT 09:43 UK

Blackouts cause N America chaos

Unfortunately, WhatsApp has stopped.

OK

Apple finally fix

The 'Heartbleed' security flaw that affects most of the Internet

released
e known as the go

ches Access Token Leak

their passwords to mitigate threats posed by the
of perhaps millions of account identity details.

Summary: Software Engineering

What is Software Engineering?

- The complete process of specifying, designing, developing, analyzing, and maintaining a software system.

Why is it important?

It is a path to a successful product!

- Decomposes a complex engineering problem.
- Organizes processes and effort.
- Improves software reliability.
- Improves developer productivity.

Both **technical** and **management** contributions are essential.

What can you learn in this course?

- Learn best software development best practices
- Understand how software is produced – from conception to continuous development and release
- Develop skills to effectively collaborate with others towards a common delivery goal
- Experience the responsibilities, issues and tradeoffs involved in making decisions as software engineers

Grounded by working as a team to incrementally deliver a real software product/service

Software Development Lifecycles

Today’s Outline

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

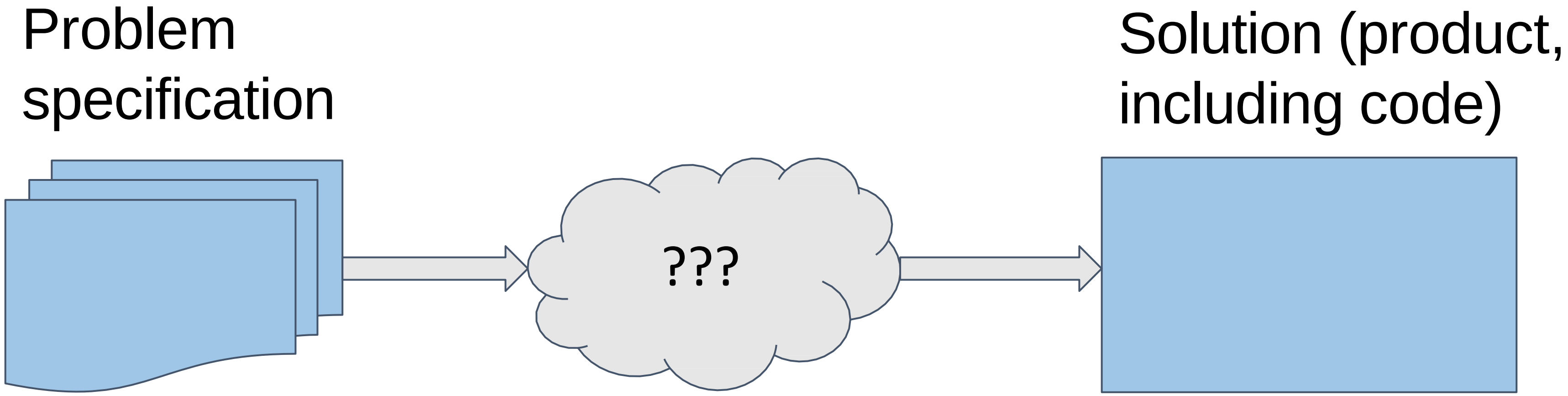
Software Engineering is …

“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

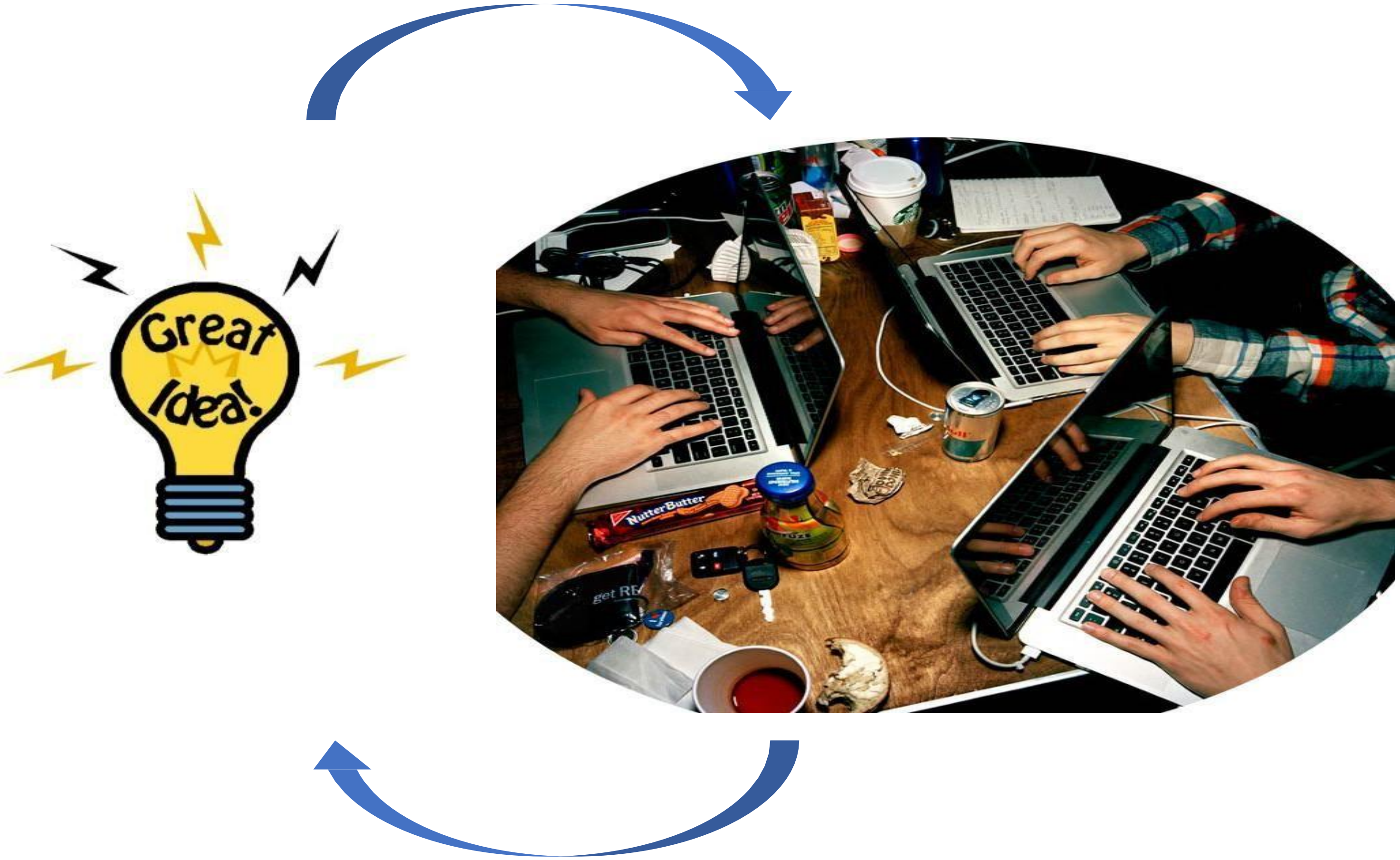


Problem specification

???

