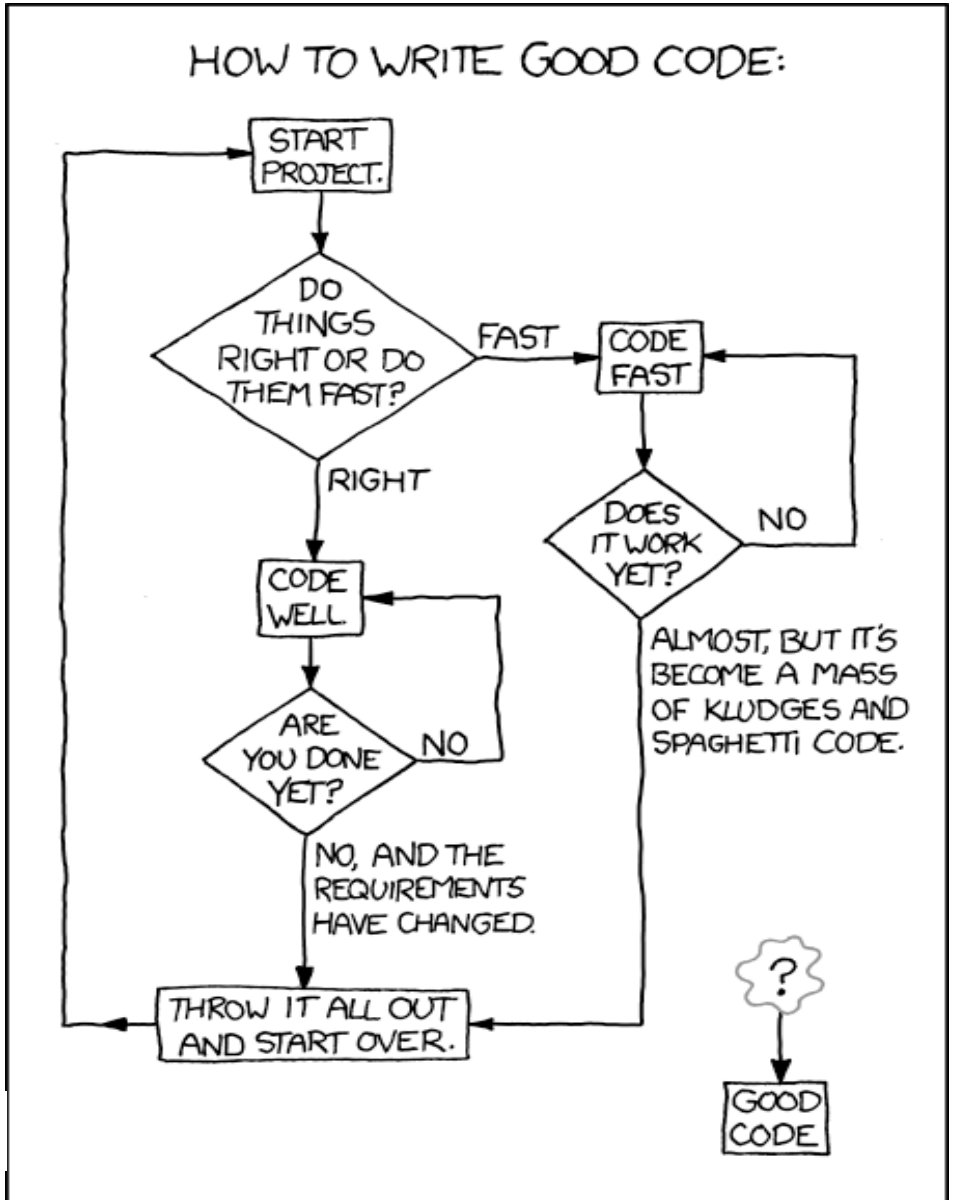


Source: xkcd 844

CPEN 321

W1 L2:
Software Lifecycle and Processes

Software Engineering

Margaret Hamilton – an American computer scientist and systems engineer. She was the Director of the Software Engineering Division of the MIT Instrumentation Lab., which developed on-board flight software for NASA's Apollo space program.

Hamilton in 1969, standing next to the navigation software that she and her MIT team produced for the Apollo project.

Her team worked on priority-based async. scheduling, prevented last-minute abort of the first moon landing

Margaret Hamilton

Hamilton in 1995

Born	Margaret Elaine Heafield August 17, 1936 (age 87) Paoli, Indiana, U.S.
Education	University of Michigan Earlham College (BA)
Occupation	Software engineer

Software Engineering

Coined the term “software engineering” during the Apollo space mission days :

- wanted to give software “legitimacy”, just like with other engineering disciplines
- “When I first came up with the term, no one had heard of it before […].
  It was a memorable day when one of the most respected hardware gurus explained to everyone in a meeting that he agreed with me that the process of building software should also be considered an engineering discipline, just like with hardware.”

Margaret Hamilton

Hamilton in 1995

Born	Margaret Elaine Heafield August 17, 1936 (age 87) Paoli, Indiana, U.S.
Education	University of Michigan Earlham College (BA)
Occupation	Software engineer

Margaret Hamilton @ ICSE 2018

Now: CEO of Hamilton Technologies.
Received the NASA Exceptional Space Act Award (2003), and the Presidential Medal of Freedom awarded by Barack Obama (2016).

https://www.youtube.com/watch?v=kTn56jJW4zY
https://www.youtube.com/watch?v=ZbVOF0Uk5lU

A Bit More History (aka girls can code)

An outline (algorithm) for what would have been the first piece of software was written by Ada Lovelace – English mathematician and writer in the 19th century.

Trial model of a part of the Analytical Engine, built by Babbage, as displayed at the Science Museum (London)[1]

It was written for the planned Analytical Engine.

Ada, Countess of Lovelace

Ada, Countess of Lovelace, 1840

Born	The Hon. Augusta Ada Byron 10 December 1815 London, England
Died	27 November 1852 (aged 36) Marylebone, London, England
Resting place	Church of St. Mary Magdalene, Hucknall, Nottingham, England
Known for	Mathematics Computing

Question

Software Engineering is about:

A – software development processes

B – development of large software systems

C – time, resources, and risk management

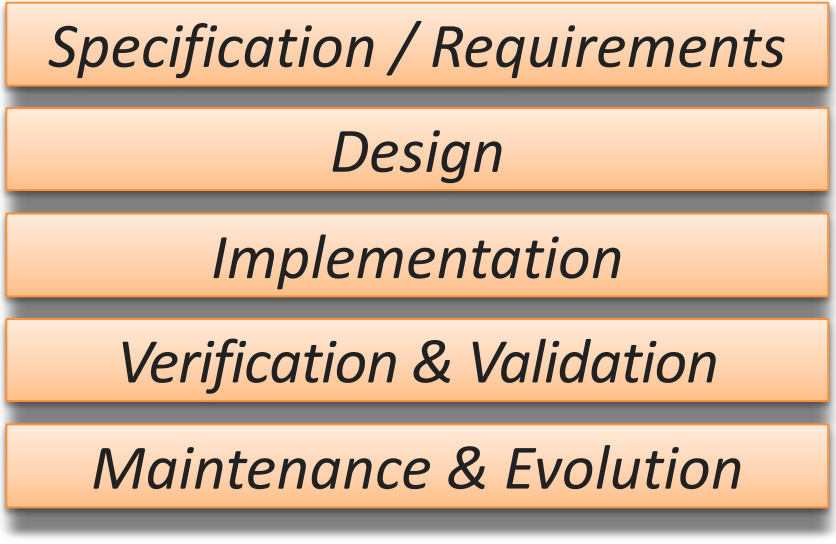
D – working with people: teammates, customers, managers, etc.

