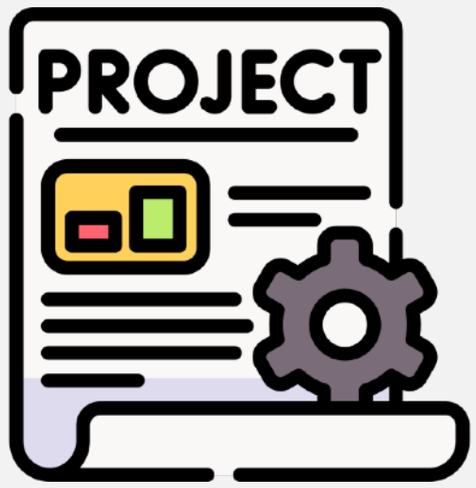
CEN 5016: Software Engineering

Fall 2025

University of Central Florida

Dr. Kevin Moran

Week 3 - Class 1: Project Planning & Agile Development



Administrivia

- Let me know if you are not on Ed Discussions
- Team-forming this week - *Due Thurs, Sept 11th*
  - Teams of 3 students
  - See Ed Discussions Post
- Assignment 1 Graded by tomorrow
- Assignment 2 out tomorrow

Software Measurement & Metrics

Simplistic Productivity Measures

- Lines of code per day?
  - Industry average 10-50 lines/day
  - Debugging + rework ca. 50% of time

- • Function/object/application points per month • Bugs fixed?
- Milestones reached?

Incentivizing Productivity

- What happens when developer bonuses are based on
  - Lines of code per day?
  - Amount of documentation written?
  - Low number of reported bugs in their code?
  - Low number of open bugs in their code?
  - High number of fixed bugs?
  - Accuracy of time estimates?

Developer Productivity Myths

- Productivity is all about developer activity
- Productivity is only about individual performance
- One productivity metric can tell us everything
- Productivity measures are useful only for managers
- Productivity is only about engineering systems and developer tools

WARNING!!

- Most software metrics are controversial
  - Usually only plausibility arguments, rarely rigorously validated
  - Cyclomatic complexity was repeatedly refuted, yet is still used
  - “Similar to the attempt of measuring the intelligence of a person in terms of the weight or circumference of the brain”
- Use carefully!
- Code size dominates many metrics
- Avoid claims about human factors (e.g., readability) and quality, unless validated
- Calibrate metrics in project history and other projects
- Metrics can be gamed; you get what you measure

Summary

- Measurement is difficult but important for decision making
- Software metrics are easy to measure but hard to interpret,
  validity often not established
- Many metrics exist, often composed; pick or design suitable metrics if needed
- Careful in use: monitoring vs incentives
- Strategies beyond metrics

Questions to Consider for Your Projects

- What properties do we care about and how do we measure them?
- What is being measured? Does it (to what degree) capture the thing you care about? What are its limitations?
- How should it be incorporated into process?
- What are potentially negative side effects or incentives?