Solution (product, including code)

One solution:Code and fix



Specification
(maybe)

Deliver
(maybe)

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

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
- Design
- Implementation
- Testing
- 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

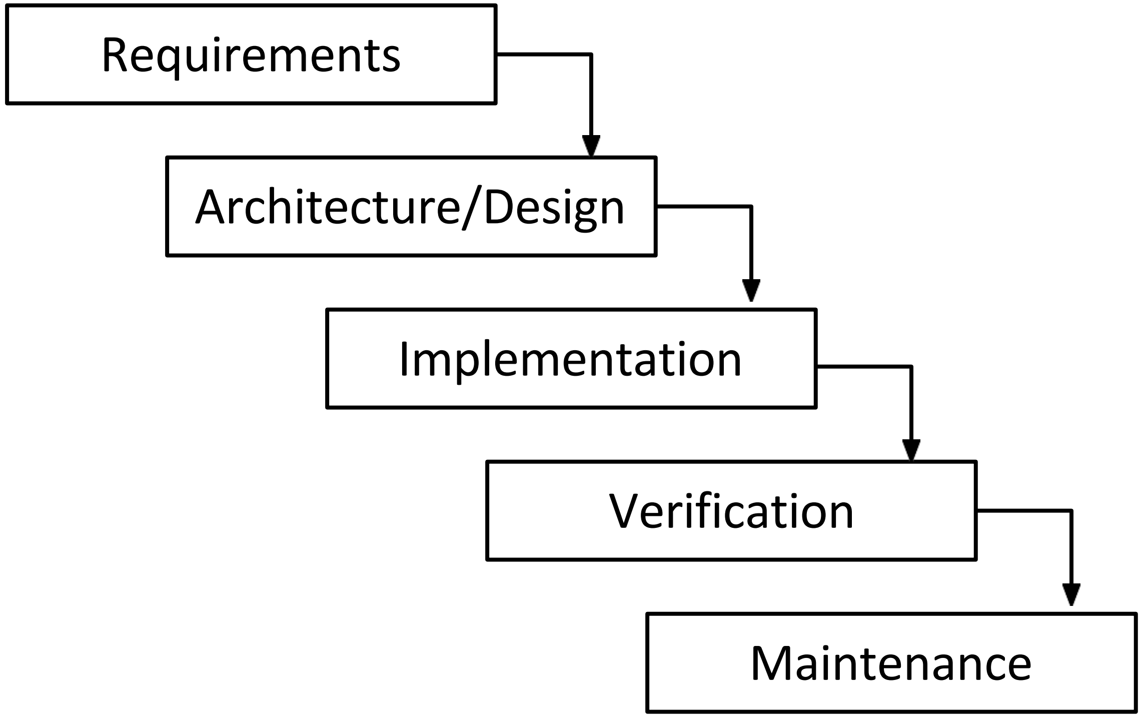
Key question: how to combine the stages and in what order

Today’s Outline

- Quick introduction
- 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)

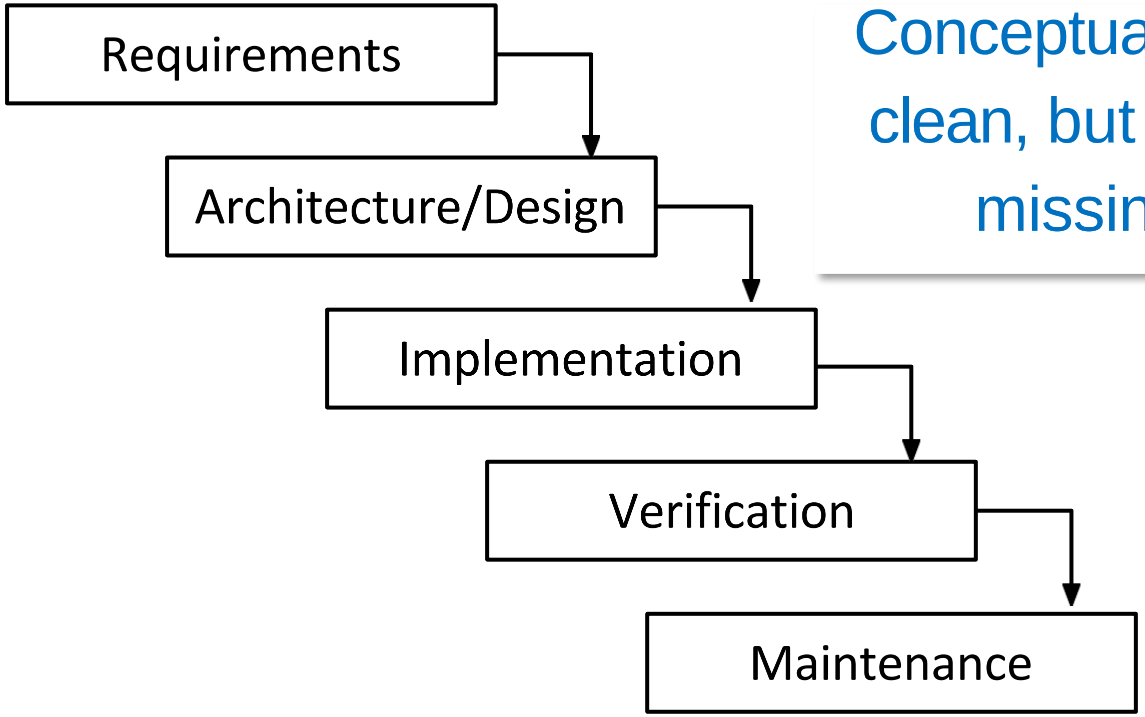
All have the same goal – deliver high quality software, on time, meeting the customer’s needs

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

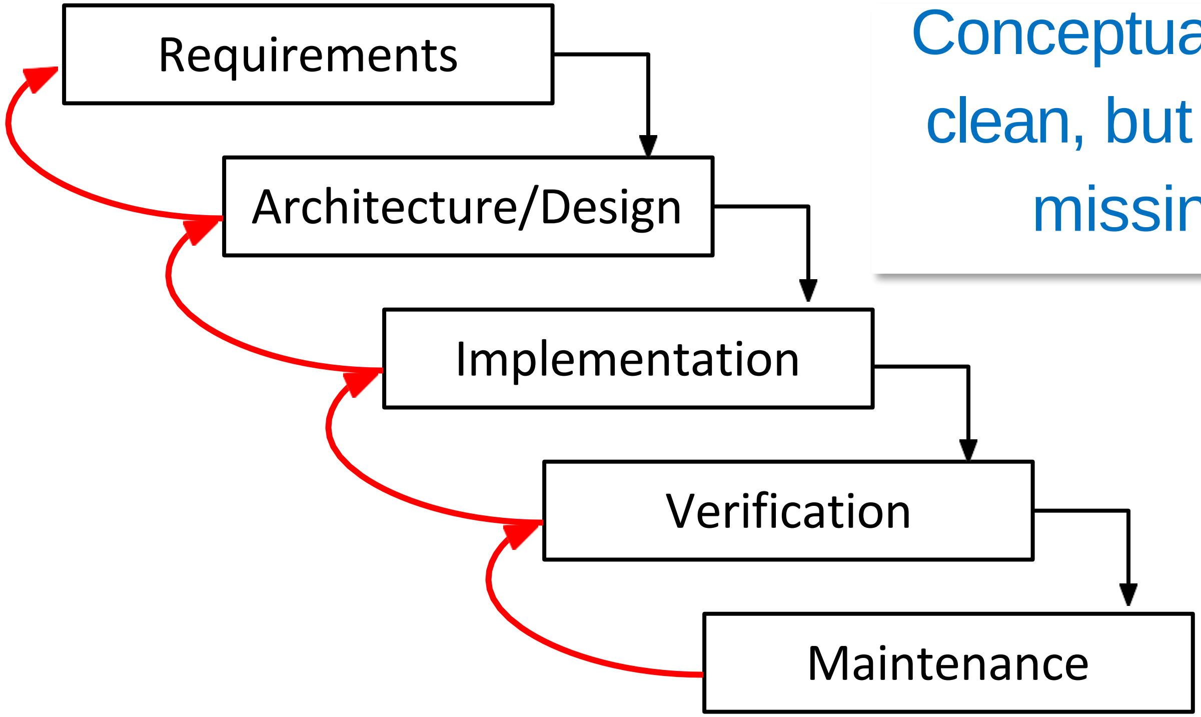
SDLC: Waterfall model



Conceptually very clean, but what's missing?