E – all of the above

Agenda

- Software Lifecycle
- Software Development Processes

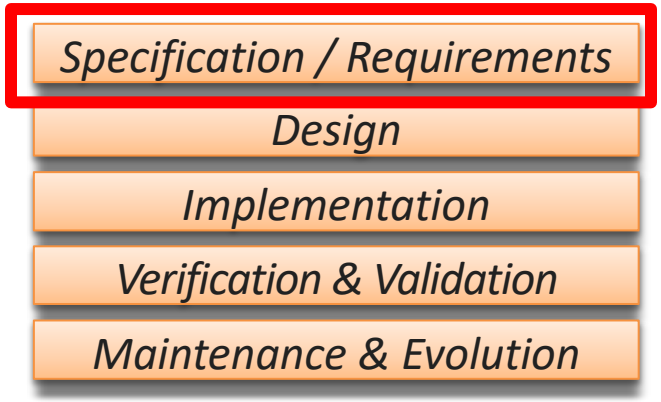
Five Essential Tasks in Software Engineering



Specification / Requirements

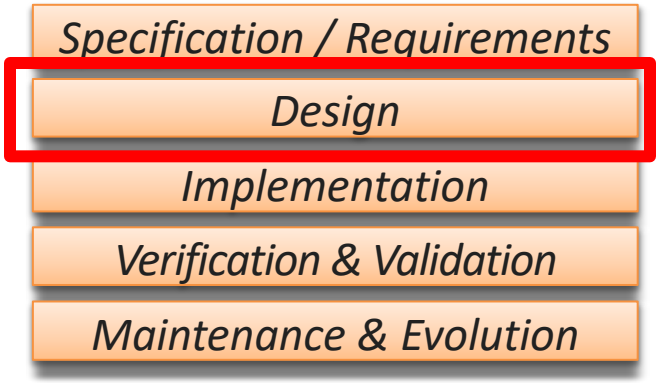
- A **software specification** is a description of a software system to be developed
  - Lays out *functional* and *non-functional* requirements

- **Requirements engineering** is the process of defining, documenting and maintaining requirements
  - Requirements elicitation
  - Requirements analysis and negotiation
  - Requirements validation
  - Requirements management



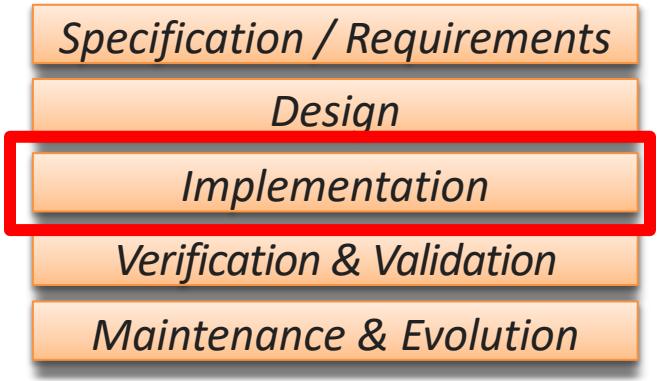
Design

- Guides the development team in building a software product
- Different aspects of design:
  - Architectural design
  - Interface design
  - Data structure design
  - Algorithm design
  - etc.
- A competent design relies on:
  - known best practices and law
  - past experience,
  - creative skill,
  - etc.



Implementation

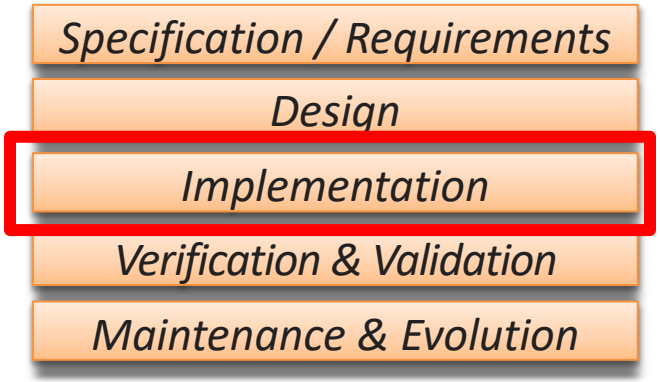
- The process of converting the software design into an executable system
- An art, a craft, and an engineering discipline…



Implementation

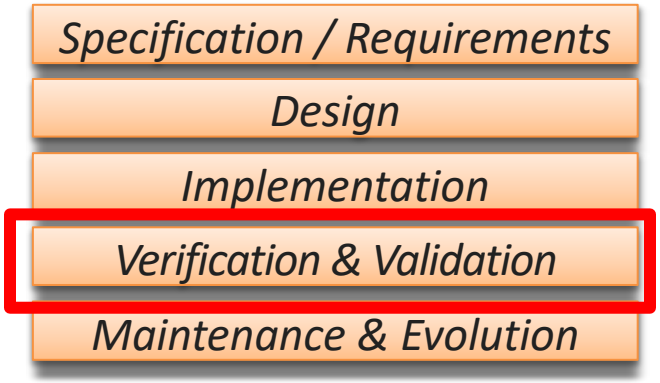
- The process of converting the software design into an executable system
- An art, a craft, and an engineering discipline…
  - Good design helps
  - Patterns and anti-patterns

- End-product must address some fundamental principles (requirements):
  - Efficiency/performance
  - Robustness
  - Usability
  - Maintainability
  - Etc.

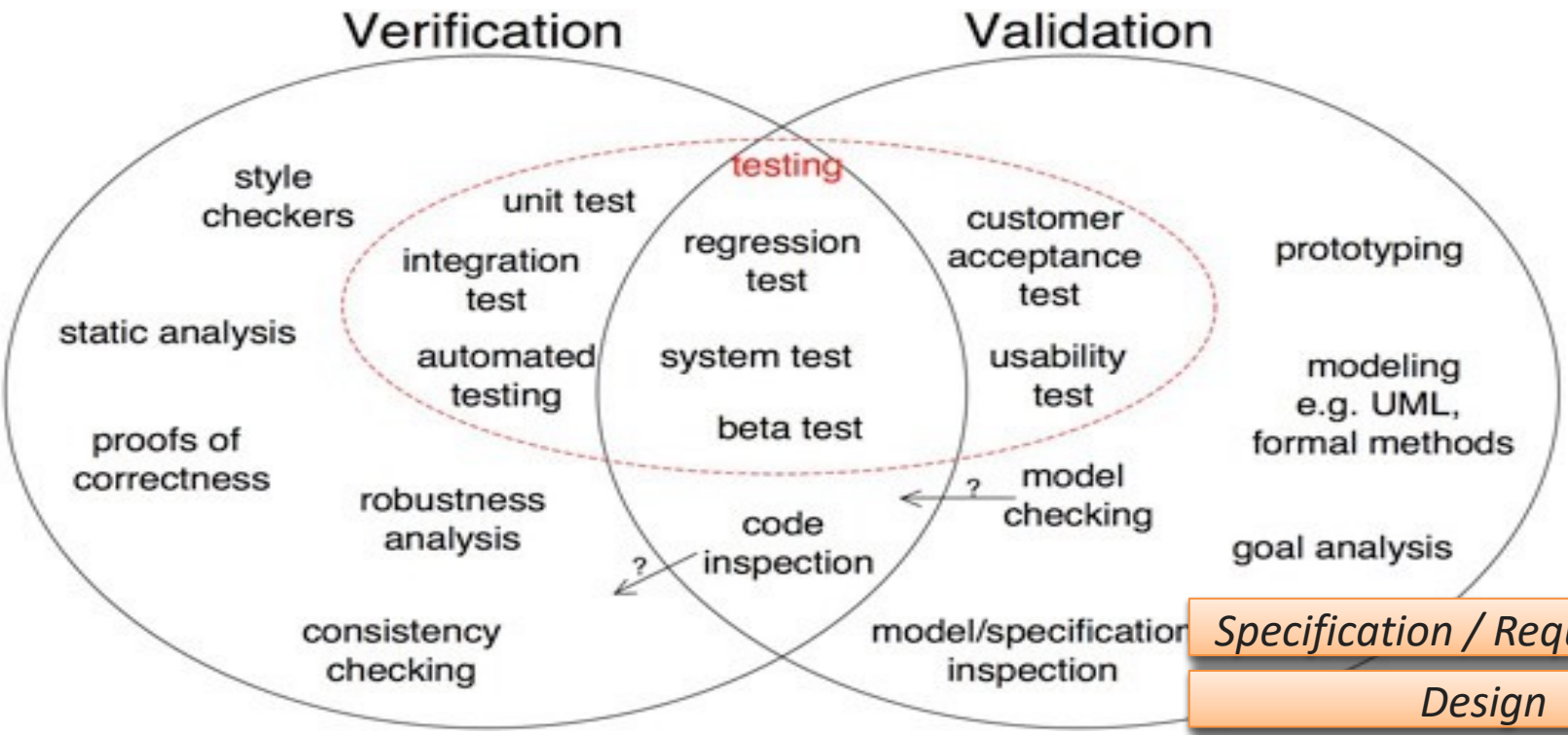


Verification & Validation

- Validation and Verification (V & V) shows that a system conforms to its specification and meets the requirements of the customer
- **Verification**: Are we building the product right?
  - Does the software meet the specification?
- **Validation**: Are we building the right product?
  - Does the specification capture the customer's needs?