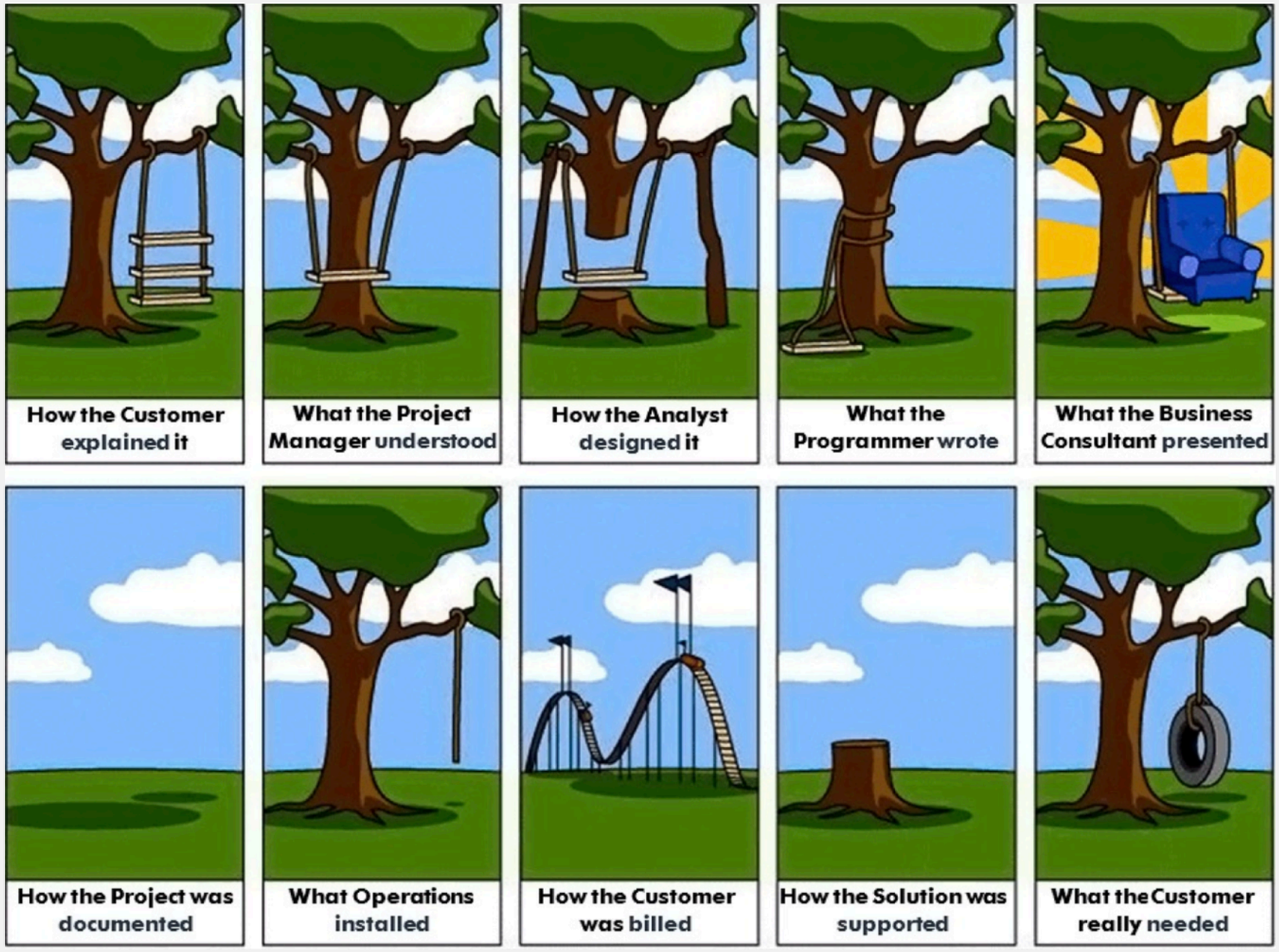
Project Planning & Agile Development

Learning Goals

- Recognize the importance of project planning
- Understand the difficulty of measuring progress
- Identify why software development has project characteristics
- Use milestones for planning and progress measurement
- Understand backlogs and user stories
- Get to know your team!

Software Process

- "The set of activities and associated results that produce a software product."