- 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

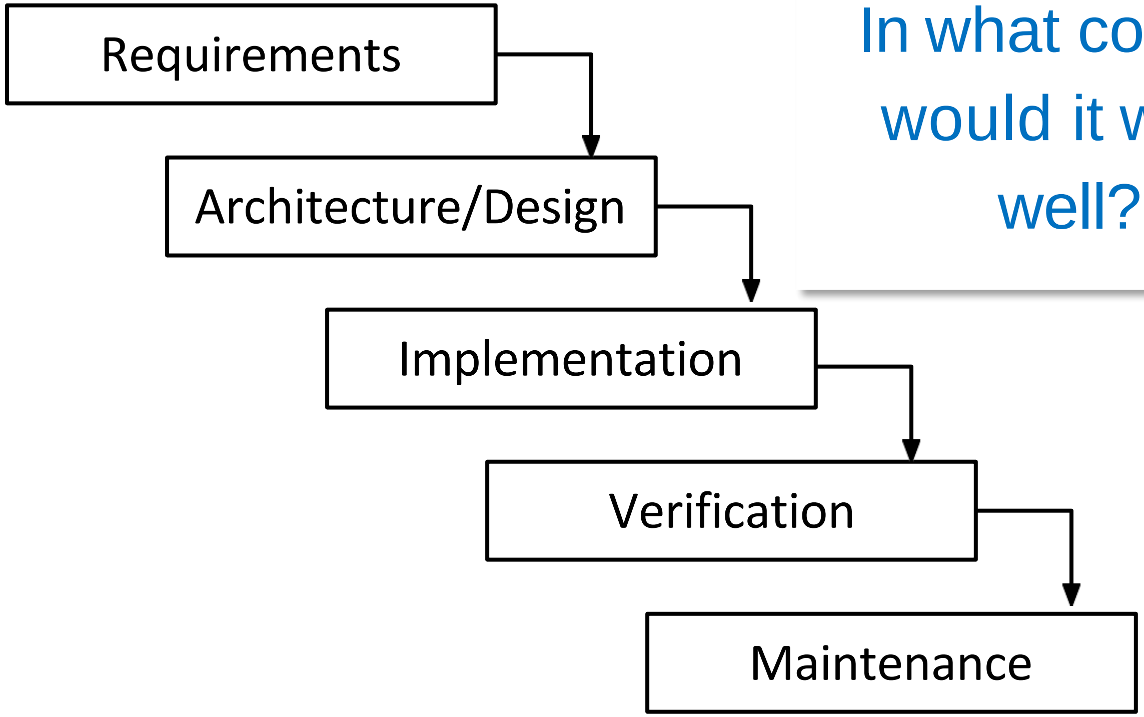
SDLC: Waterfall model



Conceptually very clean, but what's missing?

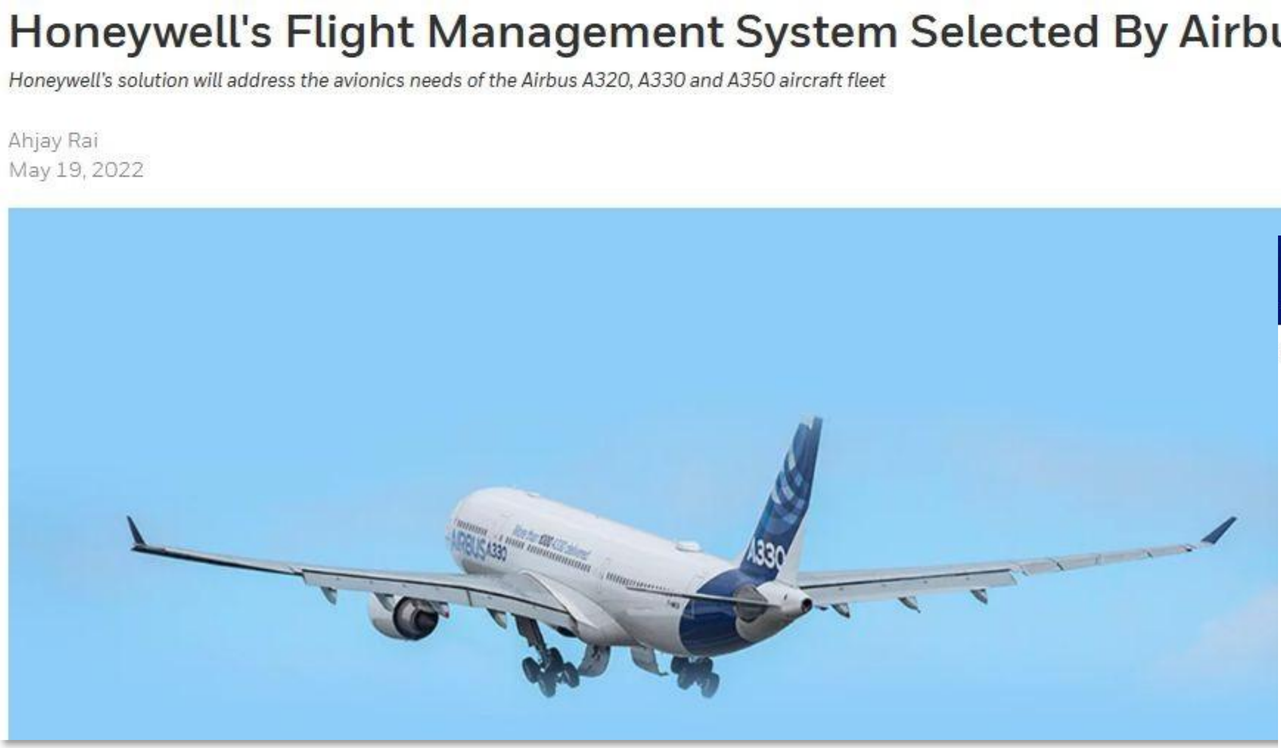
- Top-down approach
- Sequential, non-overlapping activities and steps
- Each step is signed off on and then frozen
- Most steps result in a final document
- Backsteps to correct mistakes

SDLC: Waterfall model



In what context would it work well?

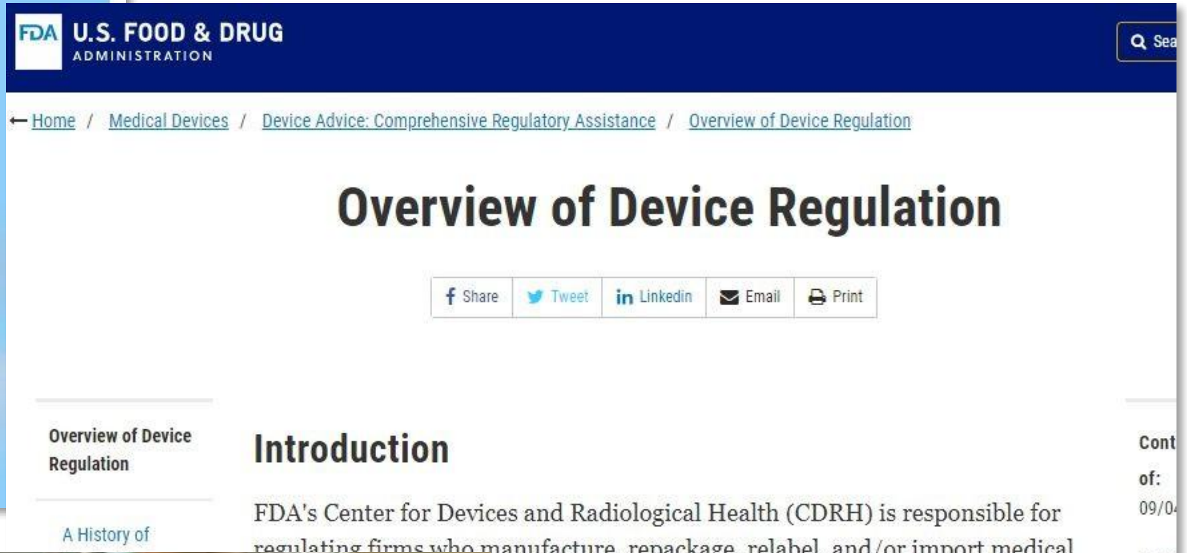
- Top-down approach
- Sequential, non-overlapping activities and steps
- Each step is signed off on and then frozen
- Most steps result in a final document