Verification & Validation



- *Specification / Requirements*
- *Design*
- *Implementation*
- *Verification & Validation*
- *Maintenance & Evolution*

Maintenance & Evolution

- The modification of a software product after delivery

- Four main types of tasks:
  - Corrective – fixing errors
  - Perfective – implementing new or changed user requirements
  - Adaptive – modifying the system to cope with changes in the environment (e.g., a new OS version)
  - Preventive – increasing software maintainability or reliability (deletion of obsolete capabilities, optimization, etc.)

Maintenance & Evolution

- The modification of a software product after delivery

- Four main types of tasks:
  - Corrective – fixing errors
  - Perfective – implementing new or changed user requirements
  - Adaptive – modifying the system to cope with changes in the environment (e.g., a new OS version)
  - Preventive – increasing software maintainability or reliability (deletion of obsolete capabilities, optimization, etc.)

Most of the time →

Maintenance & Evolution

- The modification of a software product after delivery

- Four main types of tasks:

21% – Corrective – fixing errors

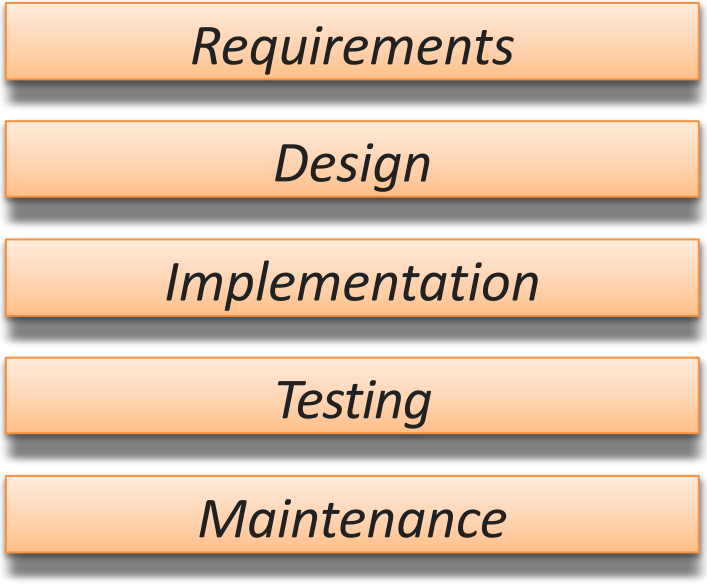
75% – Perfective – implementing new or changed user requirements

75% – Adaptive – modifying the system to cope with changes in the environment (e.g., a new OS version)

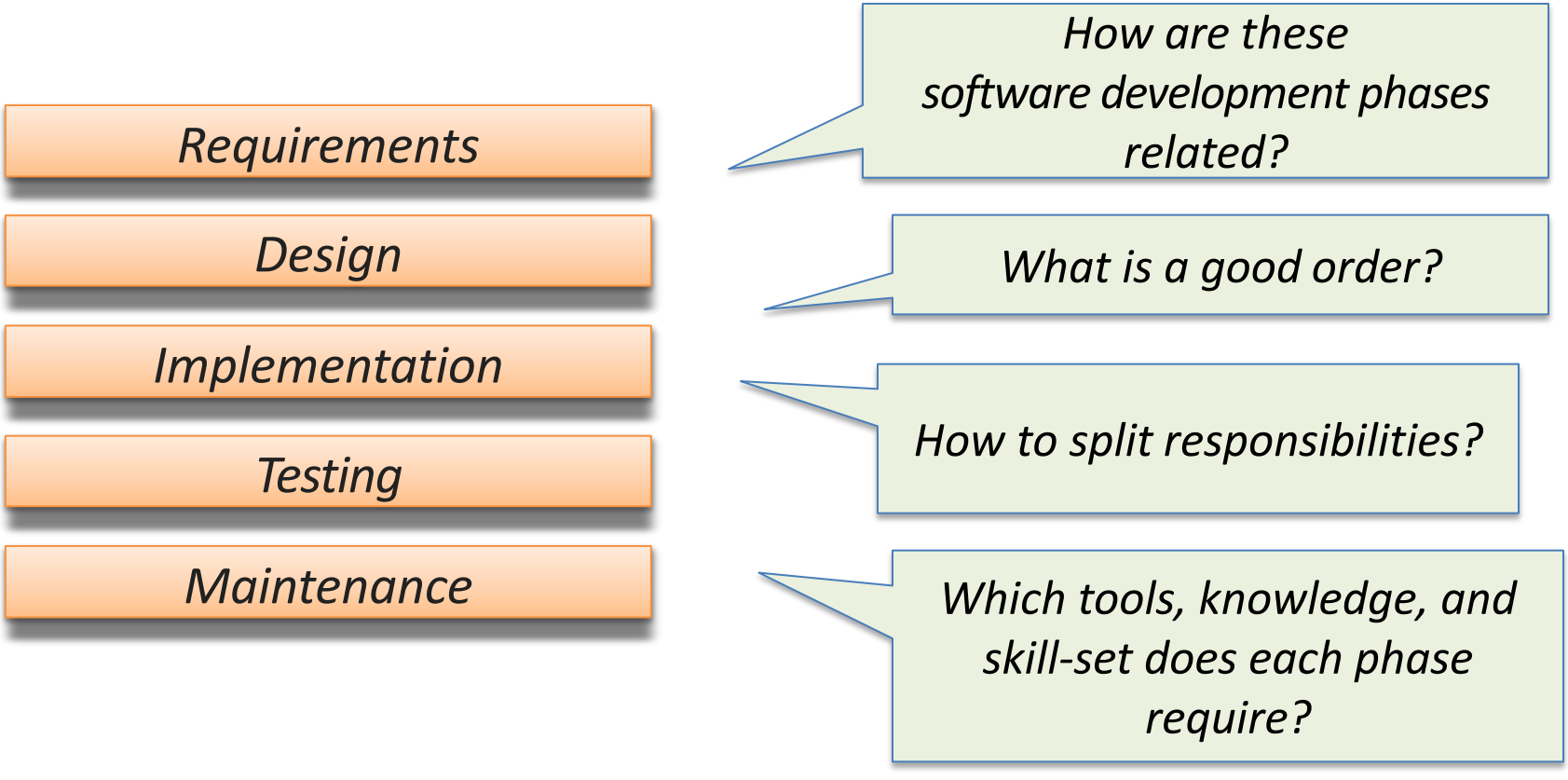
– Preventive – increasing software maintainability or reliability (deletion of obsolete capabilities, optimization, etc.)

