University of Glasgow

Advanced Professional Software Engineering 2024-25

Lecture 2 – Agile and Software Development Methodologies

S Waqar Nabi

(with thanks to Marco Tulio Valente, Universidade Federal de Minas Gerais)

Mar 2025

WORLD CHANGING GLASGOW

A WORLD TOP 100 UNIVERSITY

Agile and Software Development Methodologies

- Software Engineering – Why and What?
- Agile – Brief Review
- Technical Debt Management in Agile Teams
- DevOps, Continuous Integration, Continuous Delivery (CI/CD)

University of Glasgow

Software Engineering – Why and What?

Agile and Software Development Methodologies

- Software Engineering – Why and What?
- Agile – Brief Review
- Technical Debt Management in Agile Teams
- DevOps, Continuous Integration, Continuous Delivery (CI/CD)

NATO Conference (Germany, 1968)

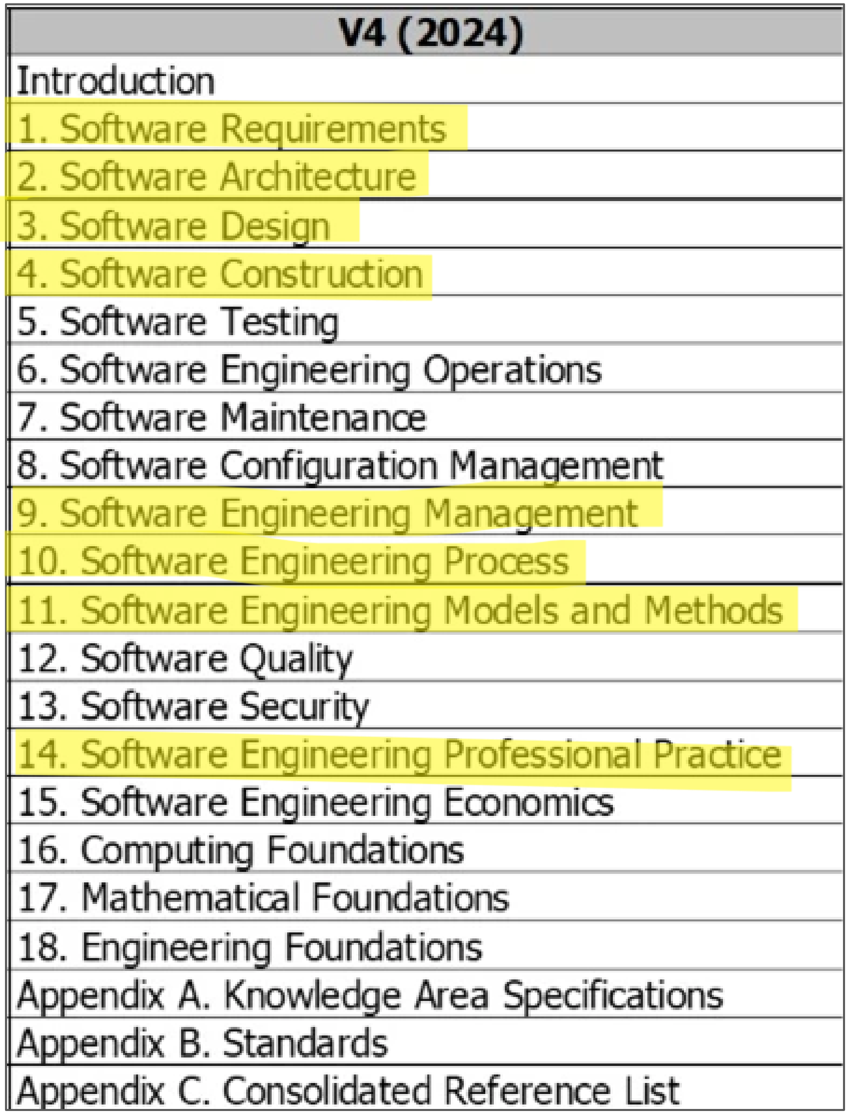
Often regarded as the event that established the field of Software Engineering

Working Conference on Software Engineering

Areas of Software Engineering

SWEBOK®

Guide to the Software Engineering Body of Knowledge

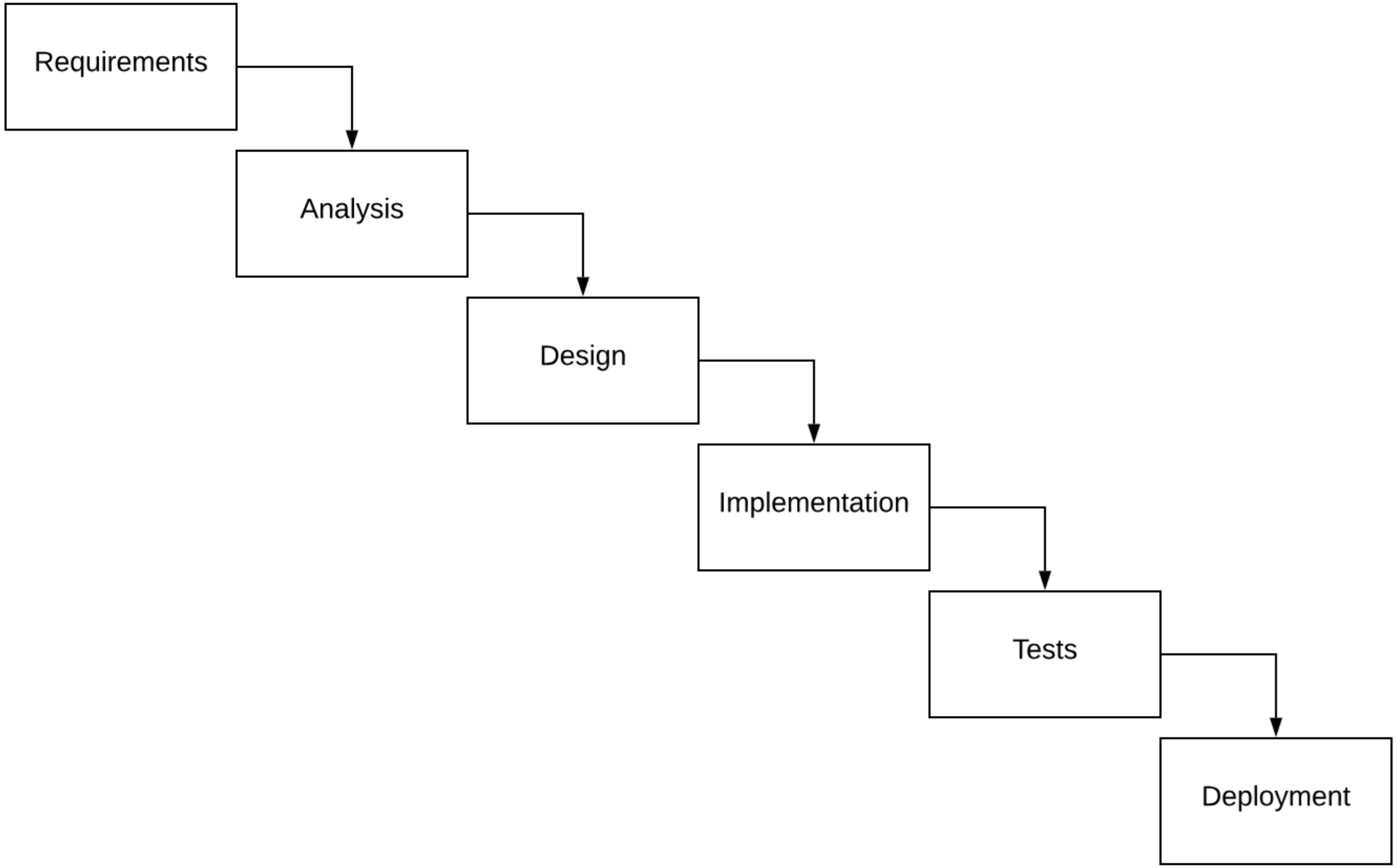


V4 (2024)
Introduction
1. Software Requirements
2. Software Architecture
3. Software Design
4. Software Construction
5. Software Testing
6. Software Engineering Operations
7. Software Maintenance
8. Software Configuration Management
9. Software Engineering Management
10. Software Engineering Process
11. Software Engineering Models and Methods
12. Software Quality
13. Software Security
14. Software Engineering Professional Practice
15. Software Engineering Economics
16. Computing Foundations
17. Mathematical Foundations
18. Engineering Foundations
Appendix A. Knowledge Area Specifications
Appendix B. Standards
Appendix C. Consolidated Reference List

Traditional Engineering

- Civil, mechanical, electrical, aviation, automotive, etc
- Has existed for thousands of years
- Two key characteristics:
    - Big Design Upfront (BDUF)
    - Sequential (Waterfall)

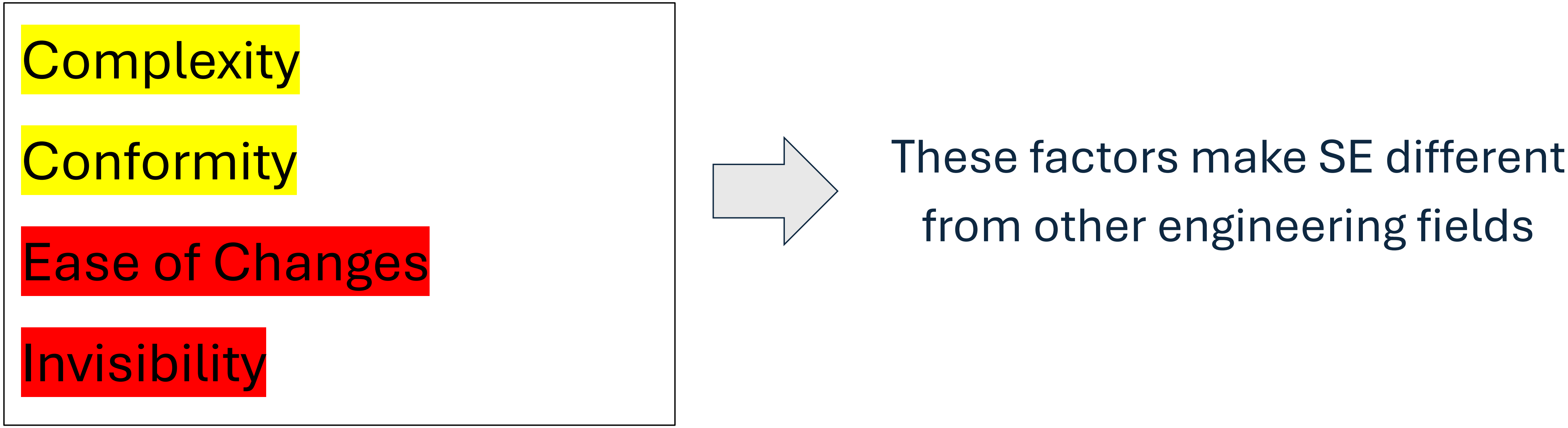
Thus, SE also started using Waterfall



Software is different

- Software Engineering ≠ Traditional Engineering
- Software ≠ (cars, bridges, houses, airplanes, phones, etc)
- Software ≠ (physical products)
- Software is abstract and flexible

Software is different



Complexity

Conformity

Ease of Changes

Invisibility

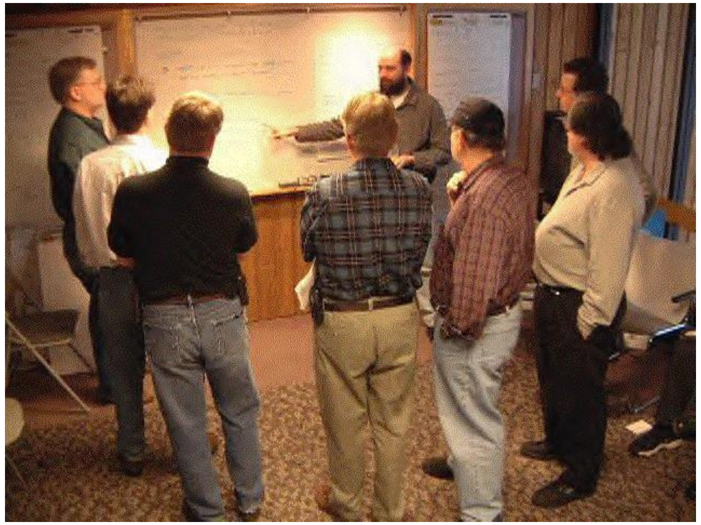
These factors make SE different from other engineering fields

Problems with Waterfall

1. Requirements often change
    - Complete requirements specification takes time
    - By the time it's finished, the world has changed!
2. Customers usually don't know what they want
3. Documentation is verbose and quickly becomes outdated

Agile Manifesto (2001)

- Meeting of 17 software engineers in Utah
- New model: incremental and iterative



https://siamchamnankit.co.th/history-some-pictures-and-pdfs-of-the-agile-manifesto-meeting-on-2001-a33c40bcc2b

Types of Software Systems

The ABC of Software Engineering

- Classification proposed by Bertrand Meyer
- Three types of software:
  - Type C (Casual)
  - Type B (Business)
  - Type A (Acute)

https://bertrandmeyer.com/2013/03/25/the-abc-of-software-engineering

Casual Systems (Type C)

- Very common
- Small systems, not very important
- Can have bugs; sometimes, they are temporary
- Implemented by 1-2 devs
- They don't benefit much from what we'll study here
- The main risk is over-engineering

Business Systems (Type B)

- Very important to an organization
- Systems that benefit from what we will study here
- Risk: if we do not use SE techniques, they may become a liability, rather than an asset for organizations

Acute Systems (Type A)

- Software where nothing can go wrong, as the cost is immense, in terms of human lives and/or $$$
- Mission-critical systems