All Software Development Processes

Discuss the software that needs to be written

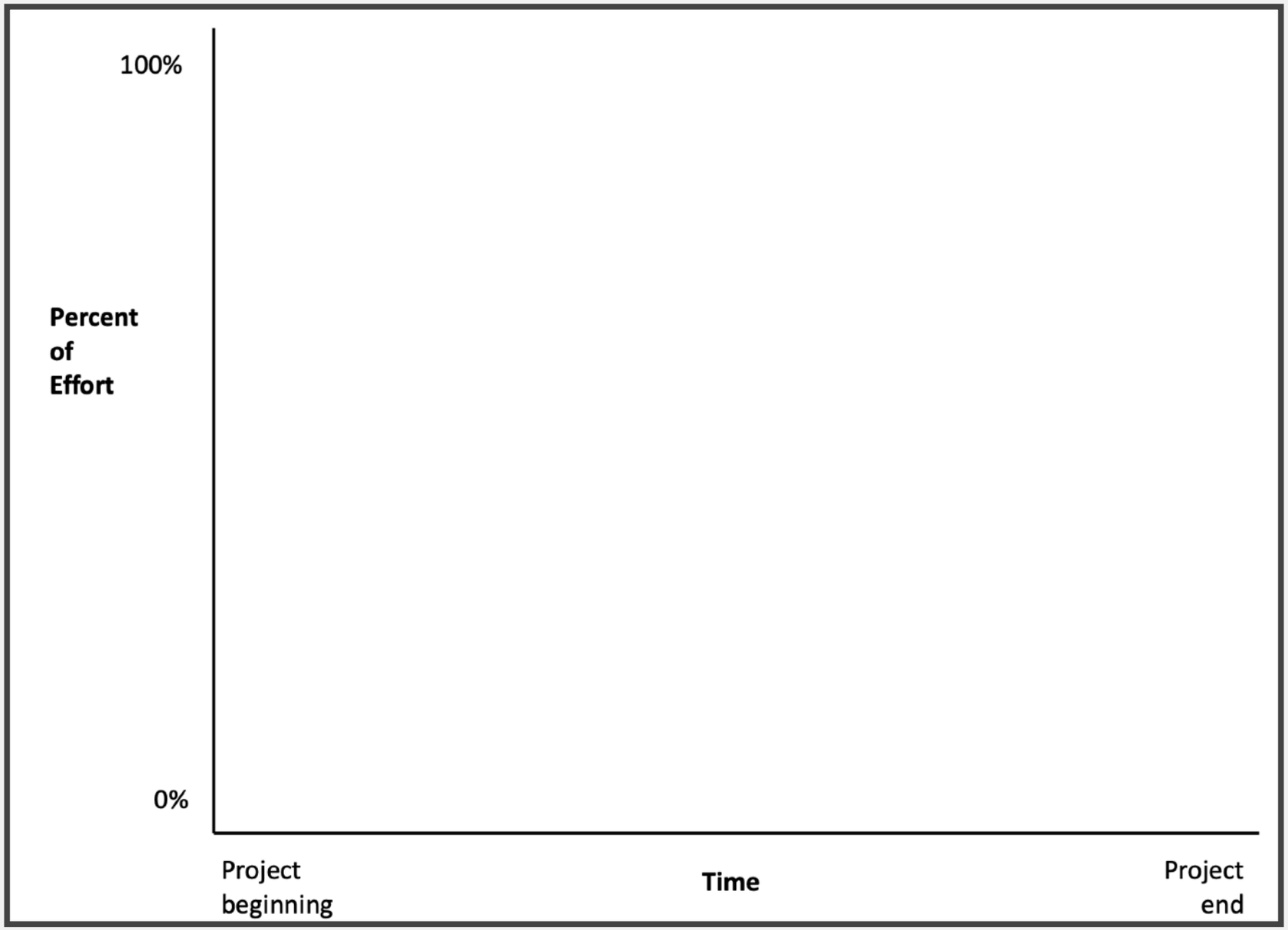
Write some code

Test the code to identify the defects

Debug to find causes of defects

Fix the defects

Effort Spent During the Process



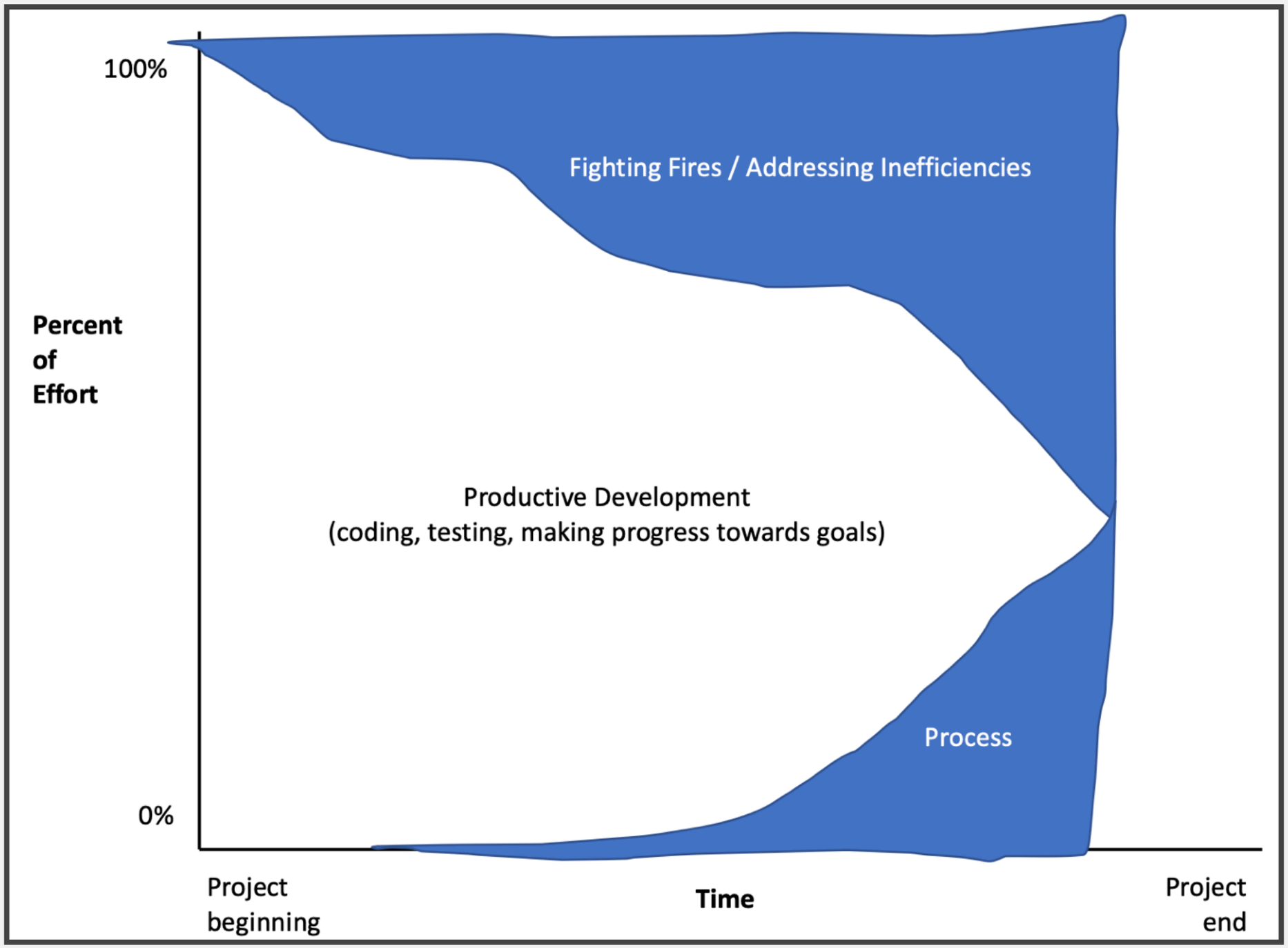
Effort Spent During the Process

Let's Improve the Reliability of this Process

- Write down all requirements
  - Review requirements
  - Require approval for all changes to requirements
- Use version control for all changes
  - Review code
- Track all work items
  - Break down feature development into small tasks
  - Write down and monitor all reported bugs
- Hold regular, frequent status meetings
  - Plan and conduct quality assurance
  - Employ a DevOps framework to push code between developers and operations