Most of the time →

Essential Tasks in Software Engineering



Essential Tasks in Software Engineering

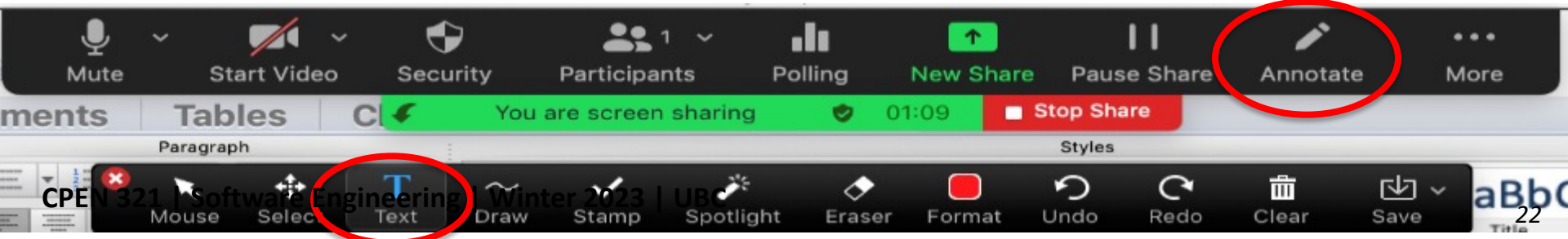


A Software Process

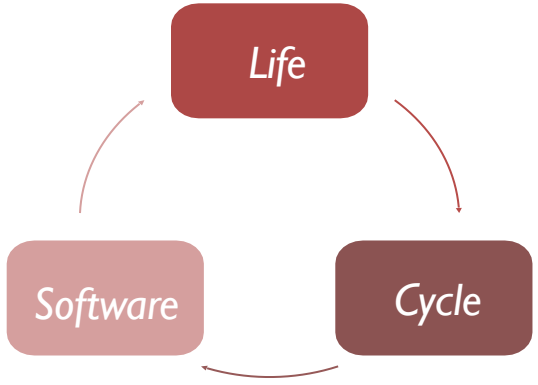
A process defines ***who*** is doing ***what***, ***when***, and ***how***
in the development of a software system

Which Software Process Models are You Familiar with?

Use Zoom Annotate Toolbox



Popular Software Development Process Models



No planning

- **Code-and-fix**: write code, fix it when it breaks

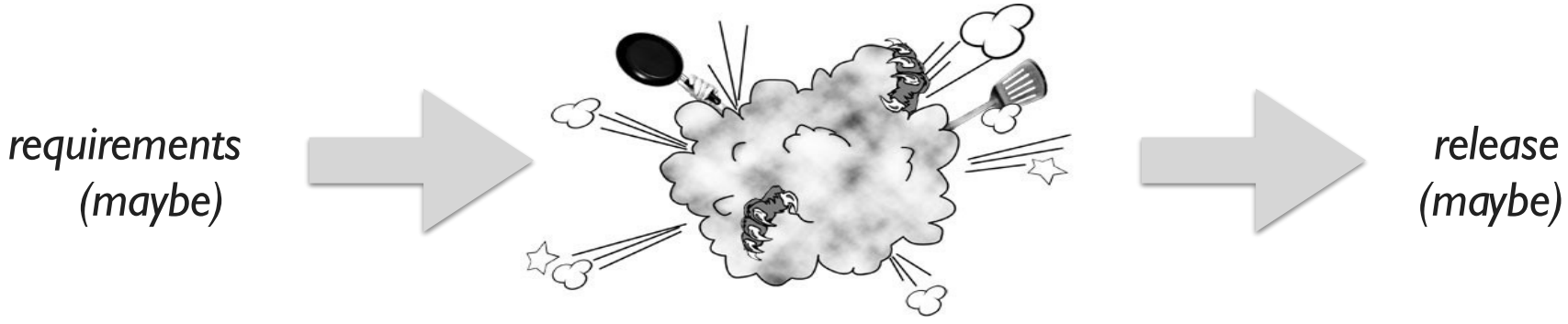
Sequential

- **Waterfall:** perform each phase in order (~1970s)

Iterative

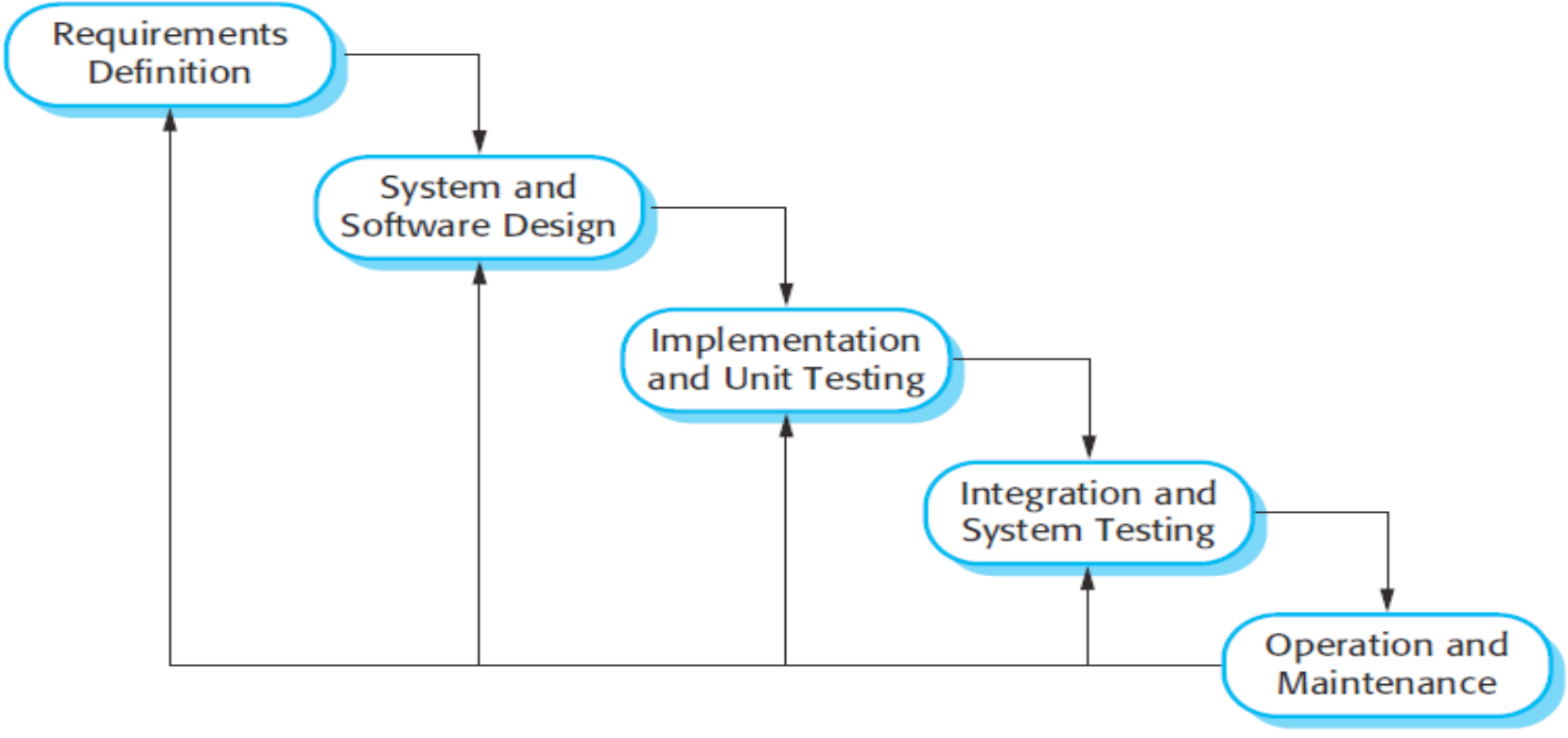
- **Staged Delivery:** waterfall-like beginnings, then, short release cycle
- **Evolutionary prototyping:** develop a skeleton system and evolve it for delivery
- **Spiral**: triage/figure out riskiest things first (1988)
- **Agile**: *a family of principles* promoting adaptive planning, evolutionary development, early delivery, and continuous improvement (1970-2005+)
  - Most popular: **Scrum** and **Kanban**