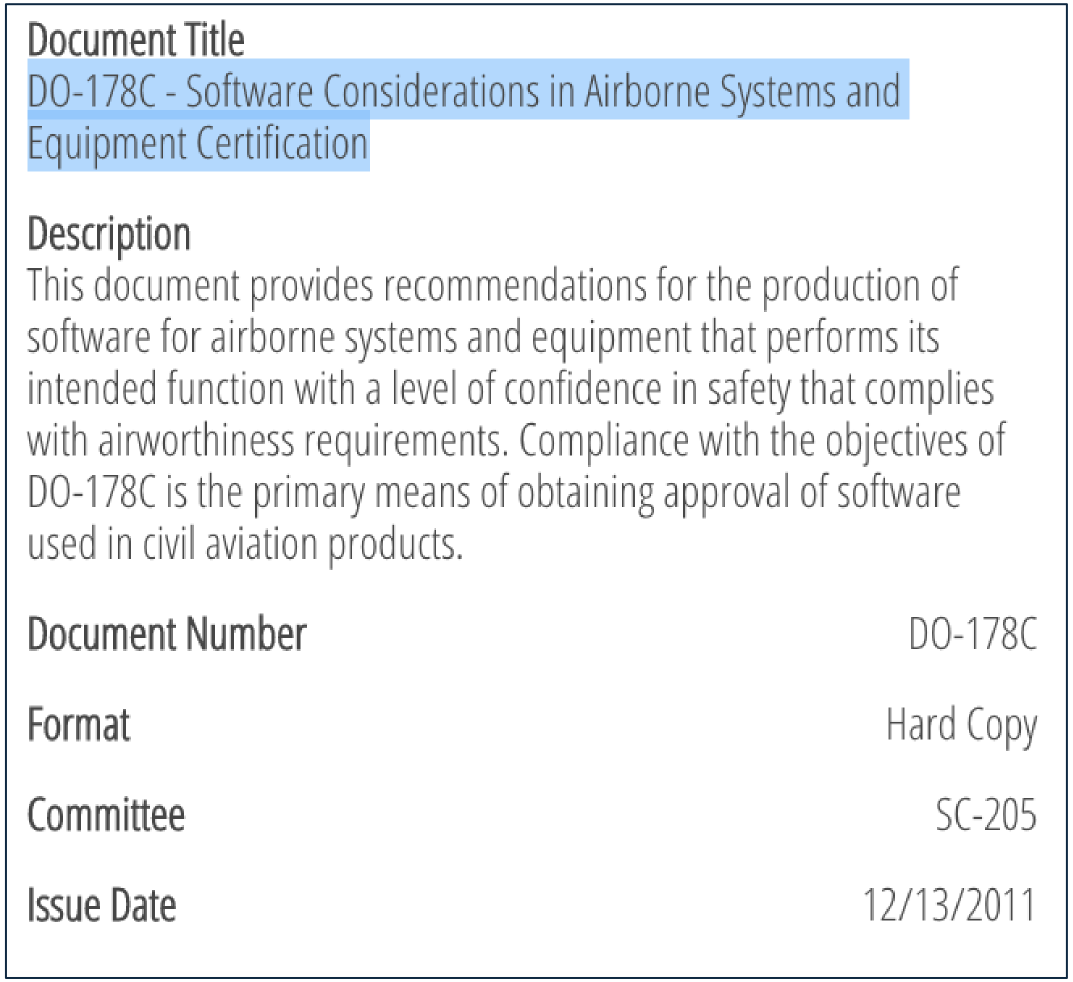
Subway

Aviation

Medicine

Acute Systems

- May require certifications
- Beyond the scope of our course
- We presume Type B system for our course project



Document Title

DO-178C - Software Considerations in Airborne Systems and Equipment Certification

Description

This document provides recommendations for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements. Compliance with the objectives of DO-178C is the primary means of obtaining approval of software used in civil aviation products.

Document Number	DO-178C
Format	Hard Copy
Committee	SC-205
Issue Date	12/13/2011

Agile – Brief Overview

Agile and Software Development Methodologies

- Software Engineering – Why and What?
- Agile – Brief Review
- Technical Debt Management in Agile Teams
- DevOps, Continuous Integration, Continuous Delivery (CI/CD)

Processes

- Activities that should be followed to build a software system
- Two main types:
  - Waterfall
  - Agile

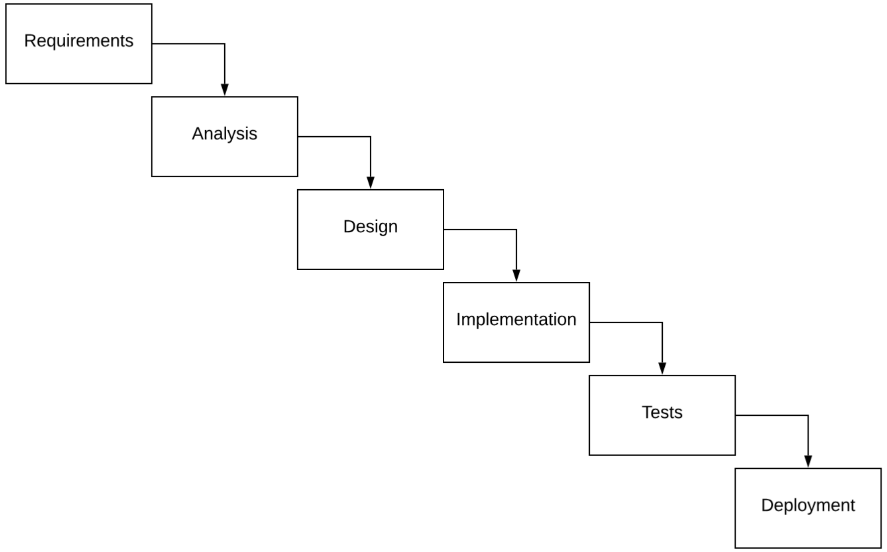
Why Follow a Process at All?

- Contemporary software too complex to be developed by a single developer
  - Software Engineering takes places in teams
- Teams require a development *process*, which allows them to:
  - organize
  - coordinate
  - motivate
  - evaluate
  - promote productivite
  - align with organization goals
  - within team members: clarify expectations and reduce misalignment
- At least a lightweight process is important

But Waterfall did not work with software

Challenge #1: Requirements

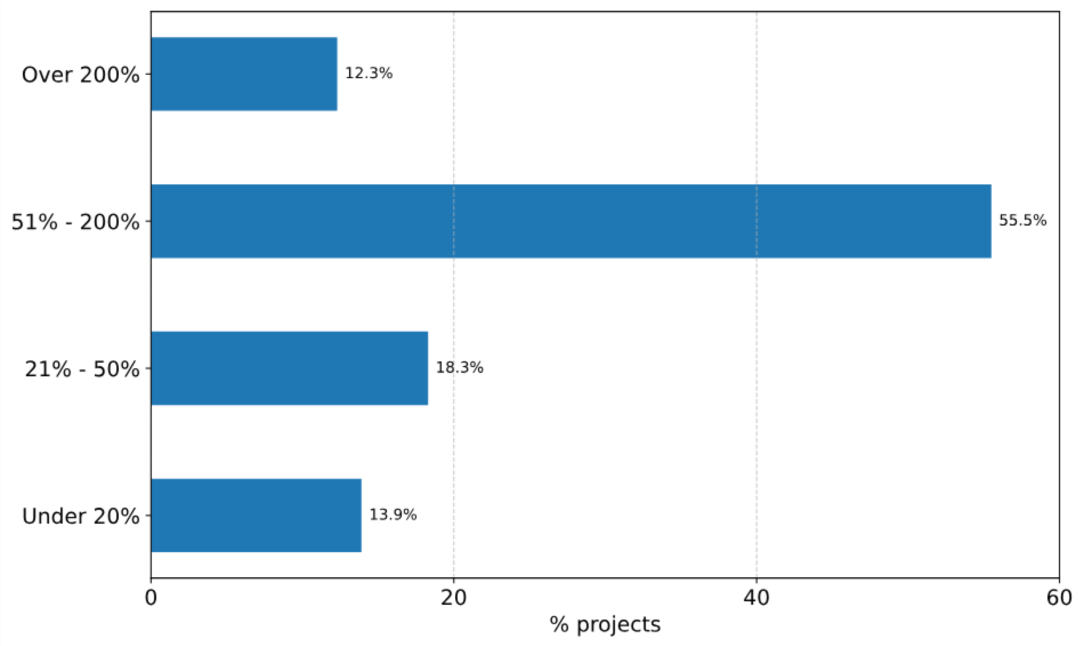
- Often, customers don't know what they want:
    - Feature space is “infinite” or hard to predict
    - World changes!
- It's not possible anymore to spend:
    - 1 year defining the requirements
    - 1 year designing the system
    - 1 year implementing the system
    - etc
- When the software is ready, it may already be obsolete!



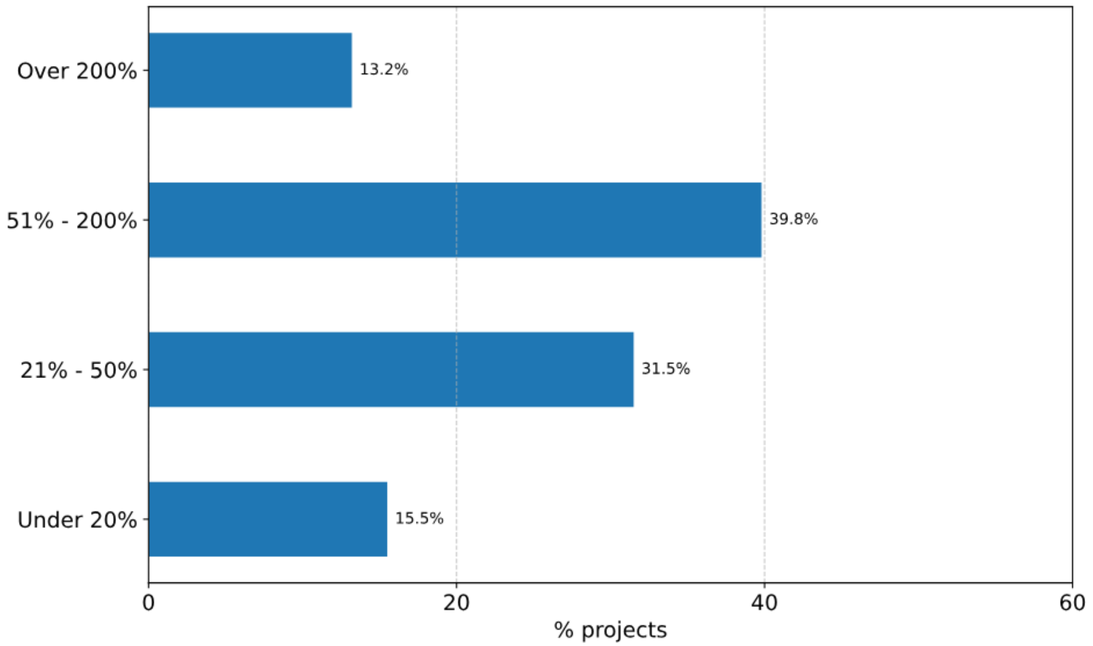
Challenge #2: Detailed Documentation

- Verbose and of limited use
- In practice, not used during the implementation phase
- Plan-and-document did not work with software

Evidence based pivot

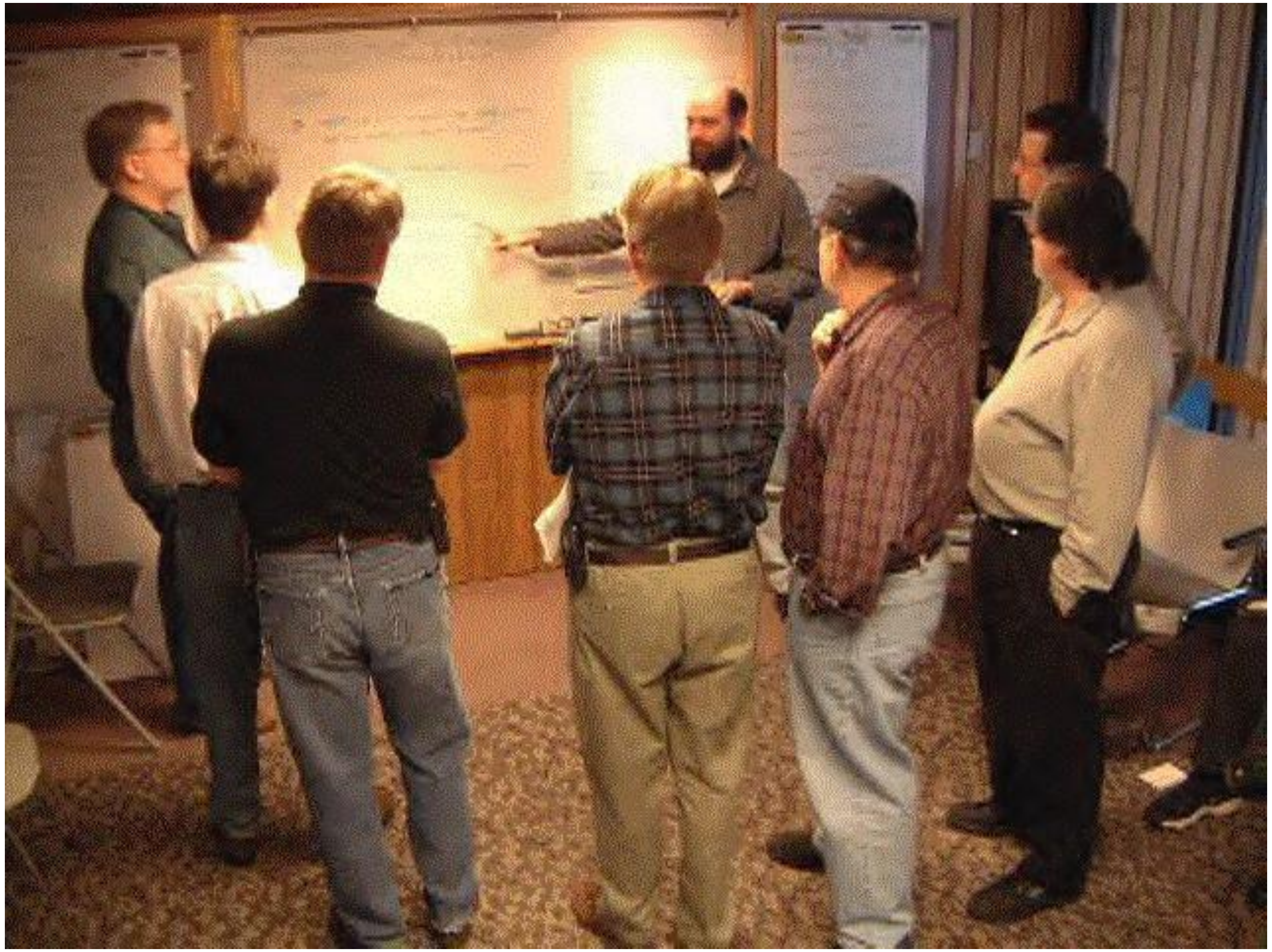


CHAOS Report (1994): percentage of projects exceeding their deadlines (for each range of overrun)



CHAOS Report (1994): percentage of projects exceeding their budgets (for each range of overrun)

Agile Manifesto (2001)



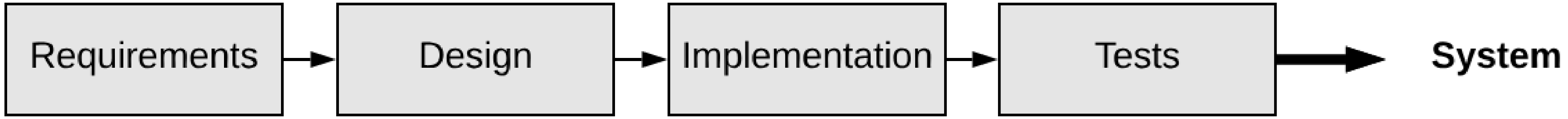
The Manifesto reads:

“Through this work, we have come to value:

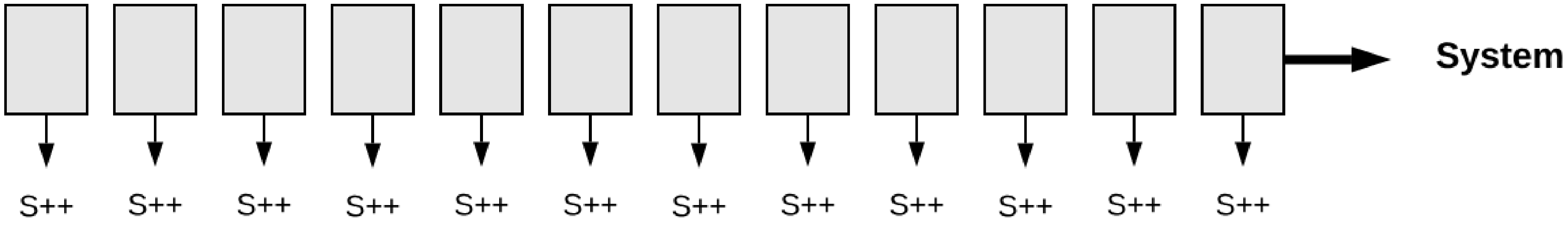
- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan.”