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



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

Overview of Device Regulation

Overview of Device Regulation

A History of

Introduction

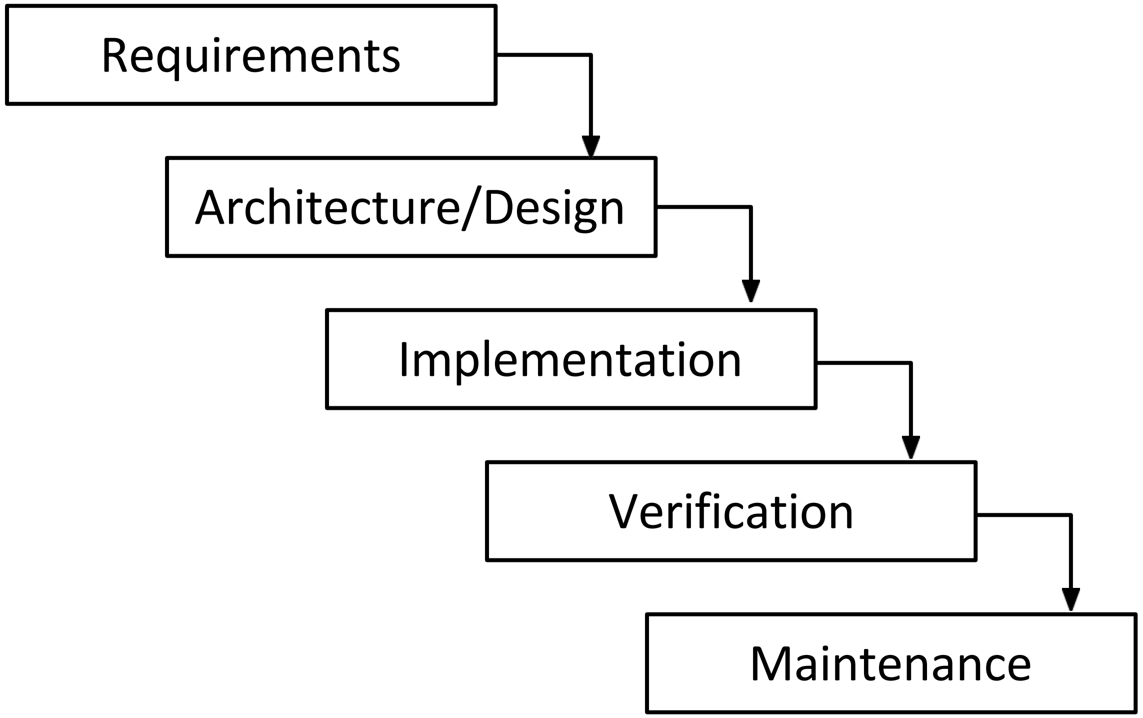
FDA's Center for Devices and Radiological Health (CDRH) is responsible for regulating firms who manufacture, repackage, relabel, and/or import medical
ted States. In addition, CDRH regulates radiation-
oducts (medical and non-medical) such as lasers, x-ray
uipment, microwave ovens and color televisions.

Cont
of:
09/0

Regu
Prod
Medi
Radia
Prod

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

Waterfall: pros and cons



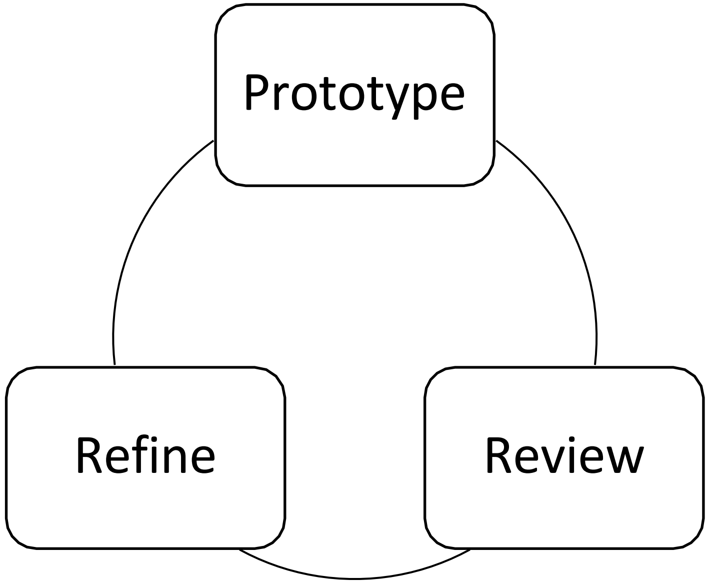
Pros:

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

Cons:

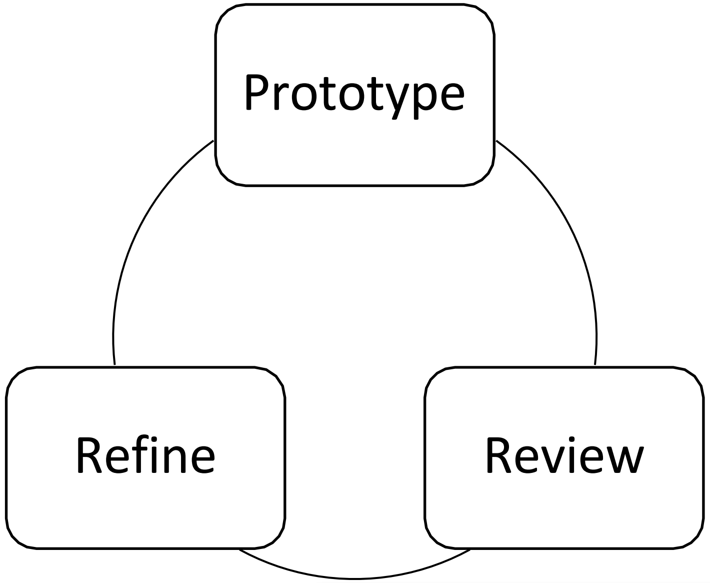
- Hard to do all the planning upfront
- Inflexible – changes are expensive
- Test and integration come late – fixes are expensive
- Final product may not match the customer’s needs

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

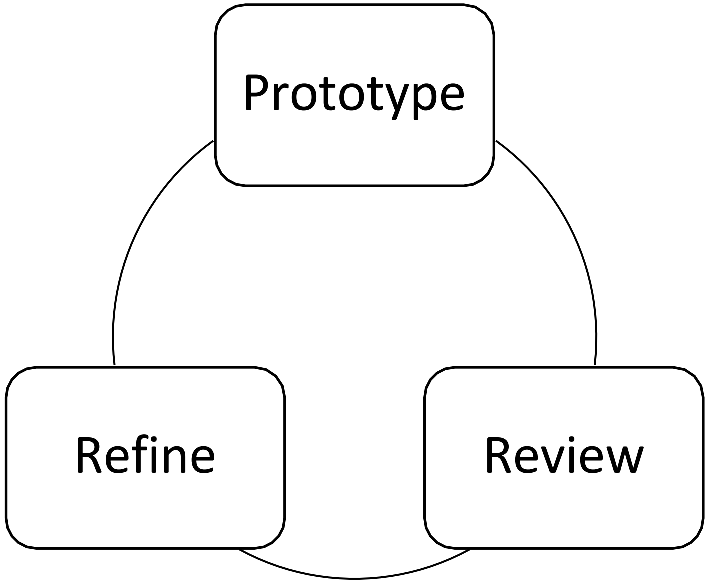
SDLC: Prototyping



In what context would it work well?