Code-and-fix Model

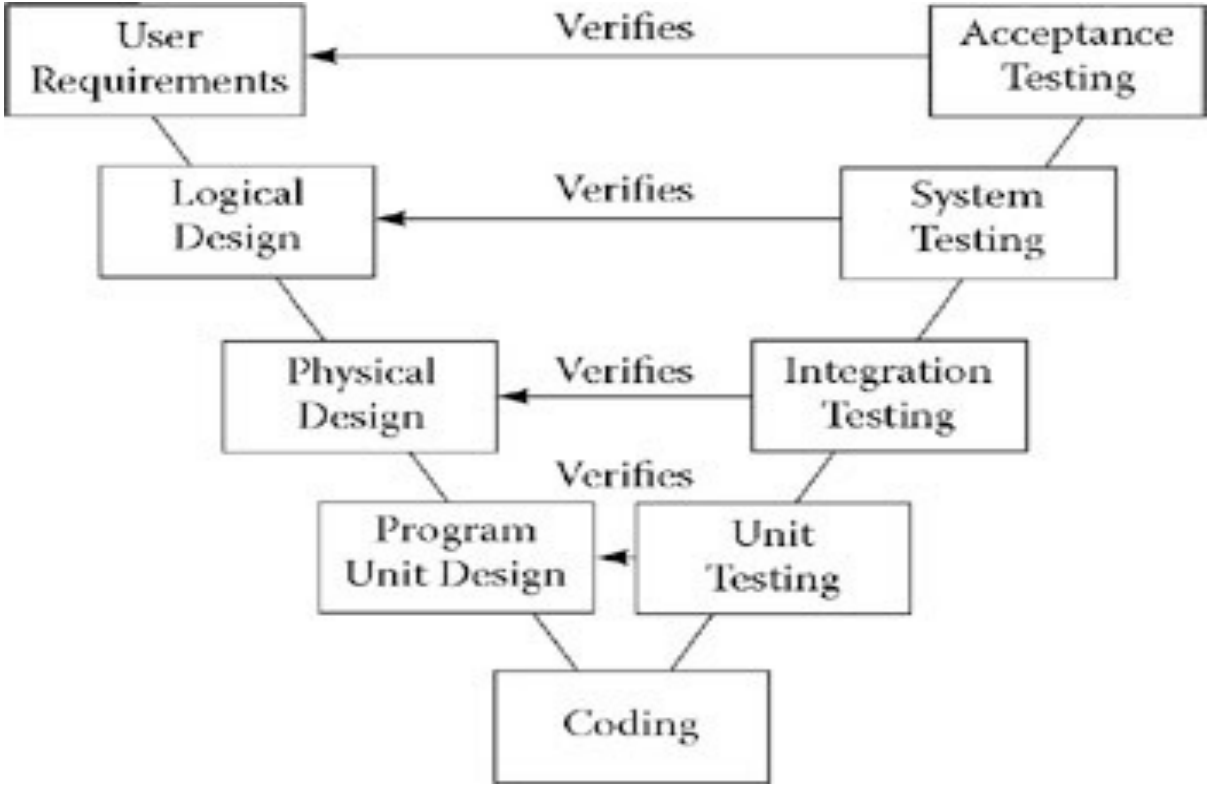


+ Applicable for very small projects and short-lived prototypes
- Unlikely to accommodate changes without a major design overhaul
- No good way to assess and mitigate risks

Fully Waterfall Model



Fully Waterfall V-Model



Is Waterfall Bad?

Waterfall Model Advantages

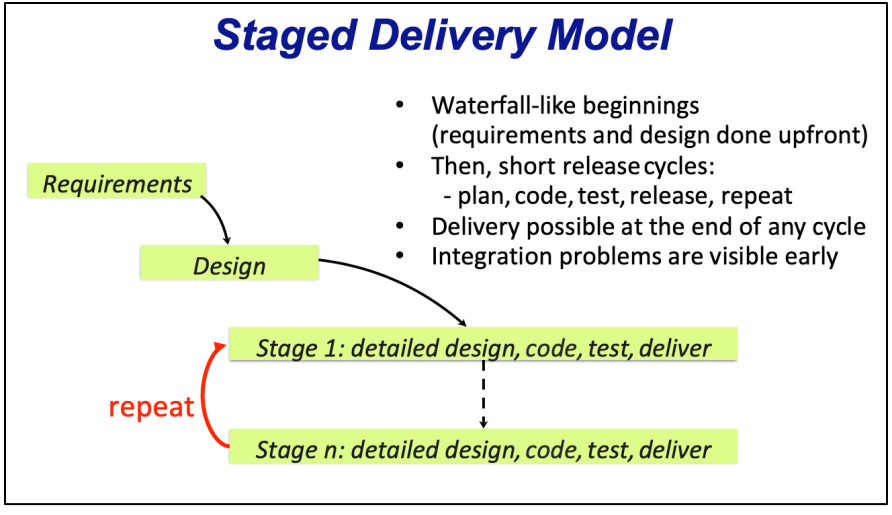
- Suitable for projects that are very well understood but complex
  - Tackles all planning up-front
  - The idea of no midstream changes equates to an efficient software development process

Waterfall Model Disadvantages

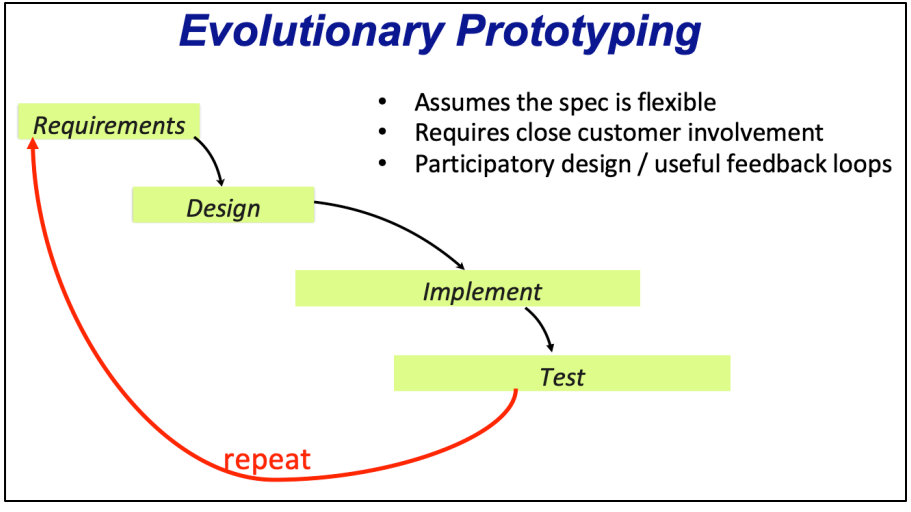
- Requires a lot of planning up-front (not always easy)
- No sense of progress until the very end
  - no code to show until almost done
- Delivered product may not match customer needs

Iterative Models

Improvements over Waterfall



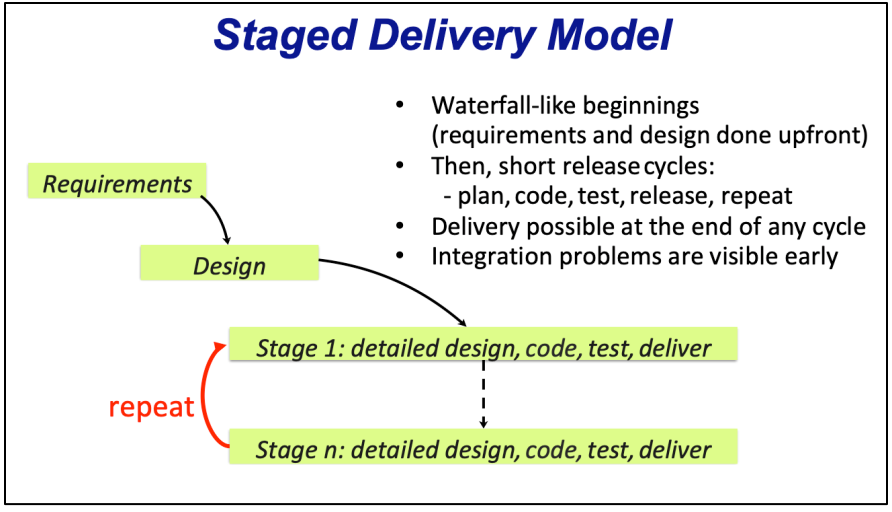
Assumes requirements are known up-front

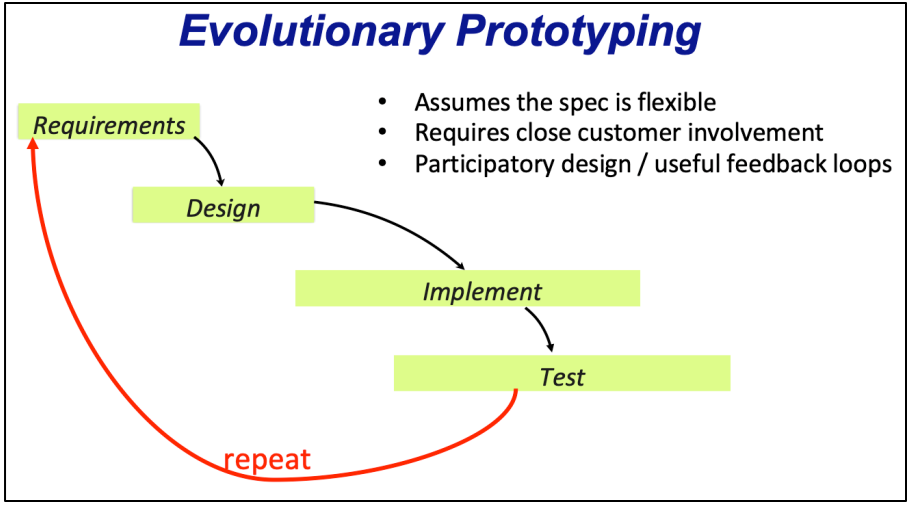


Does not assume requirements are known up-front

Iterative Models

Improvements over Waterfall





Assumes requirements are known up-front

Does not assume requirements are known up-front

Spiral (Risk Oriented)