Key idea: iterative development

Waterfall



Agile



S++ = product increment

Iterative Development

- Let's consider a large and complex system
- What's the smallest feature increment we can deliver in 15 days and validate with users?
- Validation is very important
- Customers usually don't know what they want!

Other characteristics

- Less emphasis on documentation
- Less emphasis on big design upfront (BDUF)
- Customer involvement
- New programming practices: tests, refactoring, CI/CD, etc.

Agile Methods

Agile Methods

- Agile principles are quite broad, generic, non-specific.

- To give consistency to agile ideas and make them more concrete and actionable:
  - Define specific methods, even if lightweight
  - Workflow, events, roles, practices, principles, etc

- **Remember: These are all just recommendations!**

Agile Methods

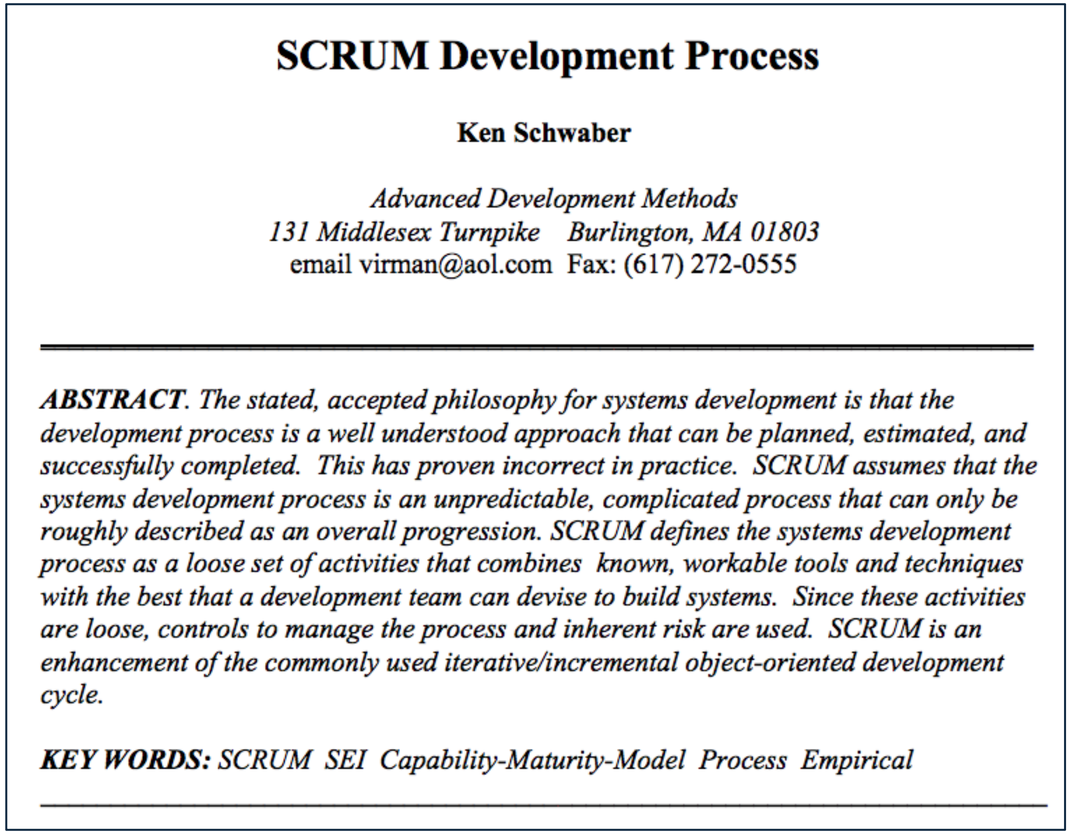
- Extreme Programming (XP)
- Scrum
- Kanban

Scrum

The default method to use for your project in this course

Scrum

- Proposed by Jeffrey Sutherland and Ken Schwaber



SCRUM Development Process

Ken Schwaber

Advanced Development Methods
131 Middlesex Turnpike   Burlington, MA 01803
email virman@aol.com Fax: (617) 272-0555

***ABSTRACT**. The stated, accepted philosophy for systems development is that the development process is a well understood approach that can be planned, estimated, and successfully completed. This has proven incorrect in practice. SCRUM assumes that the systems development process is an unpredictable, complicated process that can only be roughly described as an overall progression. SCRUM defines the systems development process as a loose set of activities that combines known, workable tools and techniques with the best that a development team can devise to build systems. Since these activities are loose, controls to manage the process and inherent risk are used. SCRUM is an enhancement of the commonly used iterative/incremental object-oriented development cycle.*

***KEY WORDS:** SCRUM SEI Capability-Maturity-Model Process Empirical*

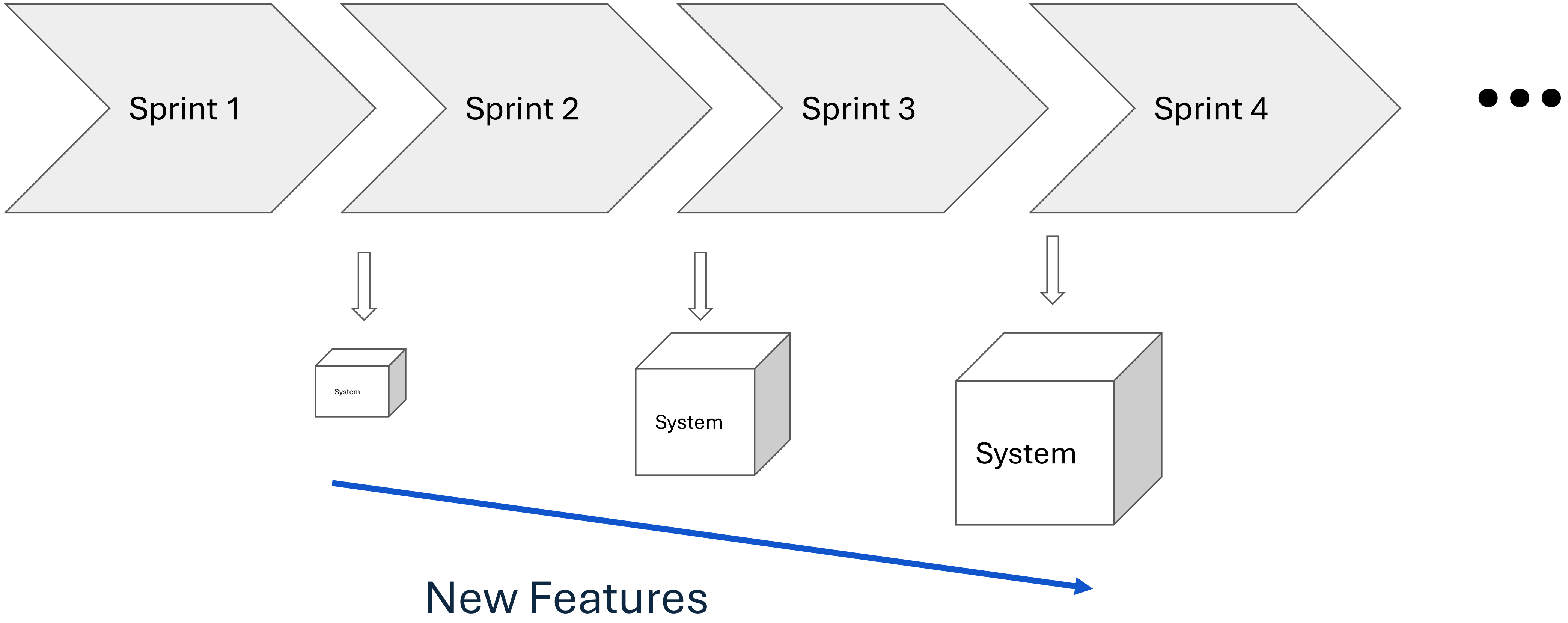
OOPSLA 1995

Scrum – Big Idea

Freeze requirements during short iterations

Main event: Sprints

- Up to 1 month, usually 15 days



What is done in a sprint?

- Team implements user stories
- User stories ⇒ features
- Example from a Q&A Forum:

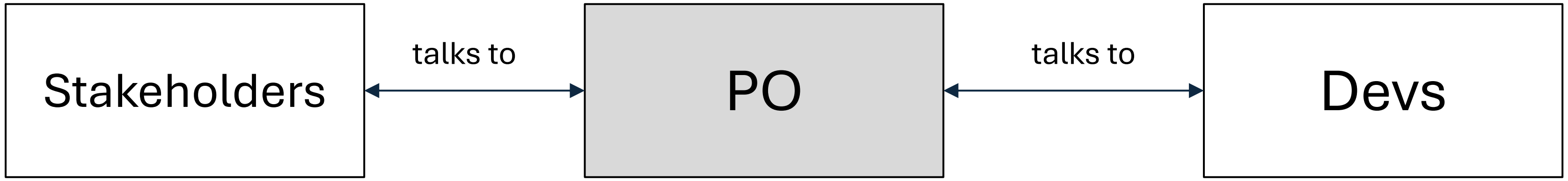
A logged-in user should be able to post questions. Since it’s a programming forum, questions may incorporate code blocks, which must be presented in a differentiated layout.

User stories are written on cards

Who writes the stories?

- Product Owner (PO): mandatory role in Scrum
- Expert in the problem domain

Scrum



- During the sprints, PO explains stories to devs
- We change from formal/written to informal/verbal specs

What does a PO do?

- Write the user stories
- Explain the user stories to the devs
- Define the "acceptance tests" for the user stories
- Prioritize the user stories

Product Backlog

- List of user stories (and other important work items)
- Two characteristics:
  - Prioritized: top stories have higher priority
  - Dynamic: stories can come and go

Which stories will be implemented in the next sprint?

- Decision taken at the start of the sprint
- In a meeting called sprint planning:
  - PO proposes stories they'd like to see implemented
  - Devs decide if they have the velocity to implement them

Important

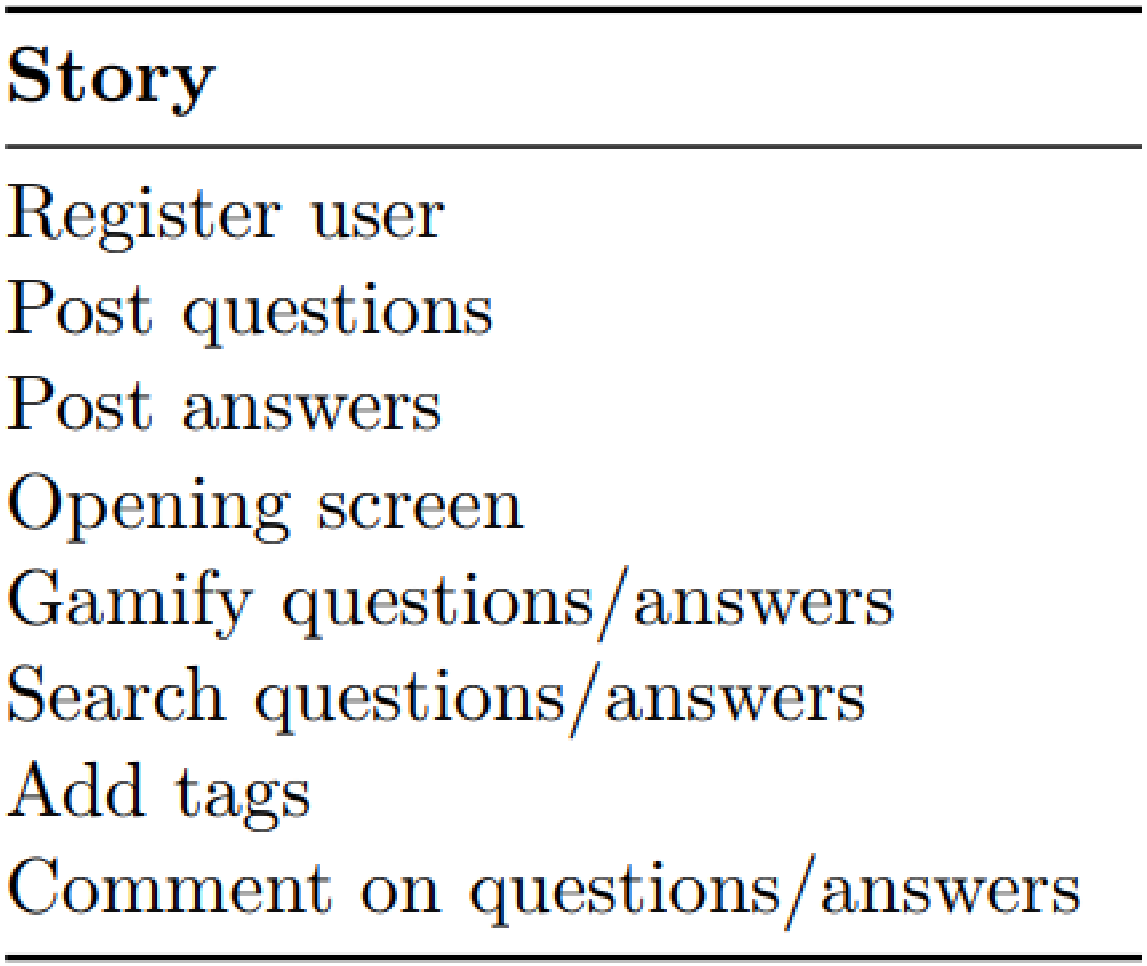
- In Scrum teams, everyone is at the same hierarchical level
- The PO is not the manager of the Devs, but an expert in the product domain
- Devs, as the technical experts, can say they won't be able to implement everything the PO wants in a single sprint

Sprint Planning

- 1st part: team defines the stories of the sprint
- 2nd part: stories are broken down into tasks, which are allocated to devs