Effort Spent During the Process

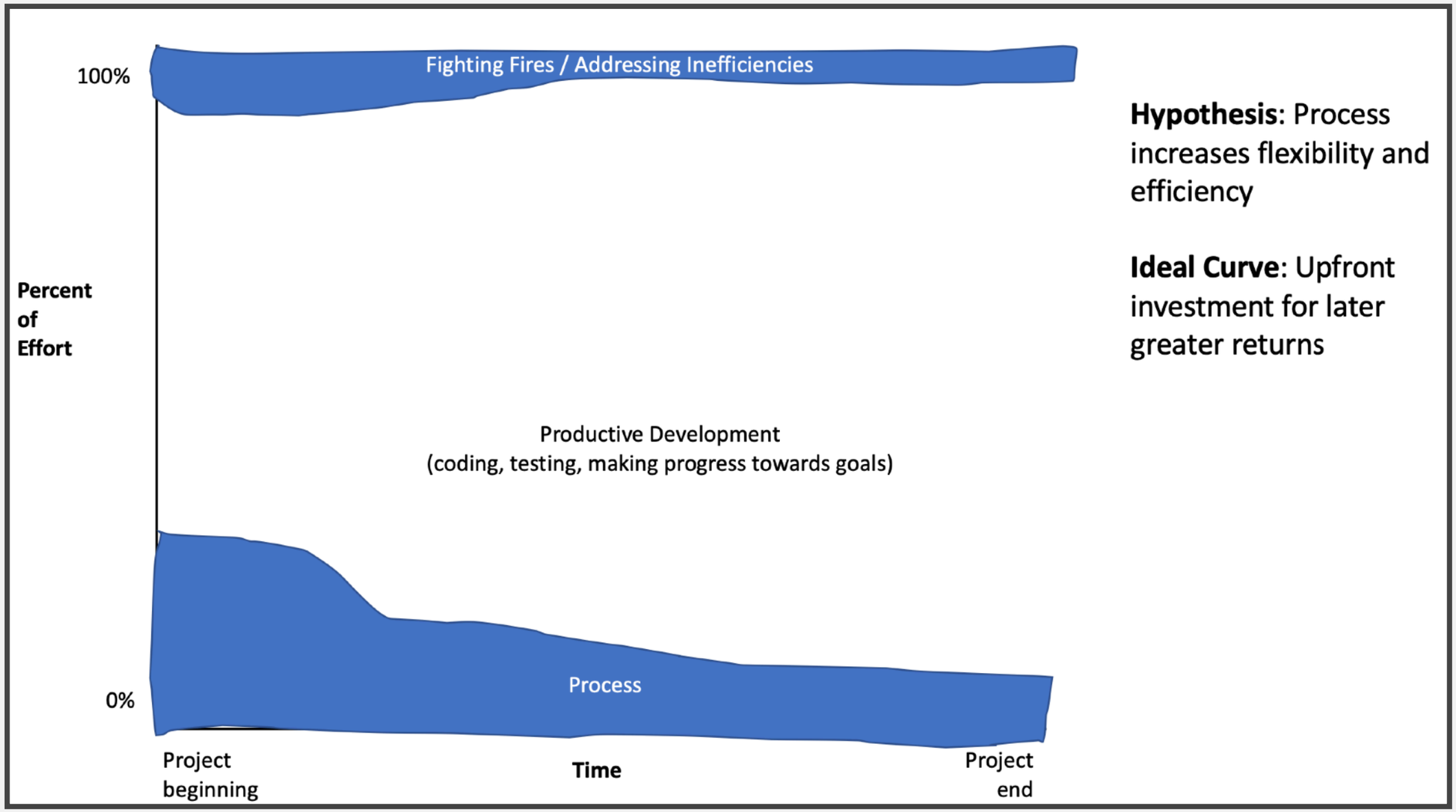
Effort Spent During the Process



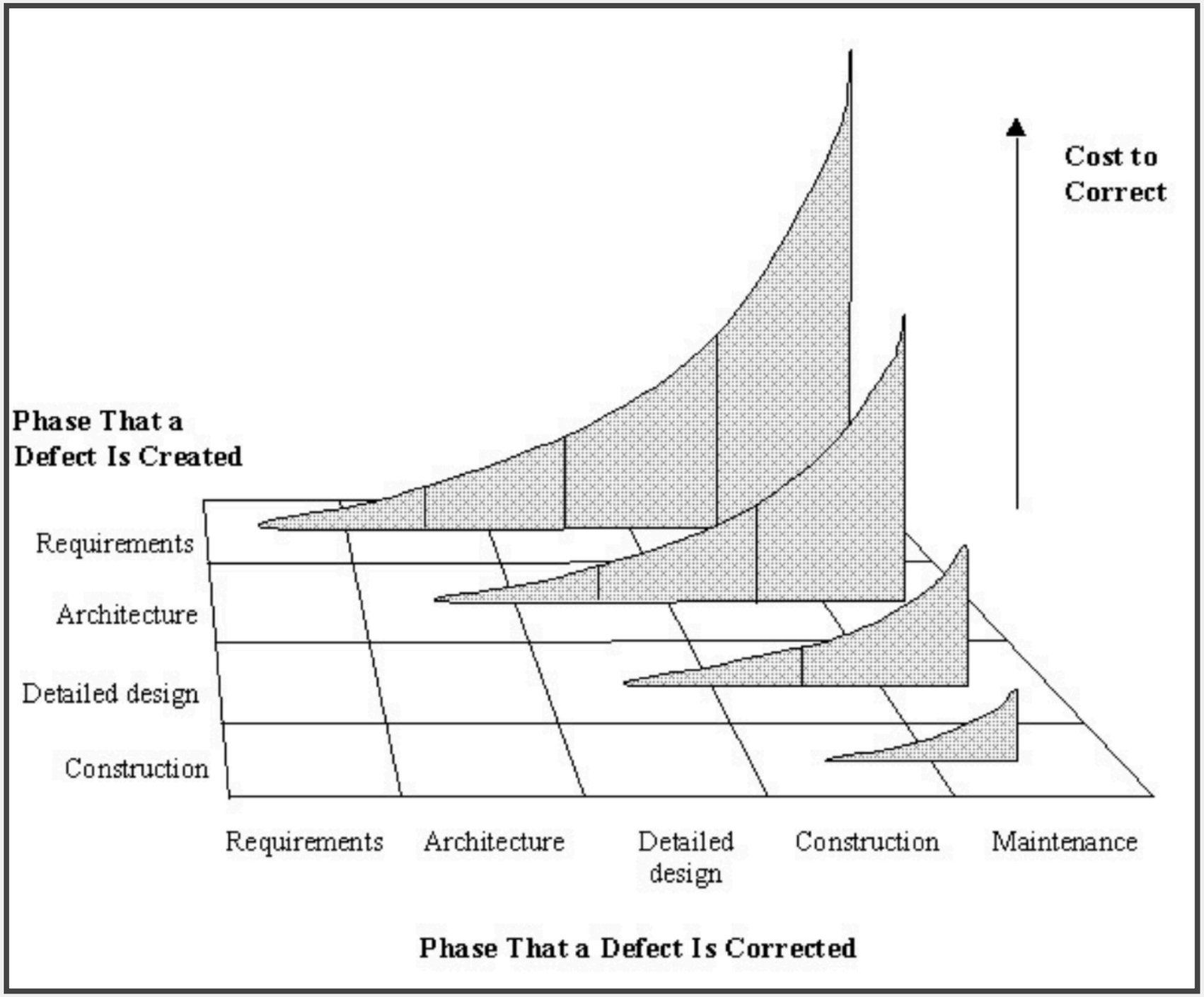
Example Process Issues

- **Change Control:** Mid-project informal agreement to changes suggested by customer. Project scope expands 25-50%
- **Quality Assurance:** Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features. Release with known defects.
- **Defect Tracking:** Bug reports collected informally. Bugs are overlooked.