- 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

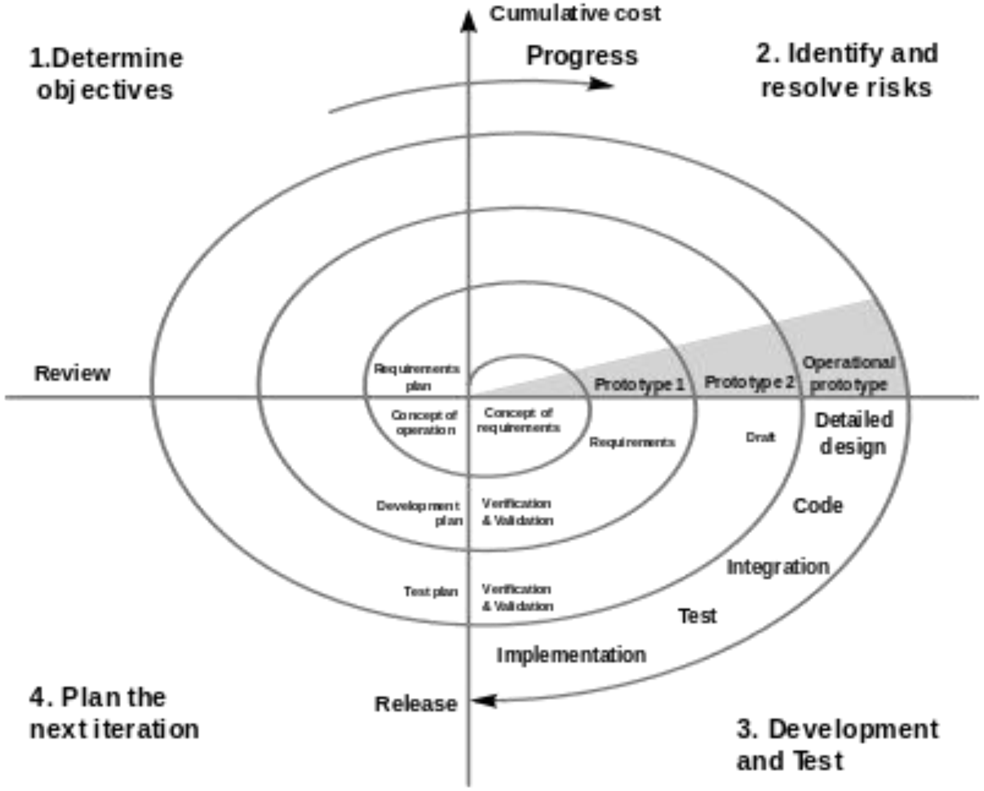
Prototyping: pros and cons



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)

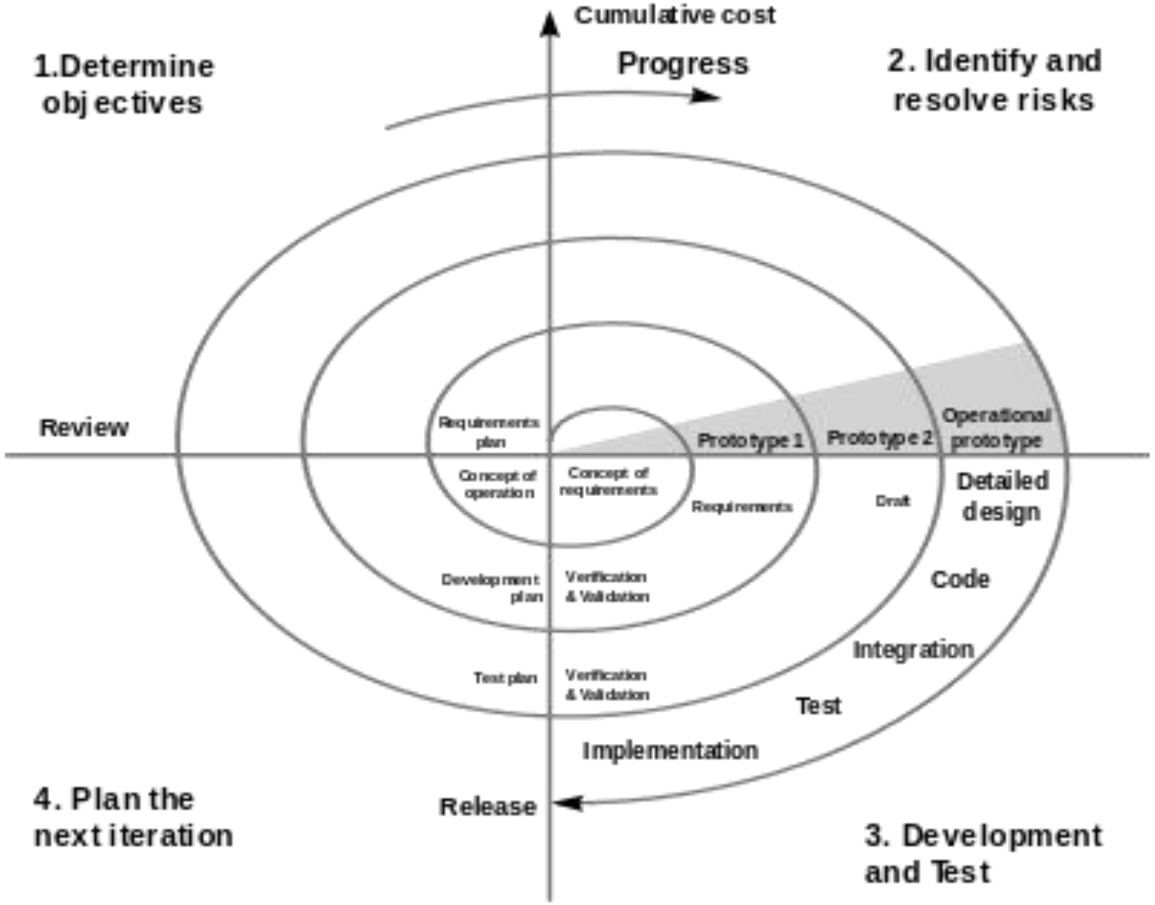
SDLC: Spiral Model



- 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)