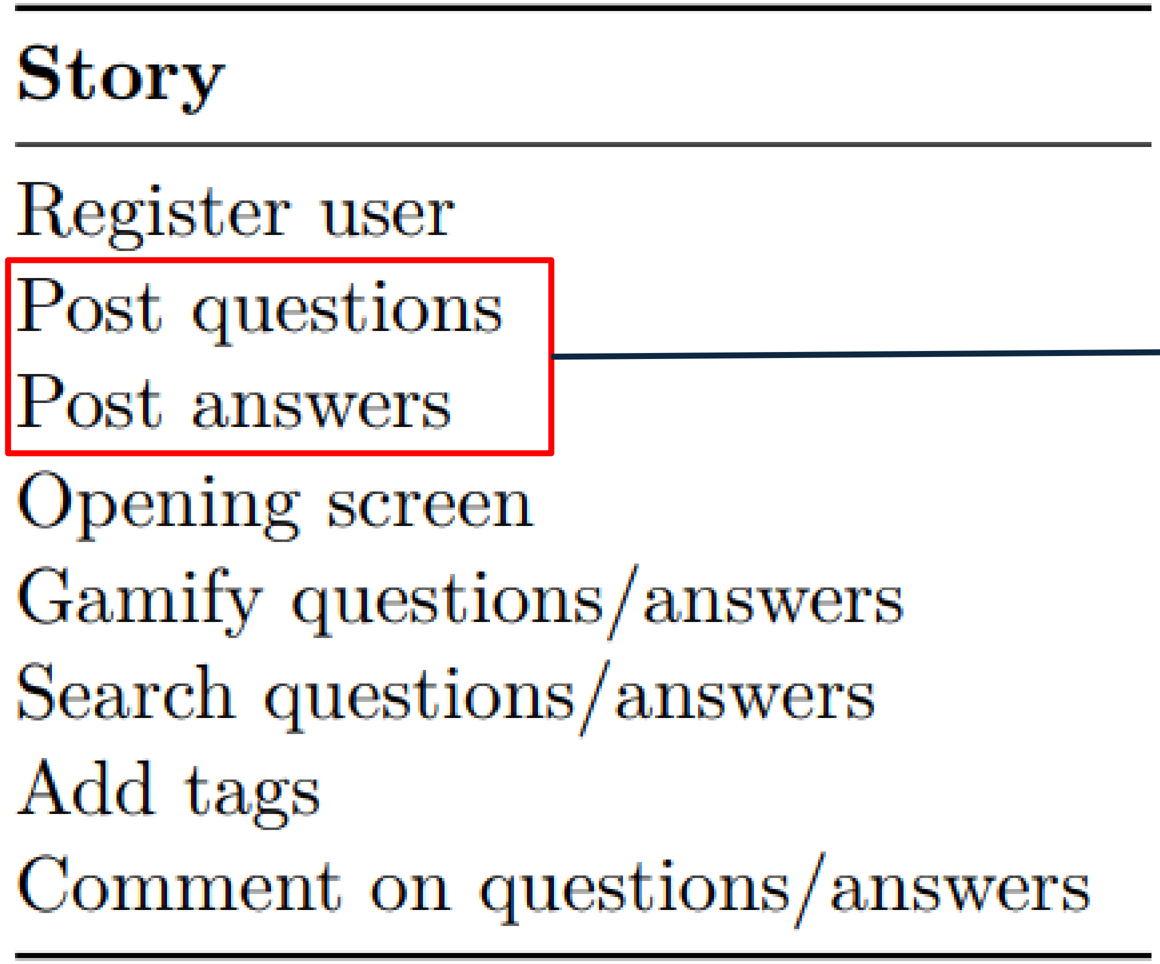
Example: Q&A Forum

Product Backlog



Story
Register user
Post questions
Post answers
Opening screen
Gamify questions/answers
Search questions/answers
Add tags
Comment on questions/answers

Product Backlog



Story
Register user
Post questions
Post answers
Opening screen
Gamify questions/answers
Search questions/answers
Add tags
Comment on questions/answers

stories selected
for the next sprint

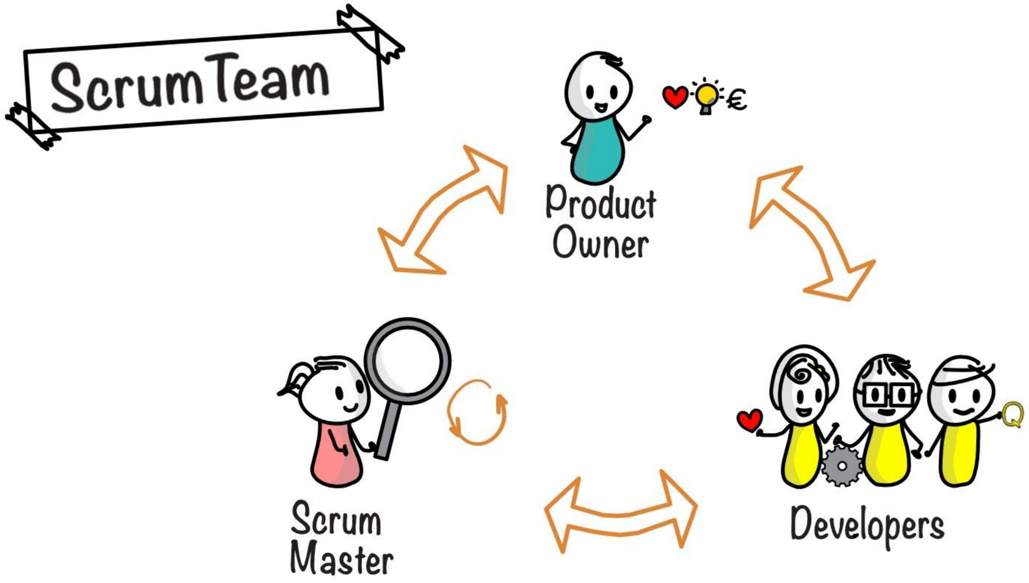
Sprint Backlog: tasks of the selected stories

- Install the database and create initial tables
- Install Node.js and Express
- Create and test a route using Express
- Implement the question page in the frontend
- Implement the backend logic for creating questions
- Implement the answer page in the frontend
- Implement the backend logic for answering question

Sprint is ready to start!

Scrum Teams

- Small (size of a basketball to a football team)
- Including 1 PO and 1 Scrum Master
- Cross-functional: devs, UX designers, data scientists, etc.



Scrum Master (SM)

- Expert who helps the team to follow Scrum
- SM is not the manager of the team, but a serving leader
- “Remover” of non-technical impediments
  - Example: developers don't have good computers
- SM can also collect process metrics
- SM can be part of more than one team

More Scrum Events

Daily Meetings / Standups (15 min)

- Each participant answers three questions:
  - What I did yesterday
  - What I intend to do today
  - What obstacles I'm facing (if any)
- Goals: Improve communication & anticipate problems

Sprint ends with two events:

Review and Retrospective

Sprint Review

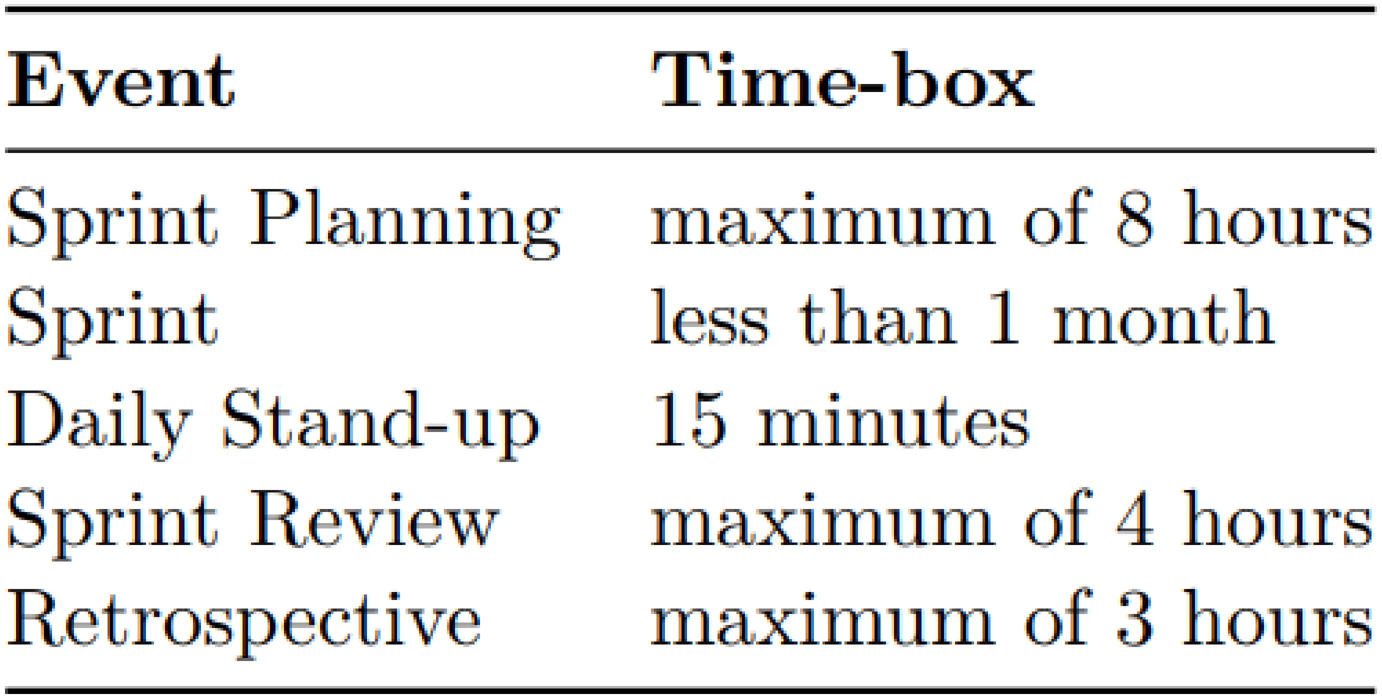
- Team shows the sprint's outcome to PO and stakeholders
- Implementation of each story can be:
    - Approved
    - Partially approved
    - Rejected
- In the last two cases, it goes back to the product backlog

Sprint Retrospective

- Last event of the sprint
- Team gathers to discuss two questions:
  - What went well in the sprint?
  - How can we improve?
- Goal: continuous improvement
- It should be a blameless meeting

More Scrum Concepts

Time-box: all events have a well-defined duration

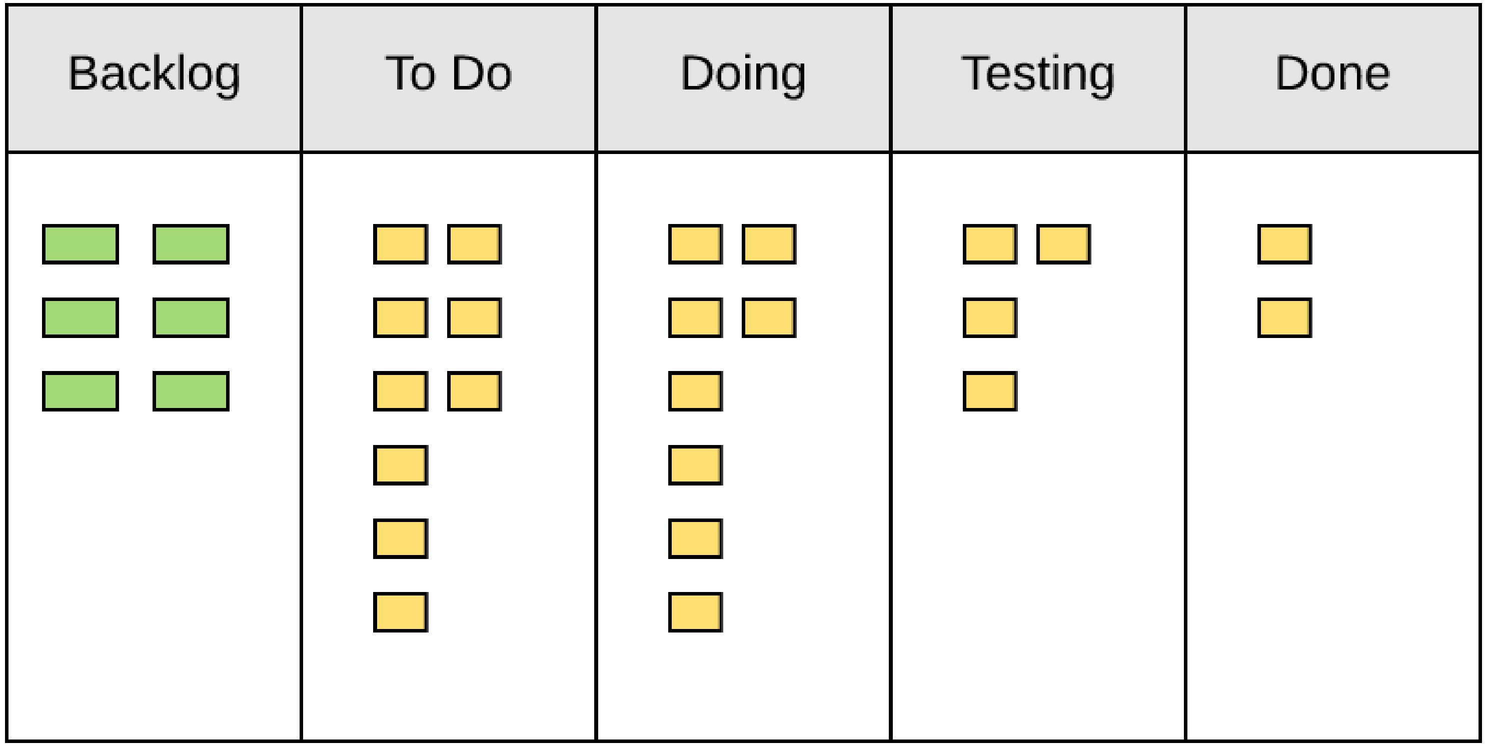


Event	Time-box
Sprint Planning	maximum of 8 hours
Sprint	less than 1 month
Daily Stand-up	15 minutes
Sprint Review	maximum of 4 hours
Retrospective	maximum of 3 hours

Done Criteria: used to consider stories done

- Also called DoD (Definition of Done)
- Example:
    - Unit tests with coverage ≥ 75%
    - Code review by another dev
    - Update documentation (if API has changed)
    - Performance test (for certain stories)