- **System Integration:** Integration of independently developed components at the very end of the project. Interfaces out of sync.
- **Source Code Control:** Accidentally overwrote changes. Lost work.
- **Scheduling:** Late project. Developers asked to re-estimate work effort weekly.

Effort Spent During the Process

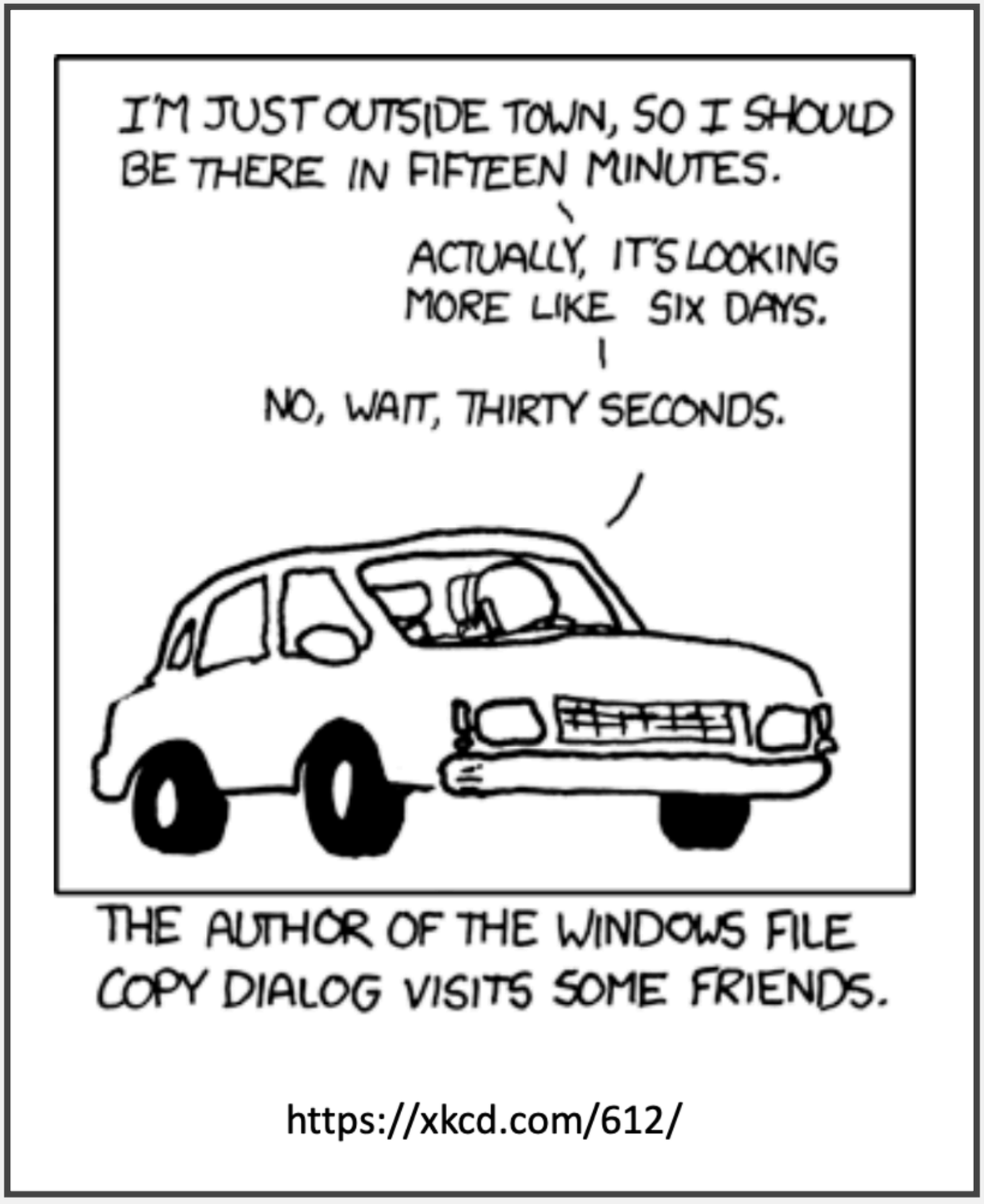


Defect Correction Effort



Planning

Time Estimation



https://xkcd.com/612/

Activity: Estimate Time

- **Task A:** Web version of the Monopoly board game with Orlando street names
  - **Team:** just you