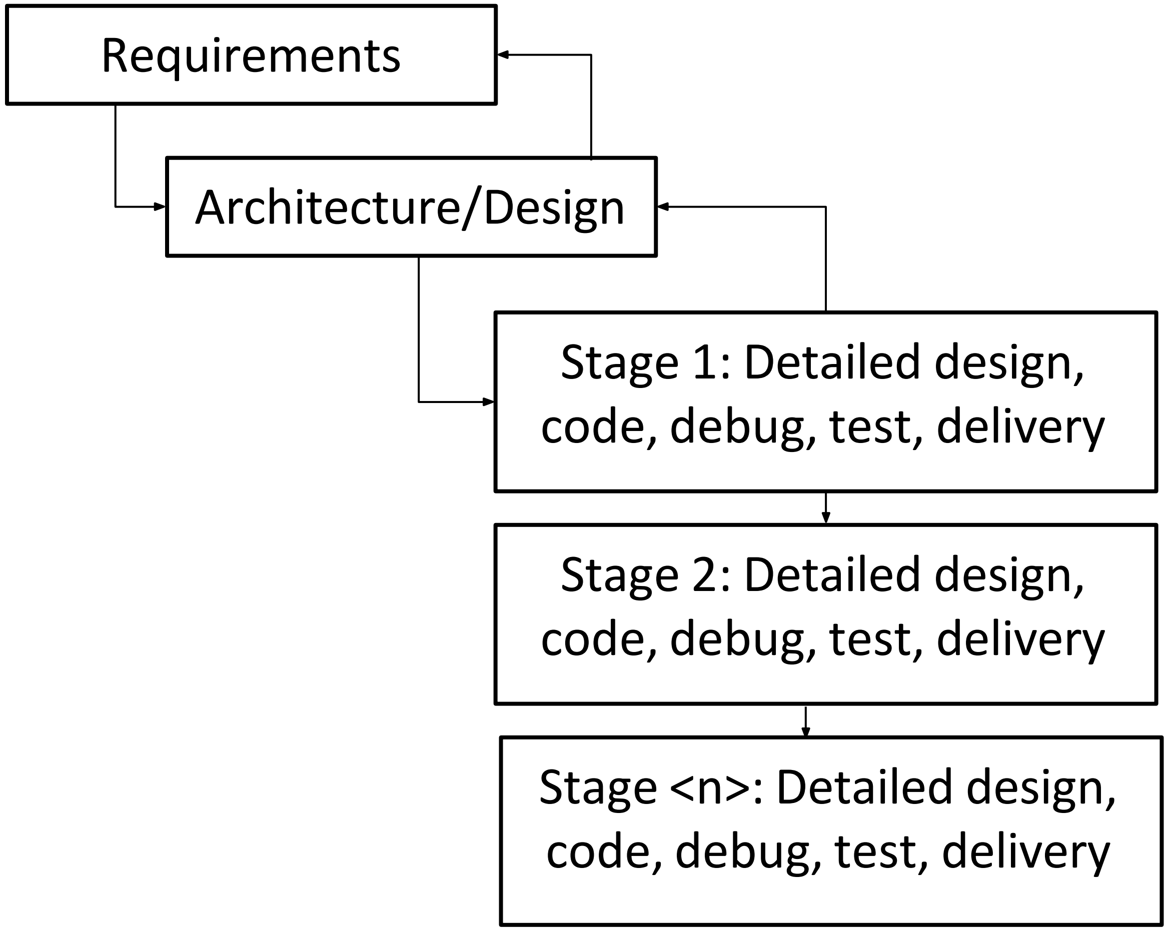
Boehm, *Spiral Development: Experience, Principles,and Refinements*

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 🤯



Requirements

Architecture/Design

Stage 1: Detailed design, code, debug, test, delivery

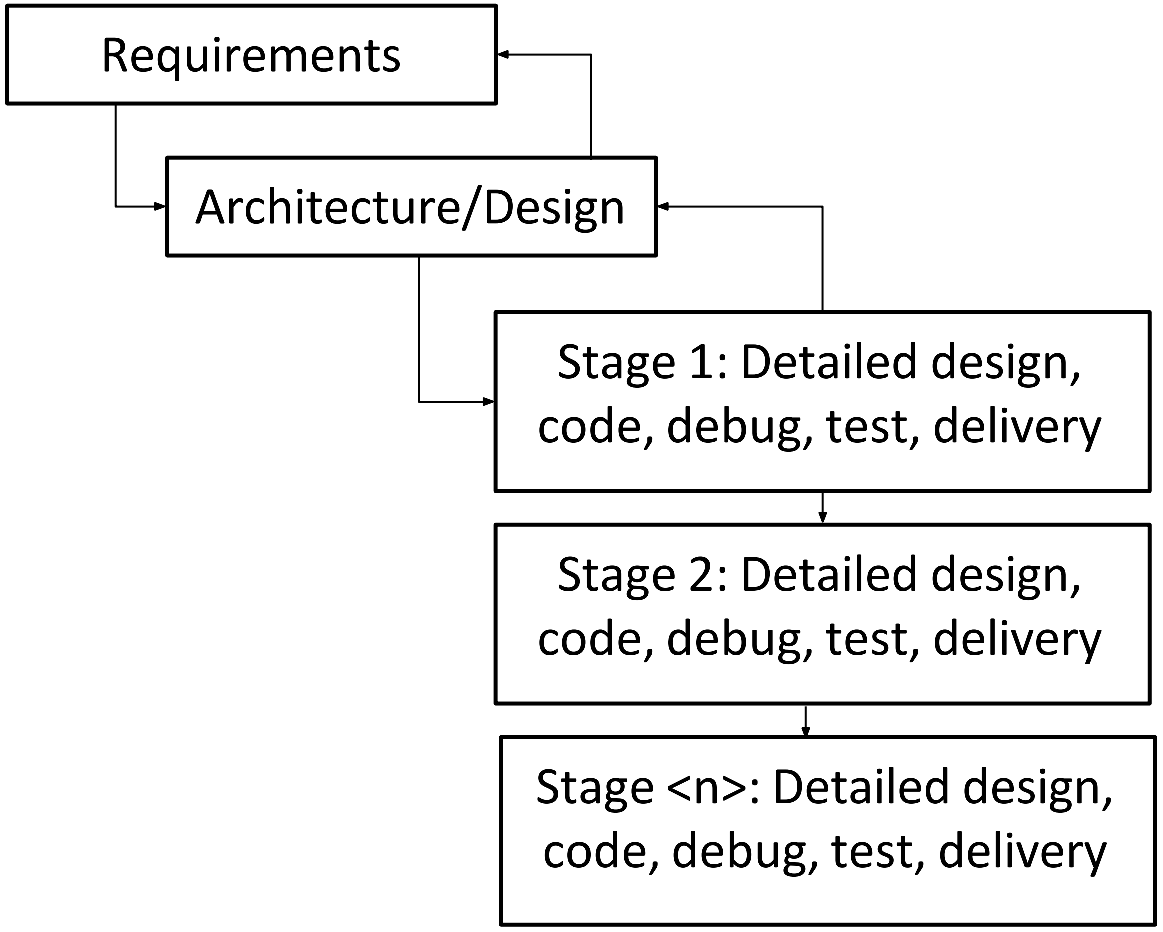
Stage 2: Detailed design, code, debug, test, delivery

Stage <n>: Detailed design, code, debug, test, delivery

- Waterfall-like planning upfront then spiral/agile-like short release cycles
- Pros: ?
- Cons: ?

McConnell: https://stevemcconnell.com/

Staged Delivery: pros and cons


Requirements
Architecture/Design
Stage 1: Detailed design, code, debug, test, delivery
Stage 2: Detailed design, code, debug, test, delivery
Stage <n>: Detailed design, code, debug, test, delivery

- 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

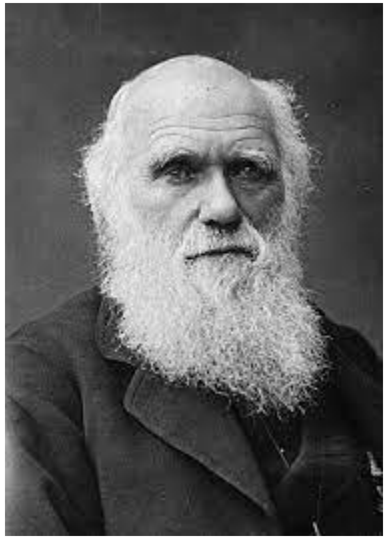
- Quick introduction
- Software development lifecycles (SDLC)
  - What and why are they needed
  - Recurring themes
  - **Popular models and their tradeoffs**
    - Waterfall model
    - Prototyping
    - Spiral model
    - Staged delivery
    - Agile (XP, Scrum)

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

WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
scottadams@aol.com
www.dilbert.com
11-26-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.

© Scott Adams, Inc./Dist. by UFS, Inc.

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.