in each iteration, identify and solve the sub-problems with the highest risk

Agile Manifesto (2001)

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

Agile Manifesto Principles

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile Development Practices

- Acceptance test-driven development (ATDD)
- **Backlogs (Product and Sprint)**
- Behavior-driven development (BDD)
- Business analyst designer method (BADM)
- **Continuous integration (CI)**
- **Cross-functional team**
- Domain-driven design (DDD)
- **Information radiators (scrum board, task board, visual management board, burndown chart)**
- **Iterative and incremental development (IID)**
- Pair programming
- Planning poker
- Refactoring
- **Retrospective**
- **Scrum events (sprint planning, daily scrum, sprint review and retrospective)**
- Story-driven modeling
- Test-driven development (TDD)
- **Timeboxing**
- User story
- User story mapping
- …

Not all are implemented by each agile method!

Not all are implemented by each team! ***Customize you process!***

SCRUM

Derived from the rugby term “scrum”
(Despite appearances, is a organized test of strength and skill)

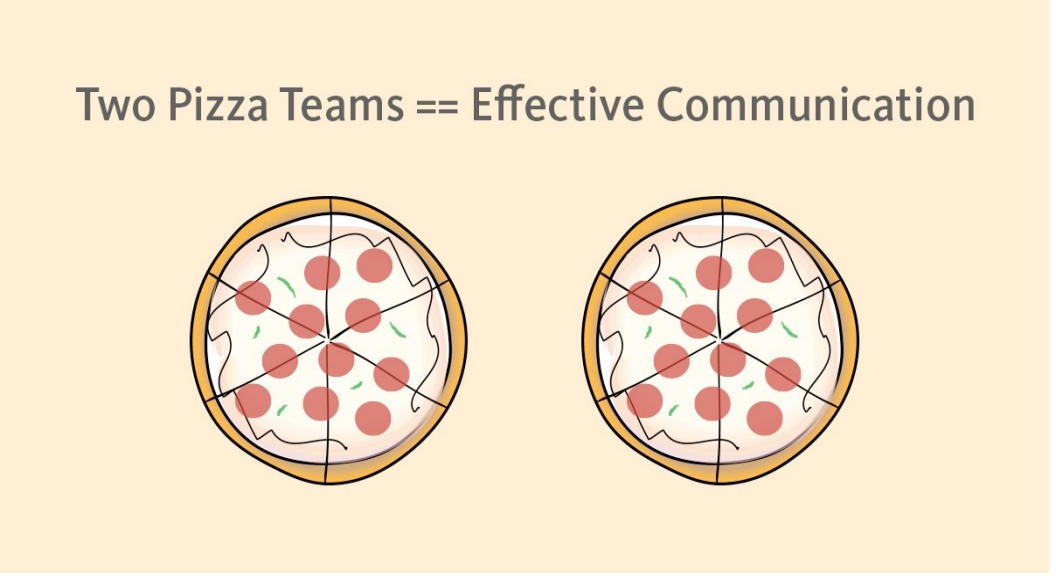
Scrum Roles

- Team members: the implementers of the product
- Scrum Master: behaves somewhat like a team lead or project manager, in working to resolve issues blocking team progress
- Product Owner: responsible for initial planning, prioritizing, represents the voice of the customer
- Users: customers or consumers of the product
- Managers: keep the team's organisation running smoothly

Scrum member rotate through roles
(especially Product Owner) each iteration

Scrum Teams

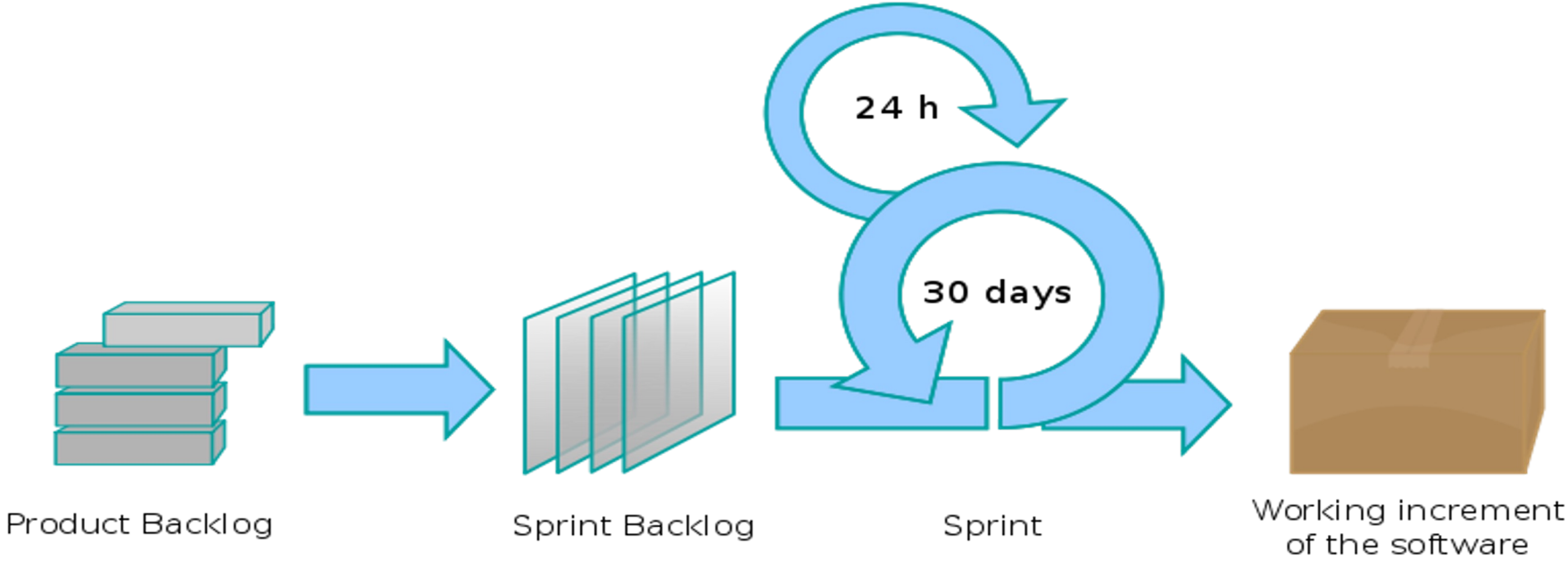
- "2 Pizza" team size (4 to 9 people)
  - "Scrum" inspired by frequent short meetings
  - 15 minutes every day at the same place and time
- Self-organizing



Sprint

- A sprint (or iteration) is the basic unit of development in Scrum.
- The sprint is a timeboxed effort; that is, it is restricted to a specific duration.
  - normally between one week and one month
  - two weeks is the most common case

Scrum Process



Source: Wikipedia

Sprint Planning

Note: Many online tools; adaptations to online

Sprint planning: Communicate the scope of work for the sprint

- Select product backlog items that can be completed in one sprint
- The recommended duration of the meeting is
  4 hours for a two-week sprint
  - First 2 hours: the scrum team selects the product backlog items they believe could be completed in that sprint
  - Second 2 hours: the development team identifies the detailed work (tasks) required to complete those product backlog items
- Result: a confirmed sprint backlog