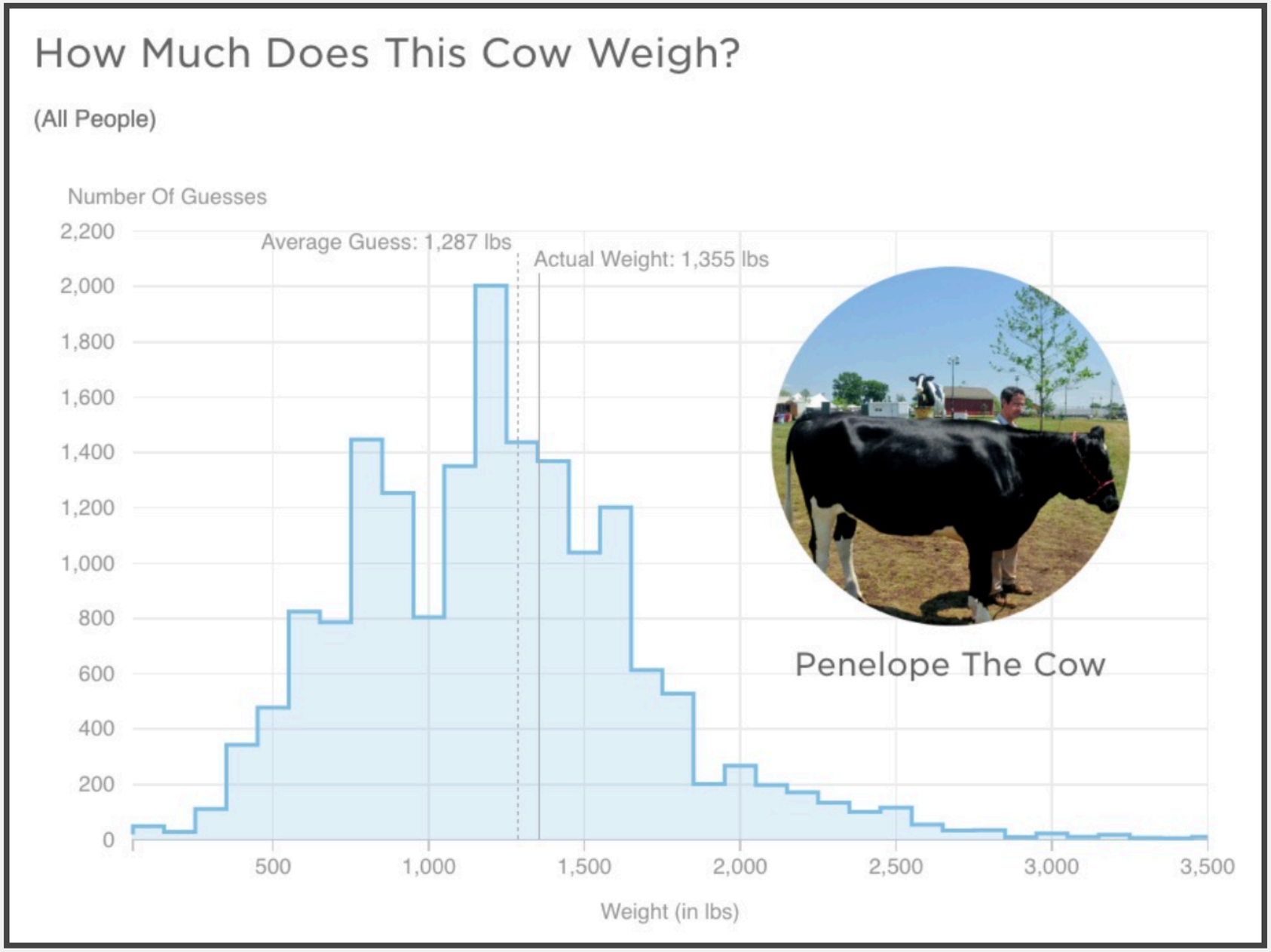
- **Task B:** Bank smartphone app
  - **Team:** you with team of 4 developers, one experienced with iPhone apps, one with background in security

- **Estimate:** 8h days, 20 workdays in a month, 220 workdays per year

Revise Time Estimate

- Do you have comparable experience to base an estimate on?
- How much design do you need for each task?
- How much testing time do you need for each task?
- Let’s break down the task into ~5 smaller tasks and estimate
  their lengths.
- Revise our overall estimate, if necessary

Wisdom of the Crowd



Measuring Progress

- “I’m almost done with the app. The frontend is almost fully implemented. The backend is fully finished except for the one stupid bug that keeps crashing the server. I only need to find the one stupid bug, but that can probably be done in an afternoon. We should be ready to release next week.”