Agile models

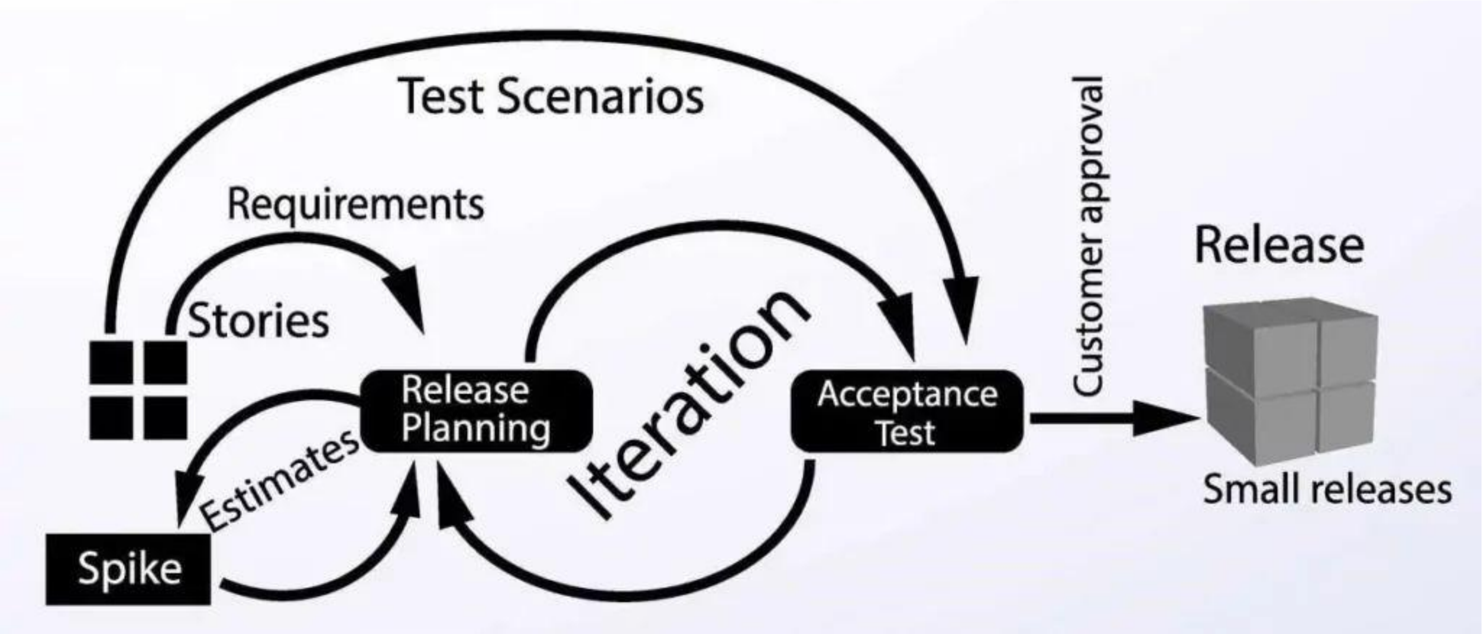
“Agile software development” is a general term for frameworks and practices outlined in the Agile Manifesto

Agile models

- Aim to deliver a high-quality product to the customer as fast as possible
- Focus on simplicity, excellence, 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

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.

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

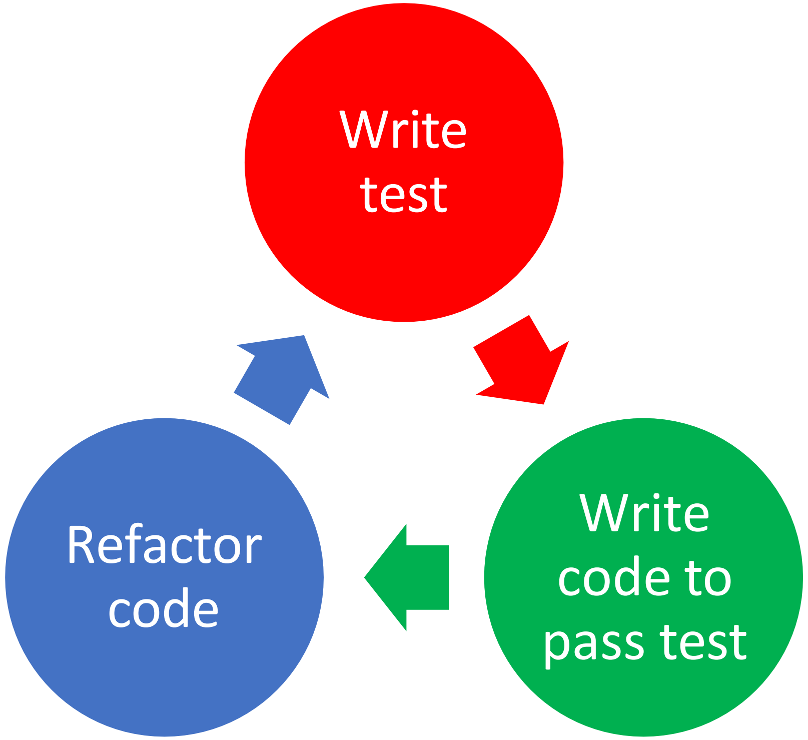
Start with requirements

Write tests before code

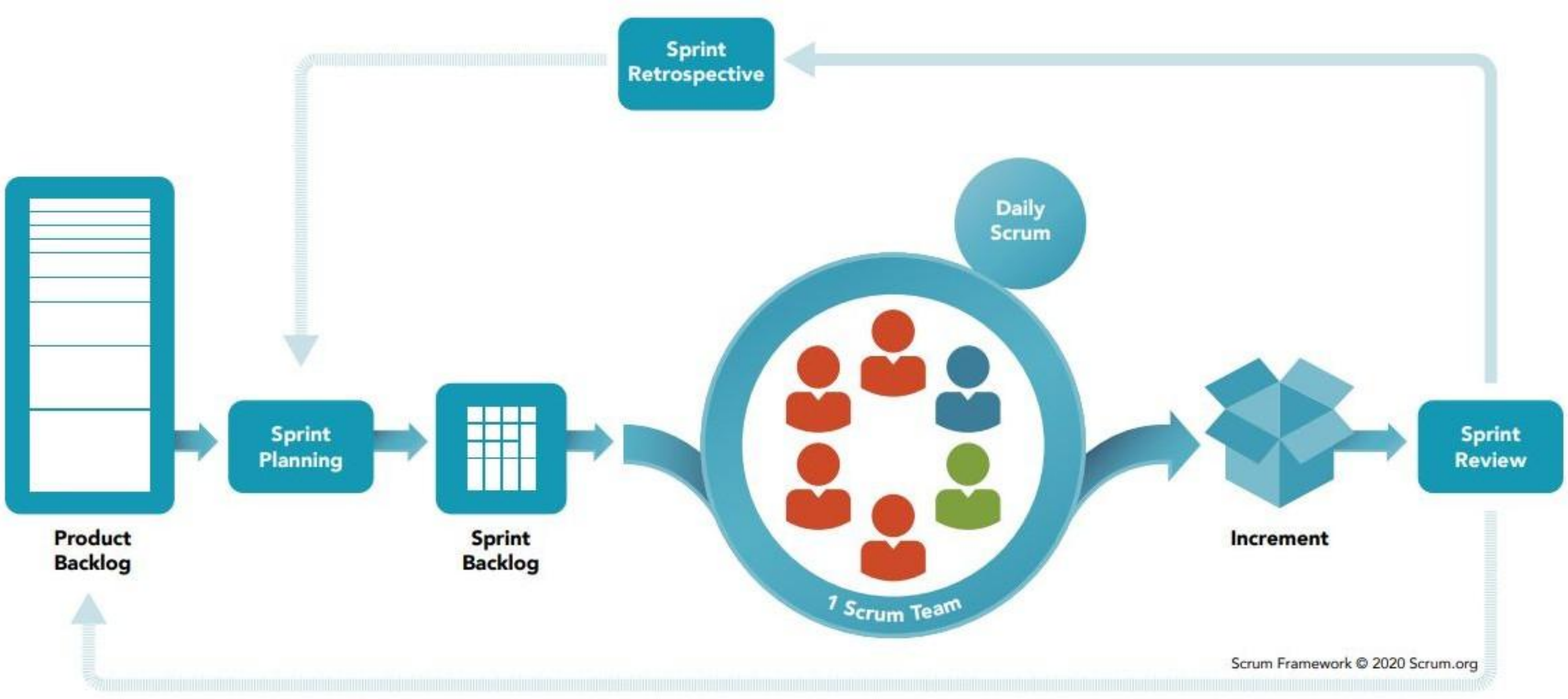
Develop code to make the tests pass

Tests run early and often

Thoughts?