Scrum Board



Backlog

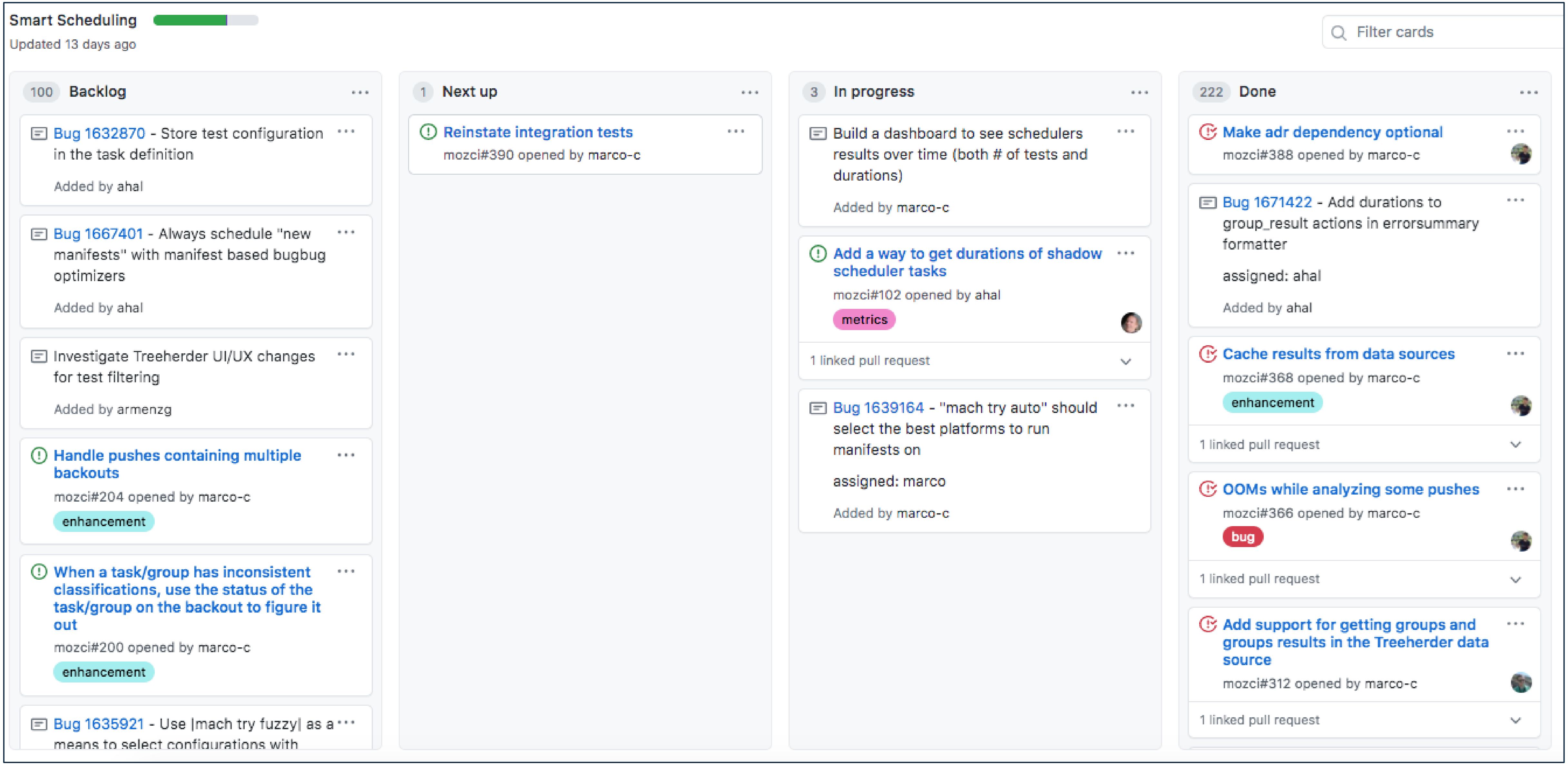
To Do

Doing

Testing

Done

Example: Mozilla project (using GitHub Projects)



Smart Scheduling

Updated 13 days ago

100 Backlog

Bug 1632870 - Store test configuration in the task definition

Added by ahal

Bug 1667401 - Always schedule "new manifests" with manifest based bugbug optimizers

Added by ahal

Investigate Treeherder UI/UX changes for test filtering

Added by armenzg

Handle pushes containing multiple backouts

mozci#204 opened by marco-c

enhancement

When a task/group has inconsistent classifications, use the status of the task/group on the backout to figure it out

mozci#200 opened by marco-c

enhancement

Bug 1635921 - Use |mach try fuzzy| as a means to select configurations with

1 Next up

Reinstate integration tests

mozci#390 opened by marco-c

3 In progress

Build a dashboard to see schedulers results over time (both # of tests and durations)

Added by marco-c

Add a way to get durations of shadow scheduler tasks

mozci#102 opened by ahal

metrics

1 linked pull request

Bug 1639164 - "mach try auto" should select the best platforms to run manifests on

assigned: marco

Added by marco-c

222 Done

Make adr dependency optional

mozci#388 opened by marco-c

Bug 1671422 - Add durations to group_result actions in errorsummary formatter

assigned: ahal

Added by ahal

Cache results from data sources

mozci#368 opened by marco-c

enhancement

1 linked pull request

OOMs while analyzing some pushes

mozci#366 opened by marco-c

bug

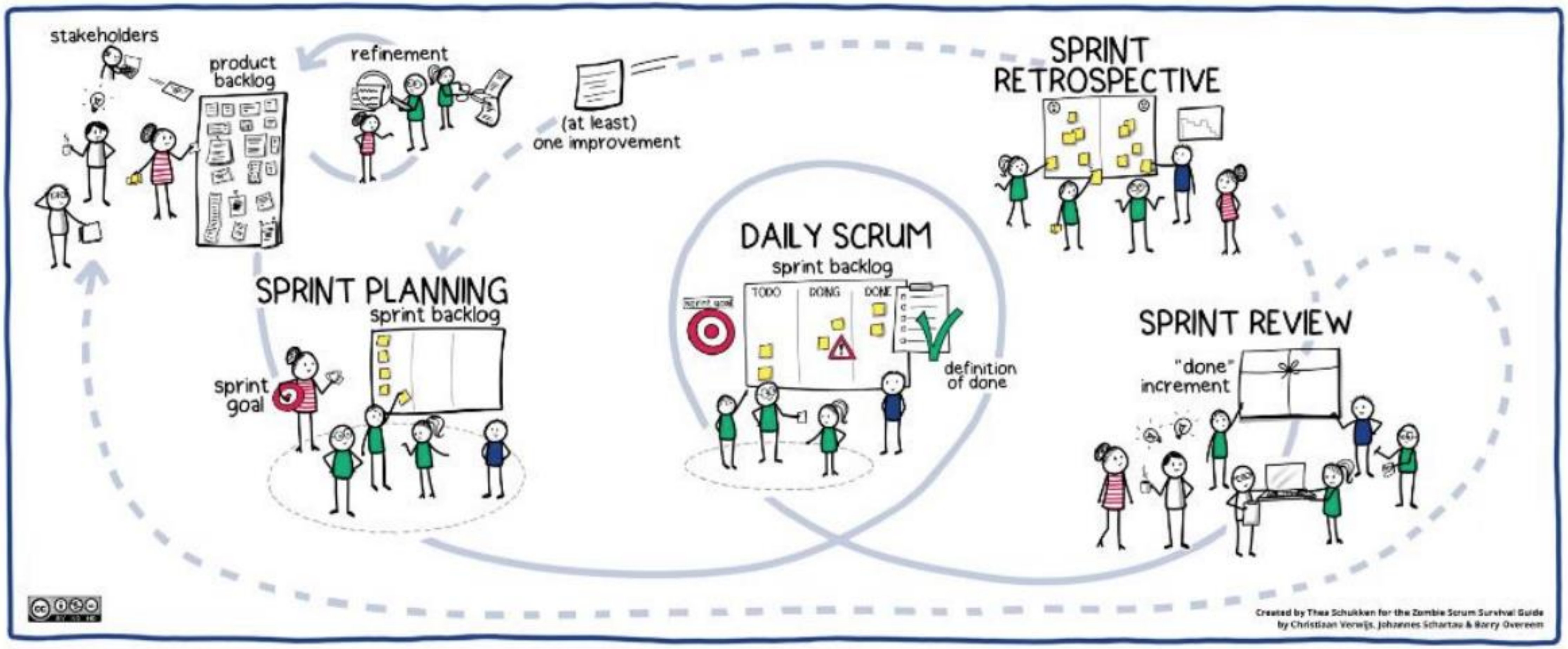
1 linked pull request

Add support for getting groups and groups results in the Treeherder data source

mozci#312 opened by marco-c

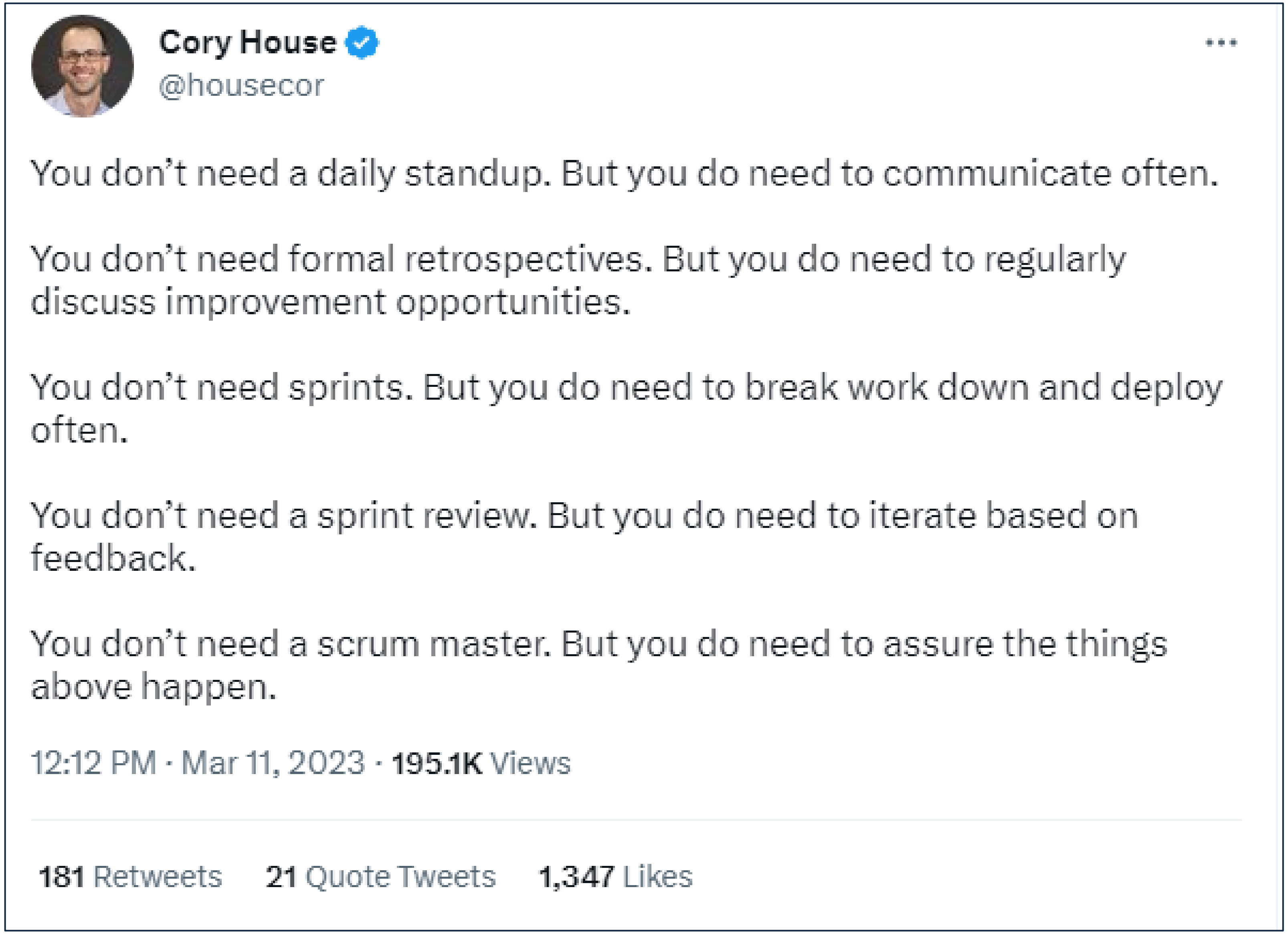
1 linked pull request

Scrum in 1 slide



https://www.scrum.org/resources/scrum-framework-reduce-risk-and-deliver-value-sooner

Interesting comment on the purpose of Scrum events



Cory House
@housecor

You don't need a daily standup. But you do need to communicate often.

You don't need formal retrospectives. But you do need to regularly discuss improvement opportunities.

You don't need sprints. But you do need to break work down and deploy often.

You don't need a sprint review. But you do need to iterate based on feedback.

You don't need a scrum master. But you do need to assure the things above happen.

12:12 PM · Mar 11, 2023 · **195.1K** Views

181 Retweets **21** Quote Tweets **1,347** Likes

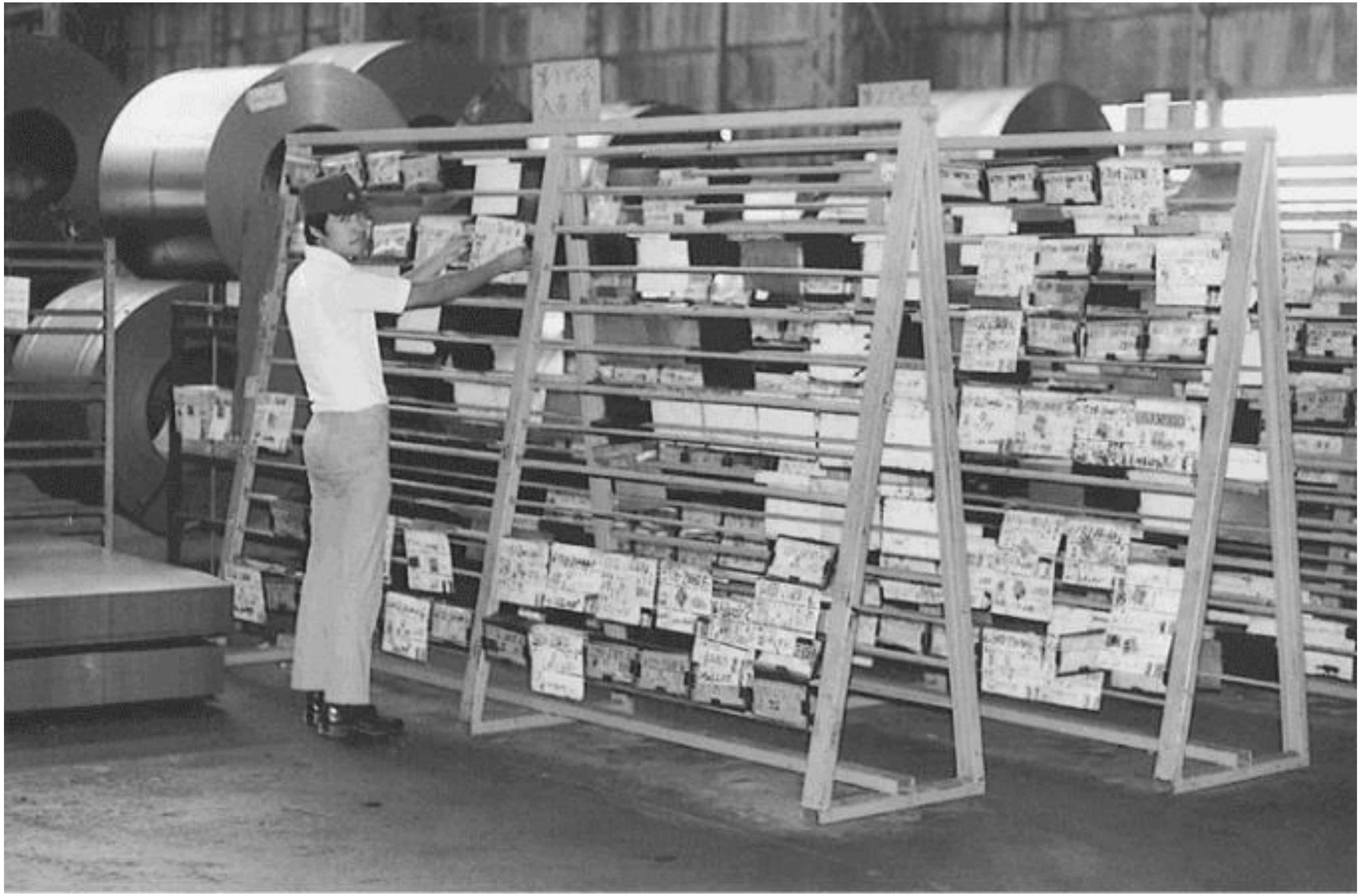
https://twitter.com/i/web/status/1634572956746776577