Note: Many online tools; adaptations to online

The Daily Scrum Meetings

- Every day
- 15-30 minutes
- Whole world is invited
  - Helps avoid other unnecessary meetings
- Only team members can talk
- Stand-up

Note: Many online tools; adaptations to online

The Daily Scrum Meetings

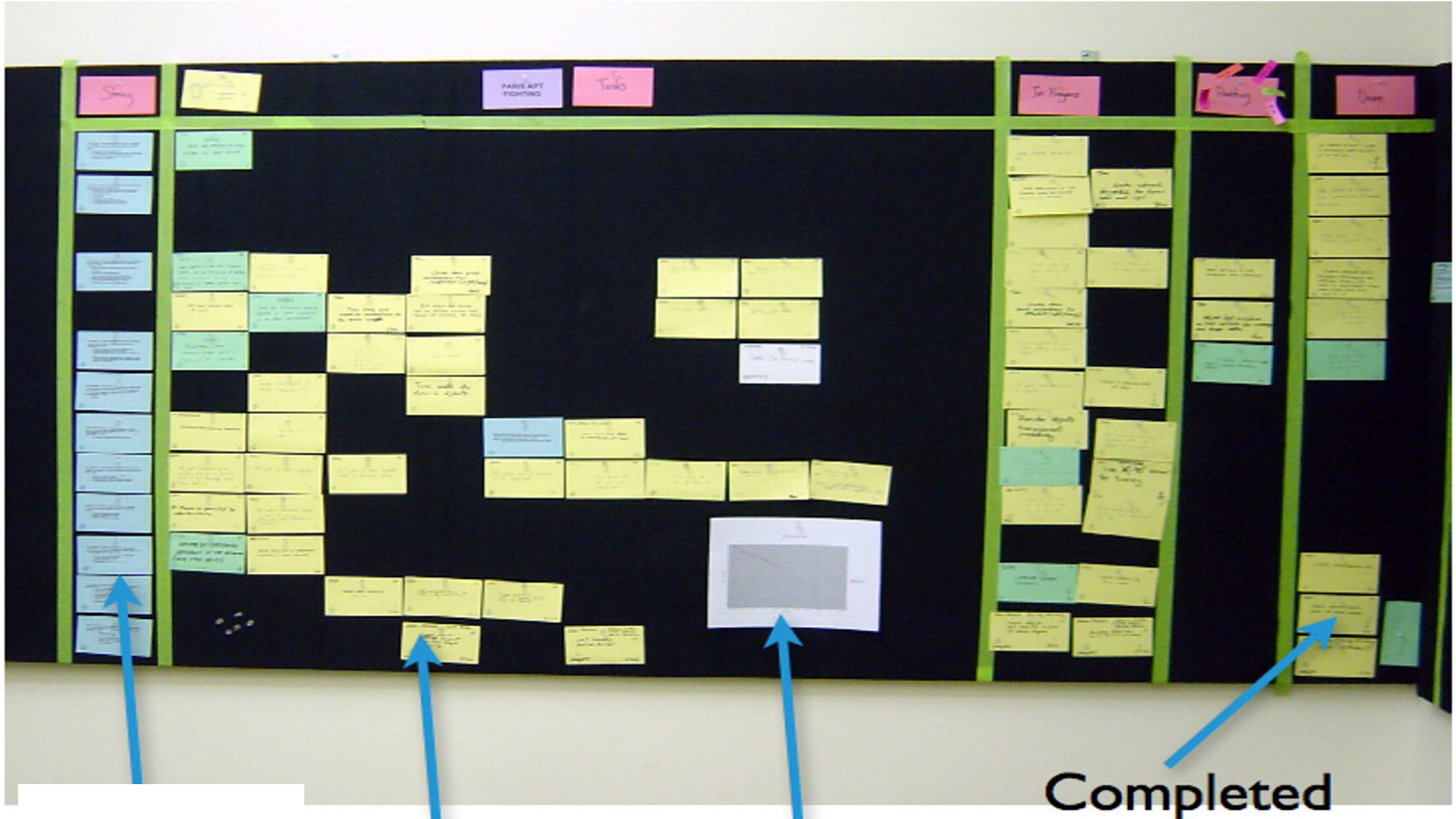
- Everyone answers 3 questions:
  1. What have you completed (relative to the Backlog) since the last Scrum meeting?
  2. What was (or is) in your way to completing this work?
  3. What will you do between now and the next Scrum meeting?

- The goal is *not* a status report for the ScrumMaster
  - Commitments in front of peers
  - The Scrum Master attempts to remove barriers
  - Not for solving technical problems

Keeping Track of Progress

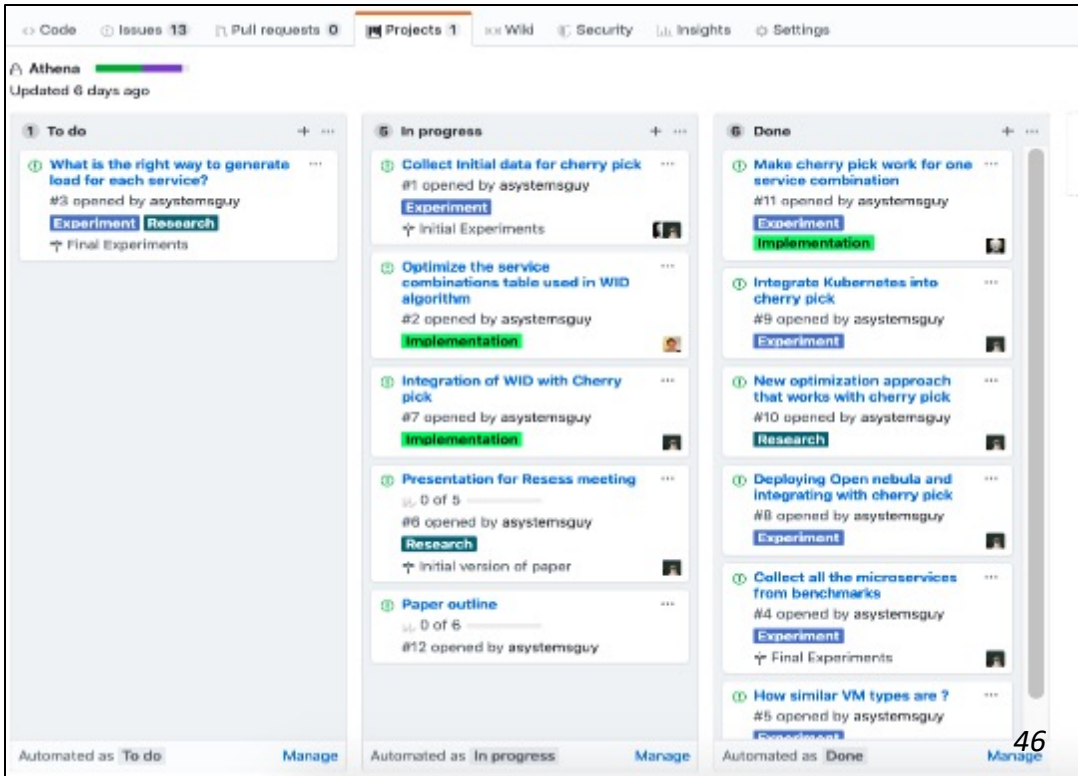
- The Task Board
- The Burndown Chart
- Sprint review and retrospective

The Task Board



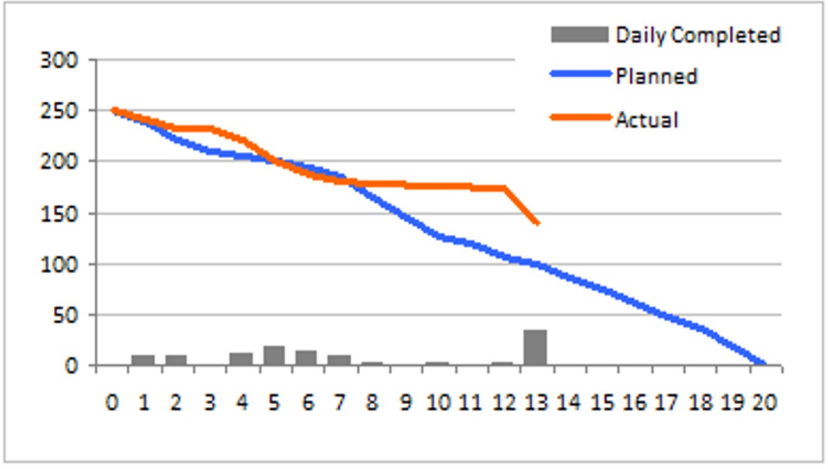
You Can (and Should) Practice It in Your Projects

- Can pick your favorite online version
- GitHub “native” solution: https://help.github.com/en/articles/about-project-boards



The Burndown Chart

- The X-axis: (real) time
  - hours, days, weeks, iterations, etc.
- The Y-axis: work remaining
  - developer-hours, developer-days, team-days, etc. required to complete the task
  - story points: the amount of effort required to implement a user story



- Are we progressing as planned? Are we ahead of the plan? Are we behind?