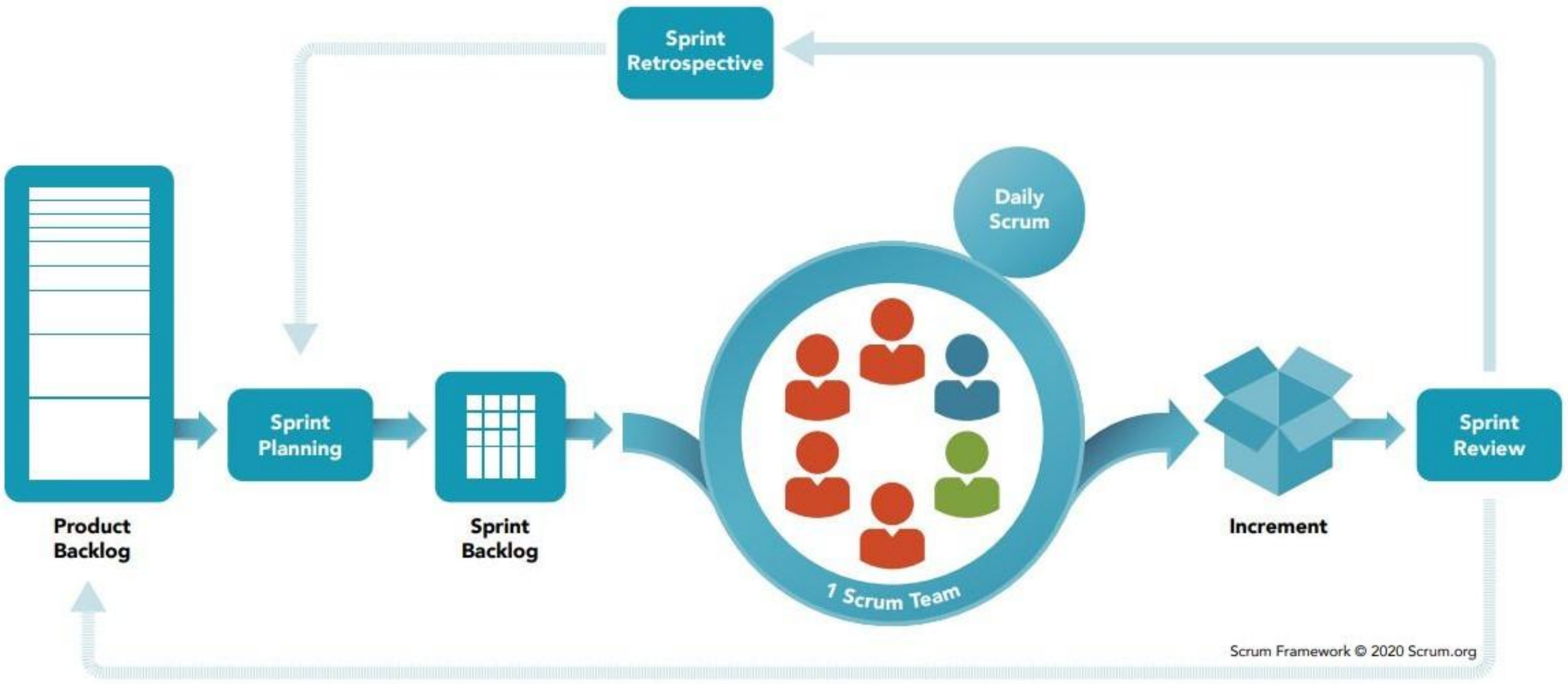


Agile SDLC: Scrum



- 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?

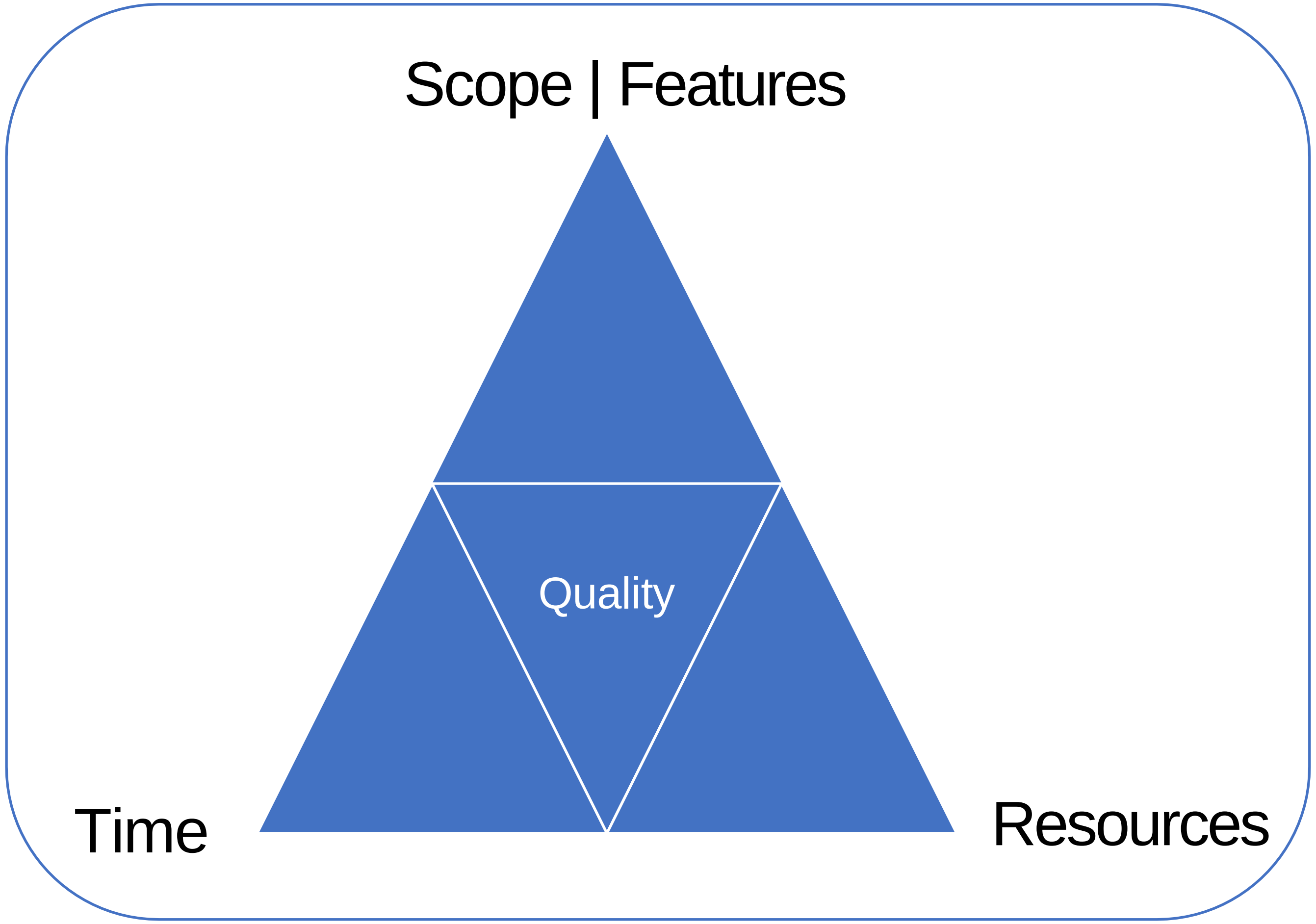
- A control system for anti-lock braking in a car
- A hospital accounting system that replaces an existing one
- An interactive system that allows airline passengers to quickly find replacement flights
- New innovative but tbd features for a social media app

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