Kanban

Kanban

- Originated in the 1950s in Japan
- Toyota Production System
- Lean manufacturing, just-in-time production, etc

Kanban = "visual card"



Kanban in Software Development

Kanban was first introduced for software development at Microsoft in 2004 by David Anderson, who noted that it:

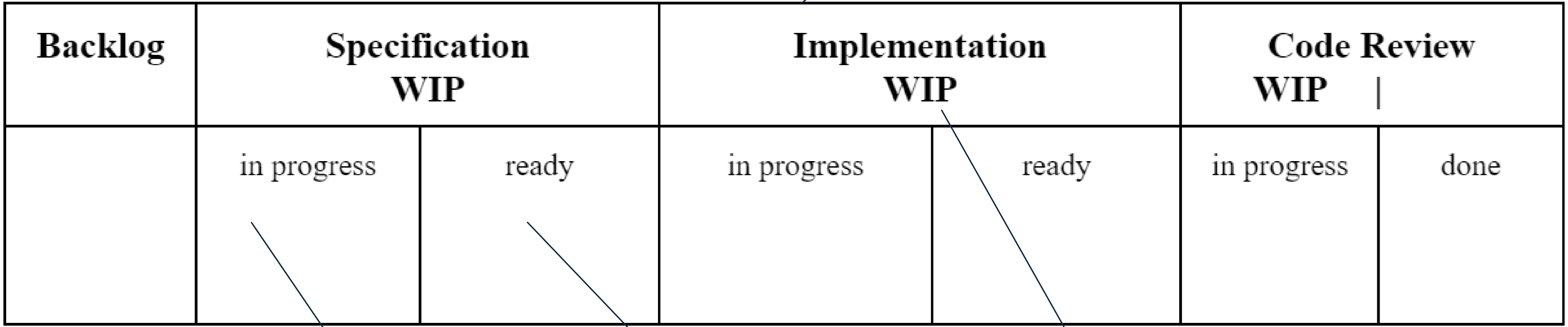
"...promotes a sustainable work pace for development teams by eliminating waste, delivering consistent value, and fostering a culture of continuous improvement."

Kanban vs Scrum

- Kanban is simpler
- No sprints
- It's not mandatory to have roles and events, including:
  - Scrum master
  - Daily Scrum, Retrospectives, Reviews
- Team defines roles and events
- Fewer artifacts
  - Main artifact: **Kanban Board**

Kanban Board

Large columns in the board: kanban steps



Backlog	Specification WIP		Implementation WIP		Code Review WIP	
	in progress	ready	in progress	ready	in progress	done

1st sub-column

2nd sub-column

We will explain shortly

Time

Kanban is a ***Pull*** System

- Members:
  a. Pull a task to work on
  b. Complete the task and move it forward on the board
  c. Go back to step (a)

Final Comments on Kanban vs Scrum

- Simpler than Scrum
  - Recommened for mature teams
  - Perhaps, start with Scrum and then move to Kanban?
  - For this course: Probably best to use Scrum

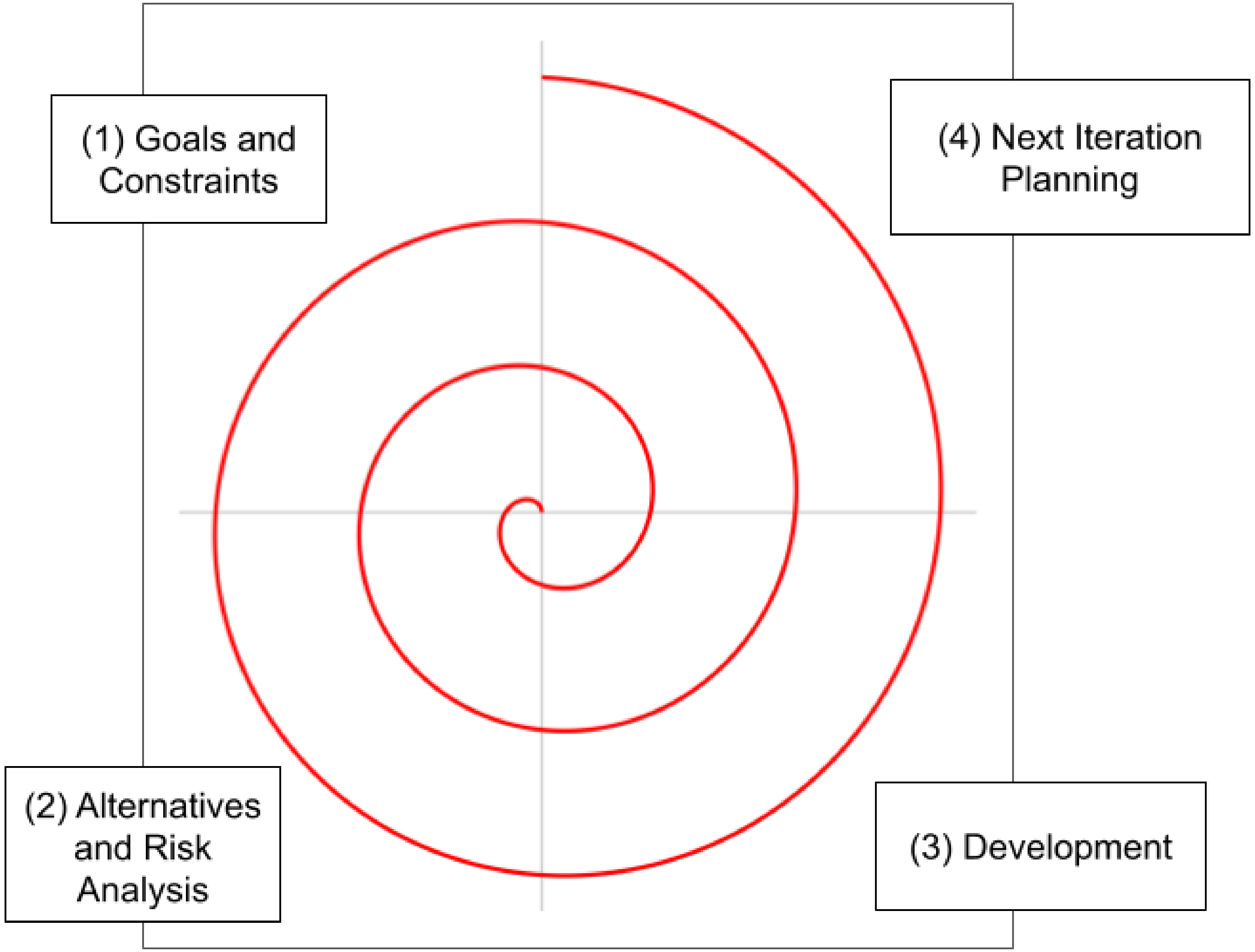
There is a section on the moodle book that compares and contrasts: XP, Scrum, and Kanban

Non-Agile Processes

Iterative Methods

- Transition Waterfall (~1970) to Agile (~2000) was gradual
- Iterative or evolutionary methods were proposed, before the dissemination of agile principles
- Examples:
    - Spiral Method (1986)
    - Rational Unified Process (2003)

Spiral Model

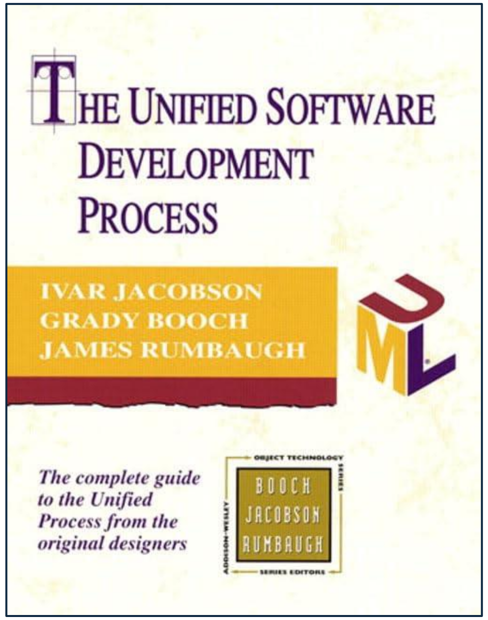

(1) Goals and Constraints
(4) Next Iteration Planning
(2) Alternatives and Risk Analysis
(3) Development

Proposed by Barry Boehm

Iterations: 6 to 24 months (then, longer than in XP or Scrum)

Rational Unified Process (RUP)

- Rational was a company acquired by IBM
- Key characteristic: plan-and-document, using UML and CASE tools

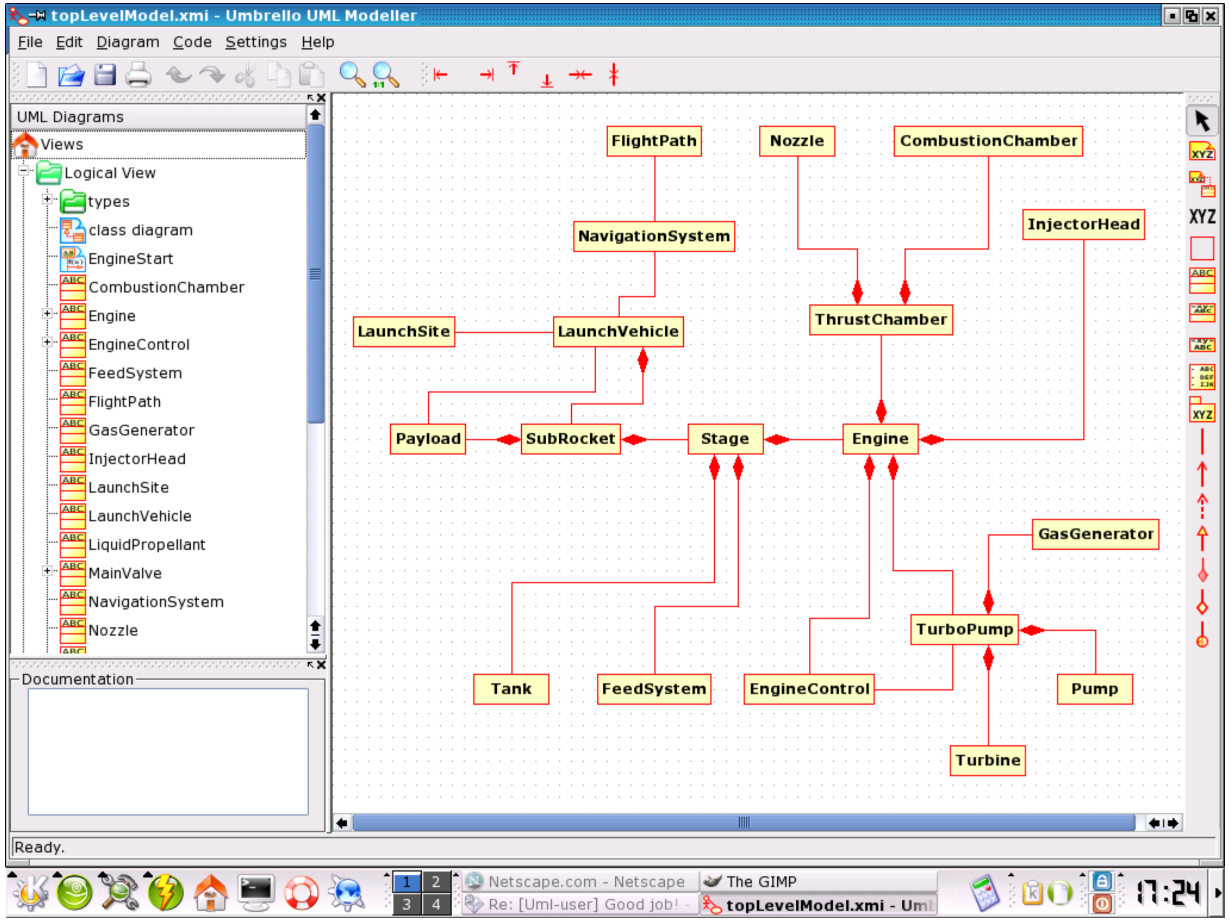

THE UNIFIED SOFTWARE DEVELOPMENT PROCESS

IVAR JACOBSON
GRADY BOOCH
JAMES RUMBAUGH

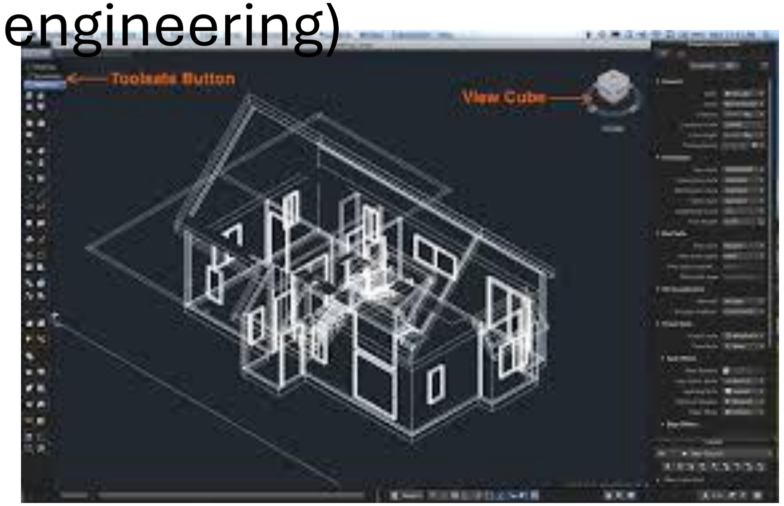
The complete guide to the Unified Process from the original designers

OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

CASE: Computer-Aided Software Engineering



Name comes from CAD systems (used in traditional engineering)



Before concluding

- Processes are not used 100% as in the textbooks
- Treat everthing you read about these methods as a suggestion
- Experimentation is important!

Technical Debt Management in Agile Teams

Agile and Software Development Methodologies

- Software Engineering – Why and What?
- Agile – Brief Review
- Technical Debt Management in Agile Teams
- DevOps, Continuous Integration, Continuous Delivery (CI/CD)

Technical Debt

- Code elements of unsatisfactory quality / “buggy”
- Initially ignored because impact negligible / not noticeable
- Overtime, can grow until the cumulative impact is significant

Technical Debt – Some Definitions

- "… **gap** between the **current state** of a software system and some hypothesized **'ideal'** state in which the system is optimally successful in a particular environment"[1]

- Other examples of such definitions include[2]:
  - "the degree of incompleteness"
  - "a backlog of deferred technical problems" and
  - "any side of the current system that is considered suboptimal from a technical perspective".

[1] Brown, Nanette, et al. "Managing technical debt in software-reliant systems." Proceedings of the FSE/SDP workshop on Future of software engineering research. 2010.

[2] Edith Tom, Aybüke Aurum, Richard Vidgen, An exploration of technical debt, Journal of Systems and Software, Volume 86, Issue 6, 2013.