Sprint Review and Retrospective

At the end of a sprint, the team holds two events:

1. **Sprint review**:
   - Review the work that was completed and the planned work that was not completed
   - Present the completed work to the stakeholders (a.k.a. the demo)
     - Unfinished work cannot be presented
   - The team and the stakeholders collaborate on what to work on next

2. **Sprint retrospective**:
   - What went well during the sprint? What could be improved in the next sprint?
   - The team identifies and agrees on continuous process improvement actions

Scrum: Challenges and Solutions – 1/2

- Teams whose members are geographically dispersed or part-time
  - Recent improvements in technology have reduced the impact of these barriers
  - Yet, the Agile manifesto asserts that the best communication is face-to-face

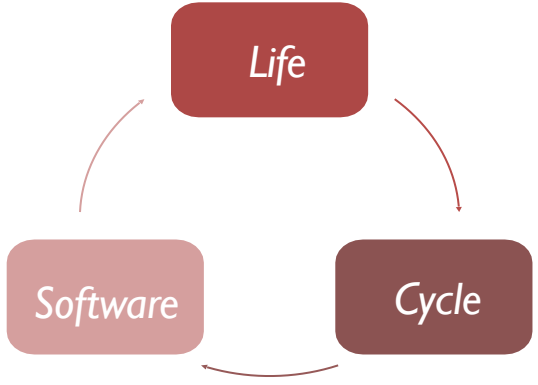
- Teams whose members have very specialized skills
  - In Scrum, developers should be able to work on any task or pick up work that another developer has started

Scrum: Challenges and Solutions – 2/2

- Products with many external dependencies
  - Deliveries of software from other teams can lead to delays and the failure of individual sprints
  - Teams should allocate time for supporting other teams (office hours)

- Products with regulated quality control (e.g., medical devices or vehicle control)
  - Product increments should be fully developed and tested in a single sprint
  - Less suited for products that need large amounts of safety and compliance testing

Popular Software Development Process Models



No planning

- **Code-and-fix**: write code, fix it when it breaks

Sequential

- **Waterfall:** perform each phase in order (~1970s)

Iterative

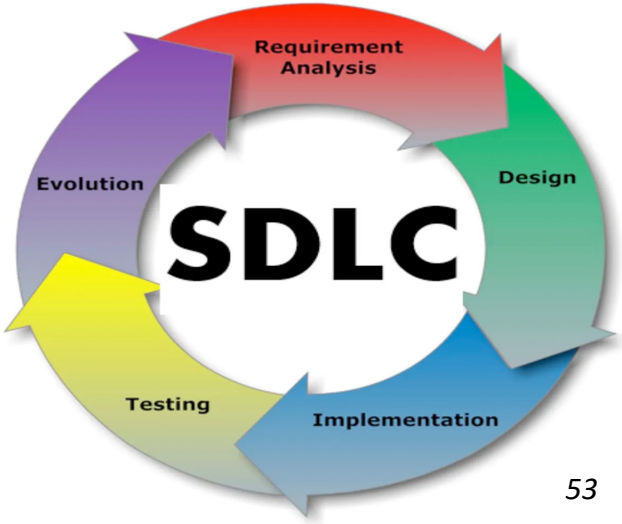
- **Staged Delivery:** waterfall-like beginnings, then, short release cycle
- **Evolutionary prototyping:** develop a skeleton system and evolve it for delivery
- **Spiral**: triage/figure out riskiest things first (1988)
- **Agile**: *a family of principles* promoting adaptive planning, evolutionary development, early delivery, and continuous improvement (1970-2005+)
  - Most popular: **Scrum** and **Kanban**

What's the best model?

- Consider
  - The task-at-hand
  - Customer involvement and feedback
  - Predictability
  - Quality control

Processes: Main Message

- Customize the processes depending on the product, organizational culture, team structure, needs, etc.
  - Process models are often combined or tailored to the environment
  - Think how much time to spend on each task and in which order!
- Follow processes, but do not over-emphasize process over product
  - Don't become a "slave" of the process



Exercise 1

Your team has to develop **the control software for a car's anti-lock braking system (ABS**). Which process model will you use and why?

A: Waterfall

B: Staged delivery (waterfall-like beginnings, then, short release cycles)

C: Evolutionary prototyping (evolve a skeleton system until delivery)

D: Spiral staged delivery (staged delivery + riskiest tasks first)

E: Spiral evolutionary prototyping (evolutionary + riskiest tasks first)

F: Agile / Scrum

Exercise 1: Good Answers

- Waterfall because the ABS system is **complex but already well-understood** system. The project can be fully planned out and there is no need to show customers early progress.
- It is a well-documented problem that is technically complex. Since the **specification will likely not change significantly**, the waterfall software process model is a good selection for this problem.
- Staged delivery because **customer interaction isn't really required during development** - we already know what ABS should do (requirements), and how it should work within the larger system (influences the design).

Question 1: Common Misconceptions

- Agile because the waterfall and staged delivery methods **would not place as much emphasis on testing**
- Agile because an ABS system is used in every car that is produced nowadays, so it is closely related to our life and personal safety. We need to **continuously pay attention to the improvements in technical excellence and design**. We can set up regular meetings to make progress on each part and meet stakeholders' requirements.
- Spiral because for software for a car's ABS, safety is the most important. Spiral can decrease risks and foresee problems.
- Staged delivery because it allows re-planning every iteration

Exercise 2

Your team has to develop **a hospital accounting system that replaces an existing system**. Which process model will you use and why?

A: Waterfall

B: Staged delivery (waterfall-like beginnings, then, short release cycles)

C: Evolutionary prototyping (evolve a skeleton system until delivery)

D: Spiral staged delivery (staged delivery + riskiest tasks first)

E: Spiral evolutionary prototyping (evolutionary + riskiest tasks first)

F: Agile / Scrum

Exercise 2: Good Answers

- Waterfall because this system is to replace an existing one, the **requirements are well laid out and understood according to past experiences**.
- Staged delivery because the minimum requirements for the system will be set by the capabilities of the existing system. The customers will not want to use it until it can at least be used as a replacement for it. **In staged delivery, the first waterfall-like phase could be used to bring it up to par with the existing system, and then the short release cycles after would improve the software based on experience**.
- Agile/scrum because it puts the premium on the end-user of the program/project. Since this program is replacing a program that is likely to be outdated, **but still correct and functioning, I would assume that the users want something more user-friendly**.

Exercise 2: Common Misconceptions

- Agile because customer satisfaction is important
- Spiral, to gradually replace the existing system
- Agile/Scrum because <the hospital> would like the new software to be **maintained and keep getting updates in the future**. The hospital accountants probably would not appreciate having to learn a new software every year if the software they are using stops getting updates and it becomes a security risk or something
- The hospital would appreciate a **functioning but perhaps an unpolished product in the short term**
- Agile because the product can/should be continuously improved. The best product is **not likely to be made on the first iteration** <…>.

What is Next?

- Prof. Rubin office hours
  - Now, here!

- Wed: Intro to UML (common specification language for software development)
  - Sahar Badihi