Technical Debt is not just “Code debt”

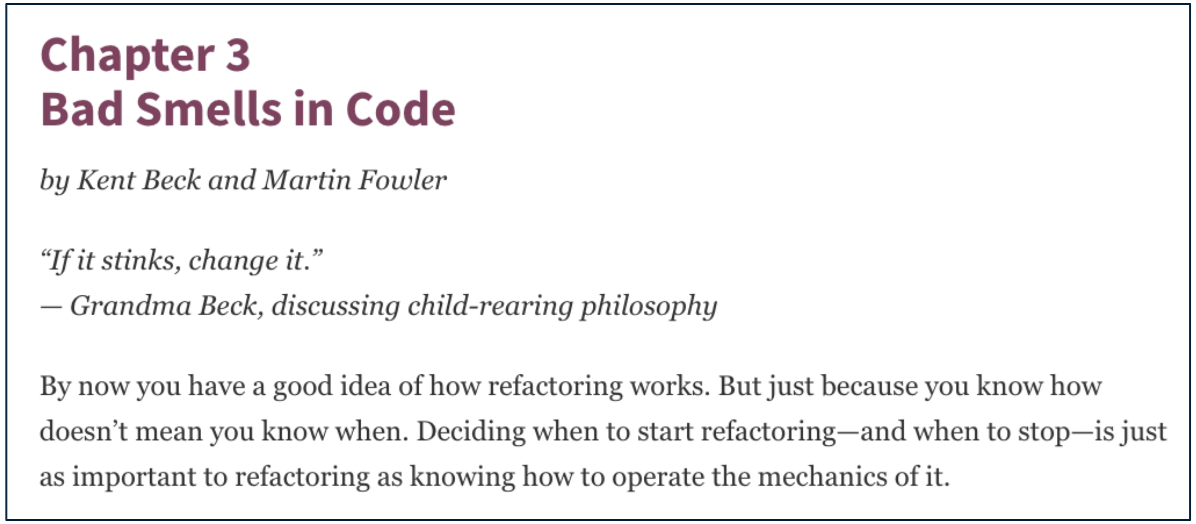
- Code debt
- Design and architectural debt
- Environmental debt
- Knowledge distribution and documentation debt
- Testing debt

Technical debt – Not always bad

- A little debt is not necessarily “bad”
  - In the Agile methodology, it is almost inevitable
  - Can help speed up the development process in the short term
-
- If allowed to accumulate though, can lead to slower development, kill productivity.
  - Agile Methodology due to its nature is specially exposed to this risk
- Global technical debt in 2010 was estimated to be $US 500 Billion.

Code (or Bad) Smells

- Indicators of low-quality code
- Code that is hard to maintain, understand, modify or test
- Therefore, it is a candidate for refactoring



Chapter 3
Bad Smells in Code

by Kent Beck and Martin Fowler

“If it stinks, change it.”
— Grandma Beck, discussing child-rearing philosophy

By now you have a good idea of how refactoring works. But just because you know how doesn’t mean you know when. Deciding when to start refactoring—and when to stop—is just as important to refactoring as knowing how to operate the mechanics of it.

Catalog of Code Smells

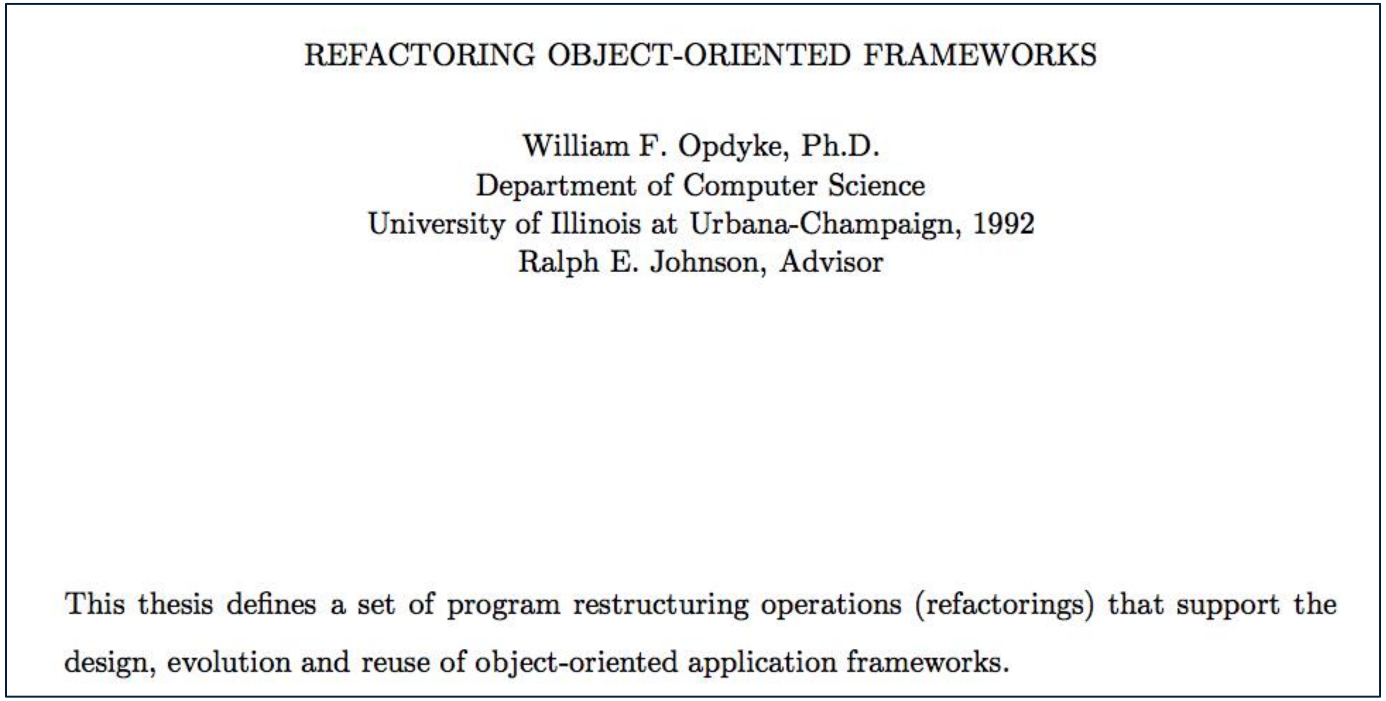
- Duplicated Code
- Long Methods
- Large Classes
- Feature Envy
- Long Parameter List
- Global Variables
- Primitive Obsession
- Mutable Objects
- Data Classes
- Comments

Addressing Technical (Code) Debt

- The main agile tool to address technical debt is: **Refactoring**
- You can however also address this more proactively, by redefining what you mean by "done"
    - E.g. Make automated testing part of the original story / bug fix

Refactoring

- Code transformations that improve maintainability without affecting external behavior



REFACTORING OBJECT-ORIENTED FRAMEWORKS

William F. Opdyke, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1992
Ralph E. Johnson, Advisor

This thesis defines a set of program restructuring operations (refactorings) that support the design, evolution and reuse of object-oriented application frameworks.

Catalog of Refactorings

- Extract Method
- Inline Method
- Move Method
- Extract Class
- Renaming
- etc

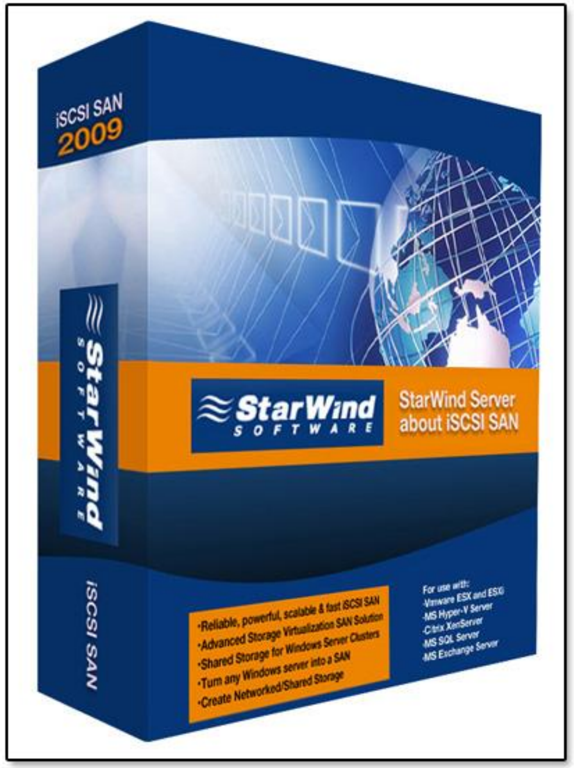
DevOps,
Continuous Integration / Continuous Delivery (CI/CD)

Agile and Software Development Methodologies

- Software Engineering – Why and What?
- Agile – Brief Review
- Technical Debt Management in Agile Teams
- DevOps, Continuous Integration, Continuous Delivery (CI/CD)

This is about the last mile…

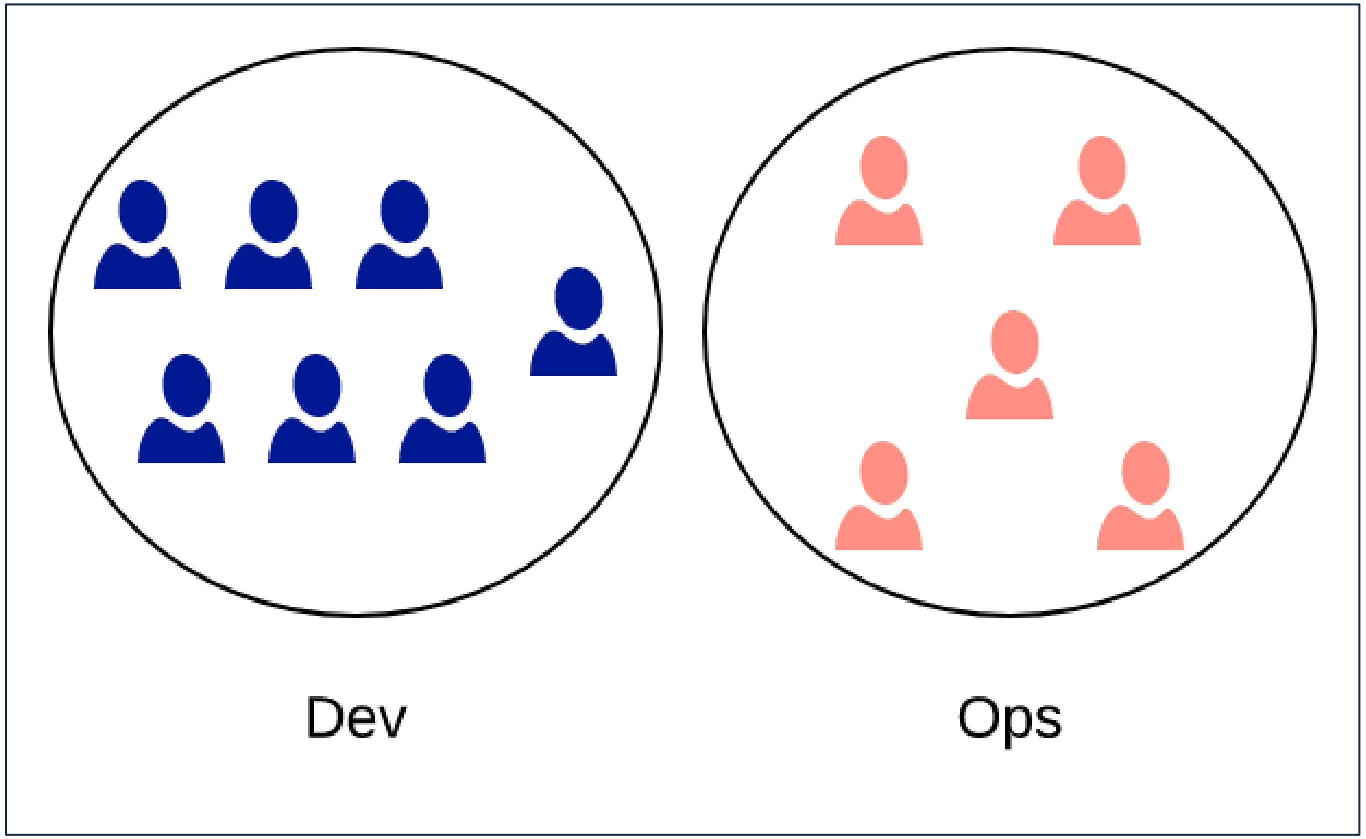
… putting the code into production



iSCSI SAN
2009
StarWind SOFTWARE
StarWind SOFTWARE
StarWind Server about iSCSI SAN
iSCSI SAN
·Reliable, powerful, scalable & fast iSCSI SAN
·Advanced Storage Virtualization SAN Solution
·Shared Storage for Windows Server Clusters
·Turn any Windows server into a SAN
·Create Networked/Shared Storage
For use with:
-Vmware ESX and ESXi
-MS Hyper-V Server
-Citrix XenServer
-MS SQL Server
-MS Exchange Server

This Photo by Unknown Author is licensed under CC BY-NC-ND

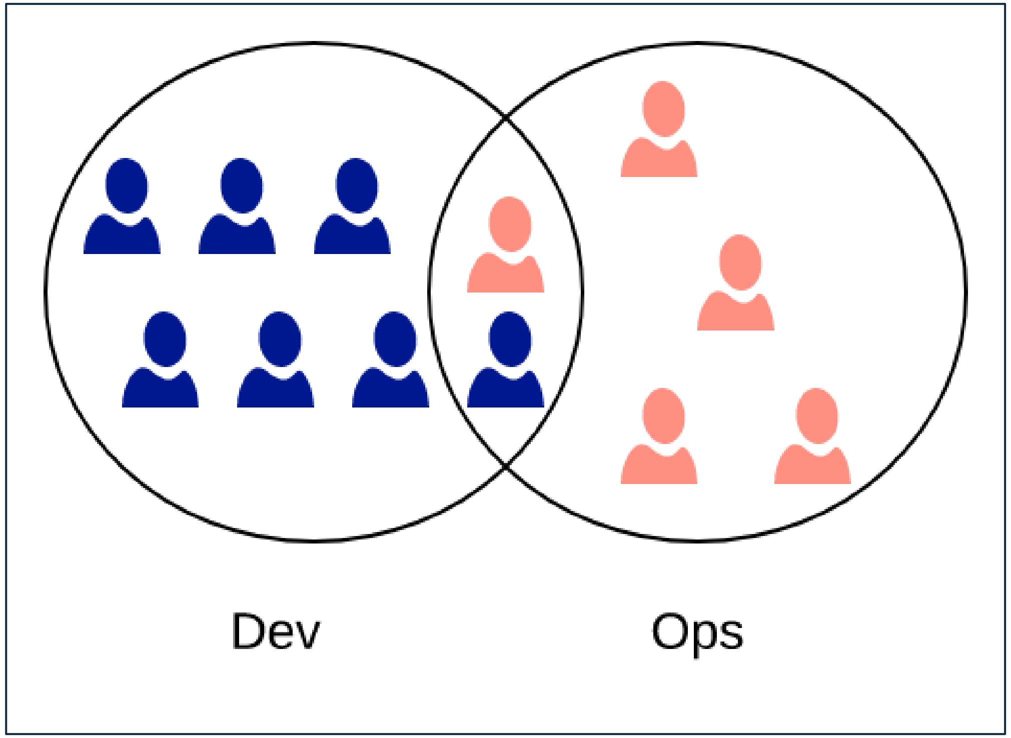
In the past, deployment was a challenging and high-risk process

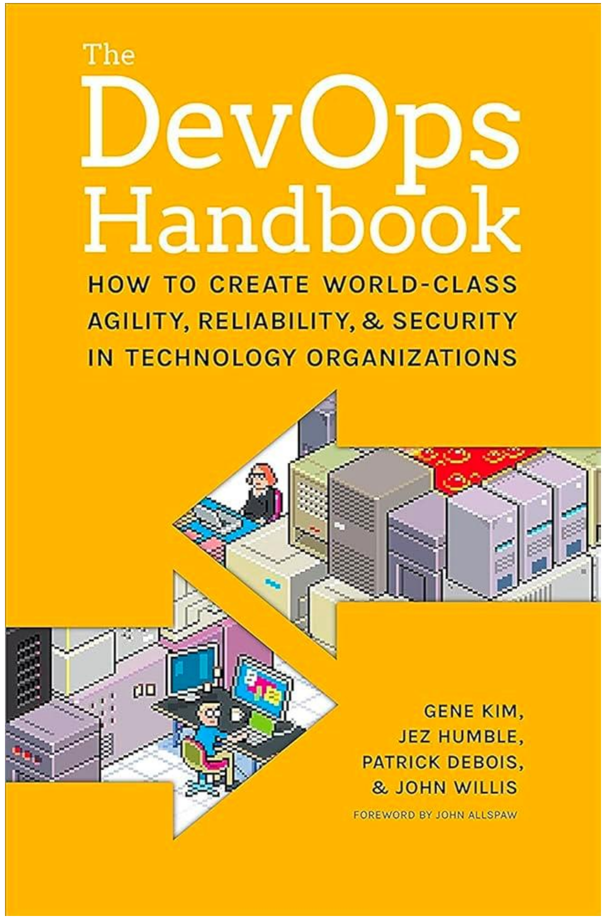


Two independent silos, with very little communication

Ops = system administrators, support, sysadmin, IT personnel, etc

Central idea of DevOps: Bridging the gap between Dev and Ops




The DevOps Handbook
HOW TO CREATE WORLD-CLASS AGILITY, RELIABILITY, & SECURITY IN TECHNOLOGY ORGANIZATIONS
GENE KIM, JEZ HUMBLE, PATRICK DEBOIS, & JOHN WILLIS
FOREWORD BY JOHN ALLSPAW

"Imagine a world where product owners, development, QA, IT Operations, and Infosec work together, not just to aid each other, but to guarantee the overall success of the organization."

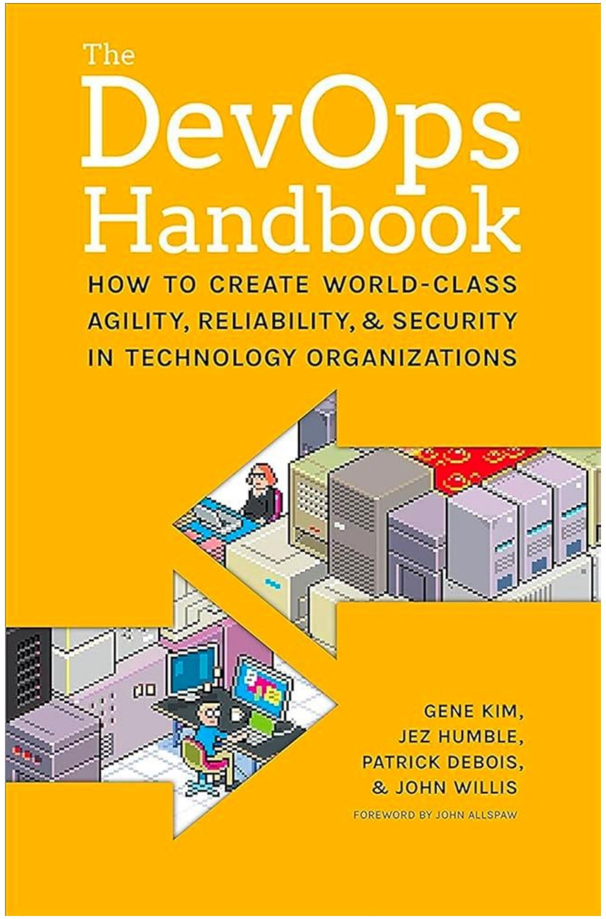
Objective: eliminate the "blame culture"

Dev: "The problem is not in my code, but in your server"

Ops: "The problem is not in my server, but in your code"

DevOps Principles

- Foster collaboration between Devs and Ops teams
- Apply an agile mindset throughout the deployment phase
- Transform deployments into a routine operation
- Deploy software every day
- Automate the deployment process



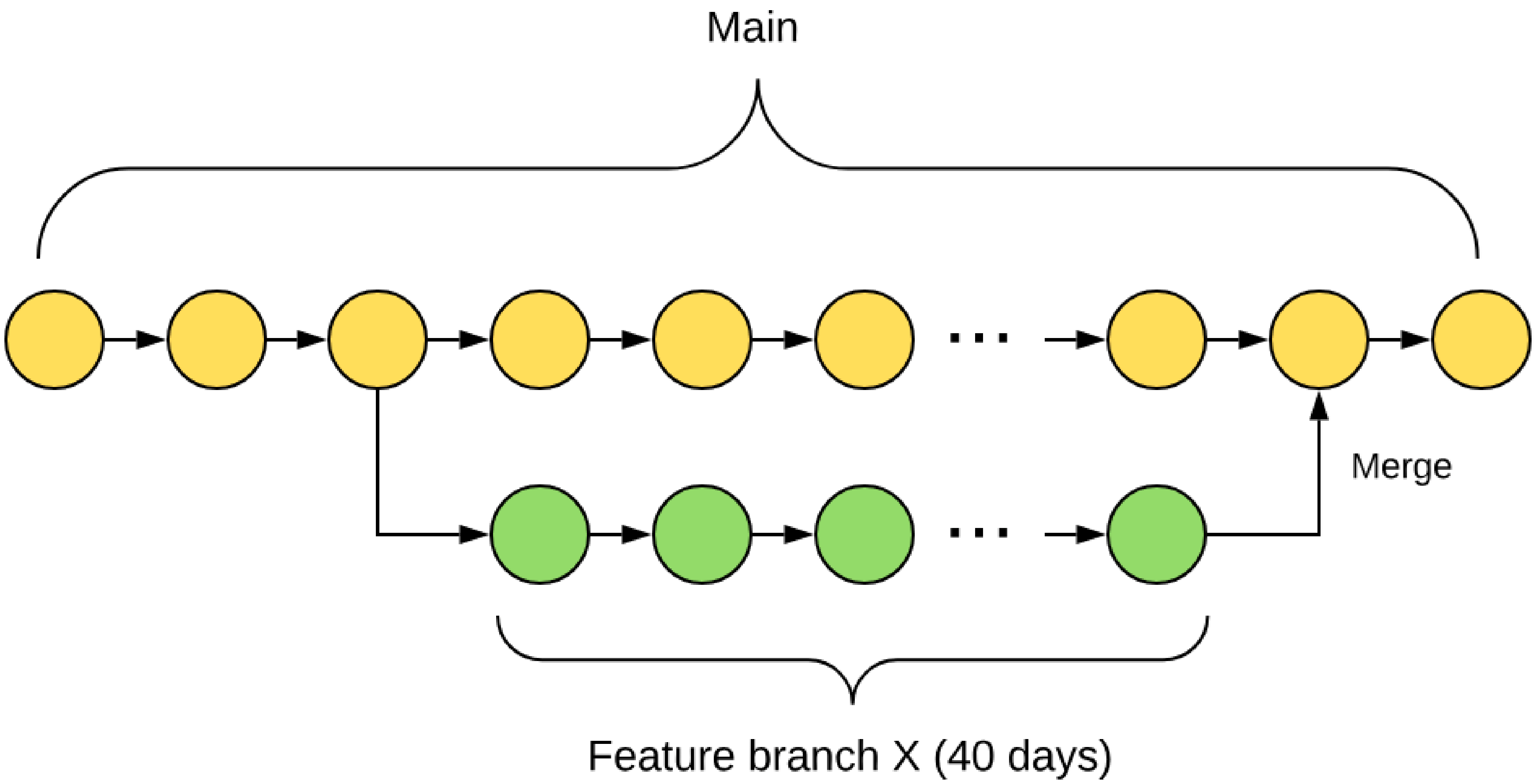
"Instead of starting deployments at midnight on Friday and spending the weekend working to complete them, deployments occur on any business day when everyone is in the company and without customers noticing—except when they encounter new features and bug fixes."

DevOps Practices

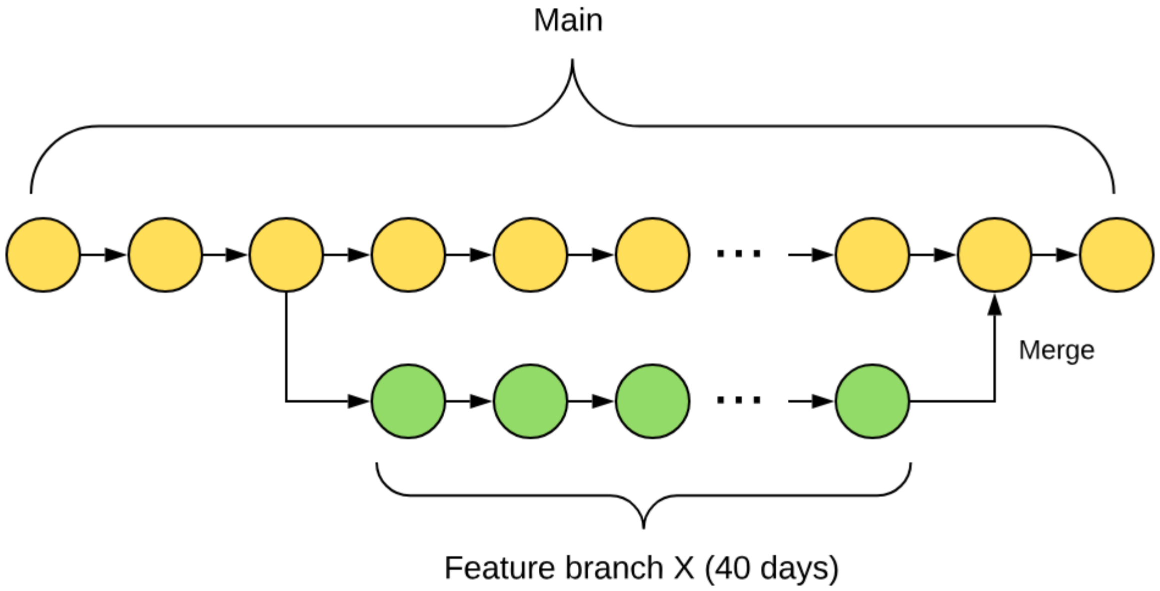
- Version Control
- Continuous Integration
- Branching Strategies
- Continuous Deployment
- Feature Flags

Continuous Integration

In the past: feature branches were very common



Result after 40 days: **merge hell**



If a task causes pain, it's best not to let it accumulate; instead, tackle it daily

Continuous Integration (CI)

- First introduced in XP
- CI emphasizes frequent code integration into the main branch
- How often? Most authors recommend at least daily

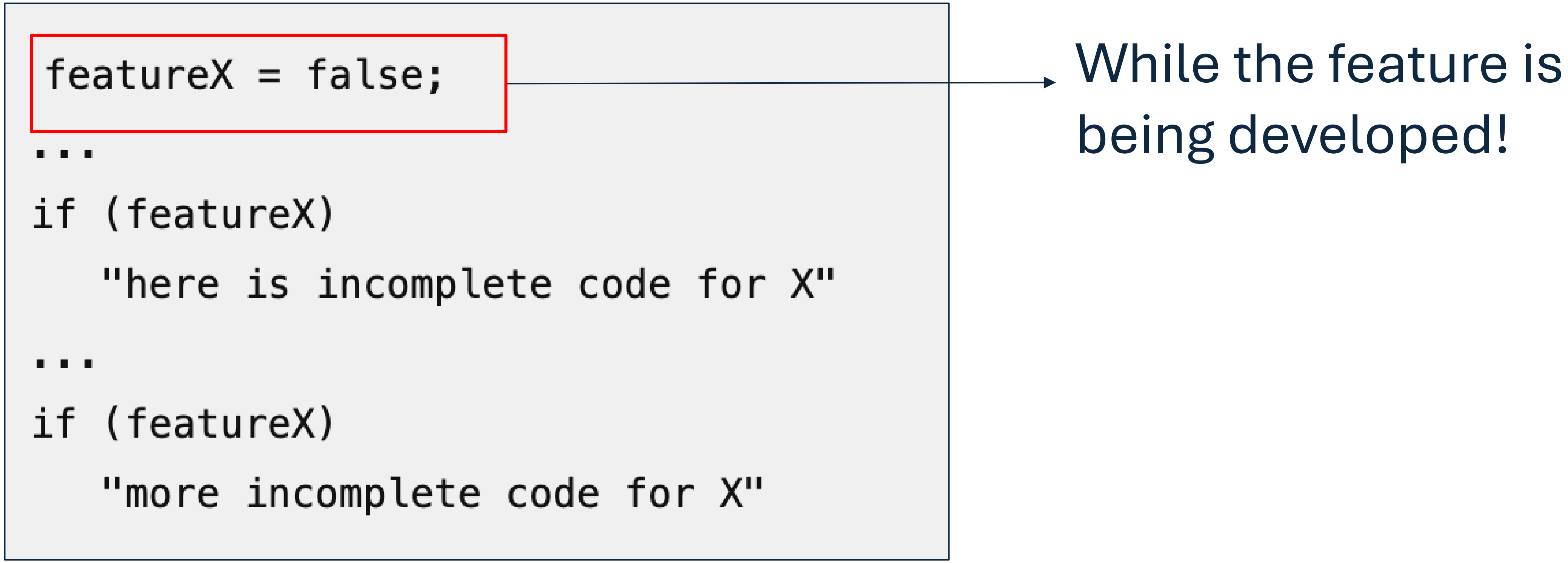
Continuous Deployment

Continuous Deployment (CD)

- CI: integrate code frequently
- CD: integrated code goes immediately into production
- Goal: rapid experimentation and feedback!

How to keep partial implementations from reaching customers?

Feature Flags (also called feature toggles)



```
 featureX = false;
...
if (featureX)
   "here is incomplete code for X"
...
if (featureX)
   "more incomplete code for X"
```

While the feature is being developed!

In Summary

- Agile methodology was a response to software engineering projects taking too long and costing too much – It is now the dominant software design methodology

- An iterative, flexible approach that prioritizes:
  - collaboration
  - customer feedback
  - continuous improvement
  - product (over documentation)

- It is an abstract idea that can be realized by various methods, e.g. XP, Scrum,. Kanban
  - Scrum is the most structured approach from these three, most popular, and the one we use by default in this course

- Technical debt is an existential risk that must be deliberately addressed

- DevOps, CI and CD are closely related approaches