Measuring Progress

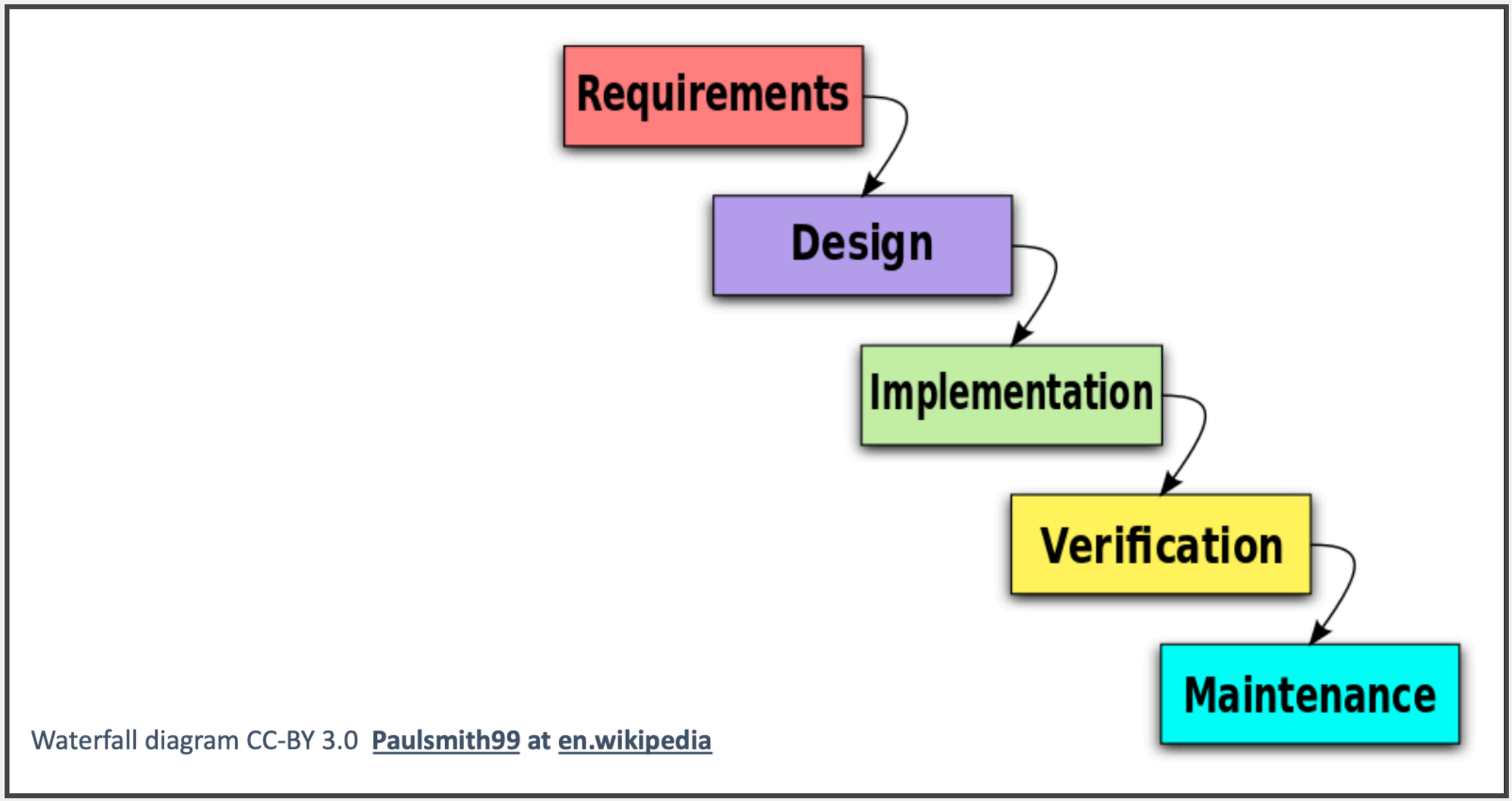
- Developer judgment: x% done
- Lines of code?
- Functionality?
- Quality?

Milestones and Deliverables Make Progress Observable

- **Milestone:** clear end point of a (sub)tasks
  - For project manager
  - Reports, prototypes, completed subprojects
    - “80% done“ is not a suitable mile stone
- **Deliverable:** Result for customer
  - Similar to a milestone, but for customers
  - Reports, prototypes, completed subsystems

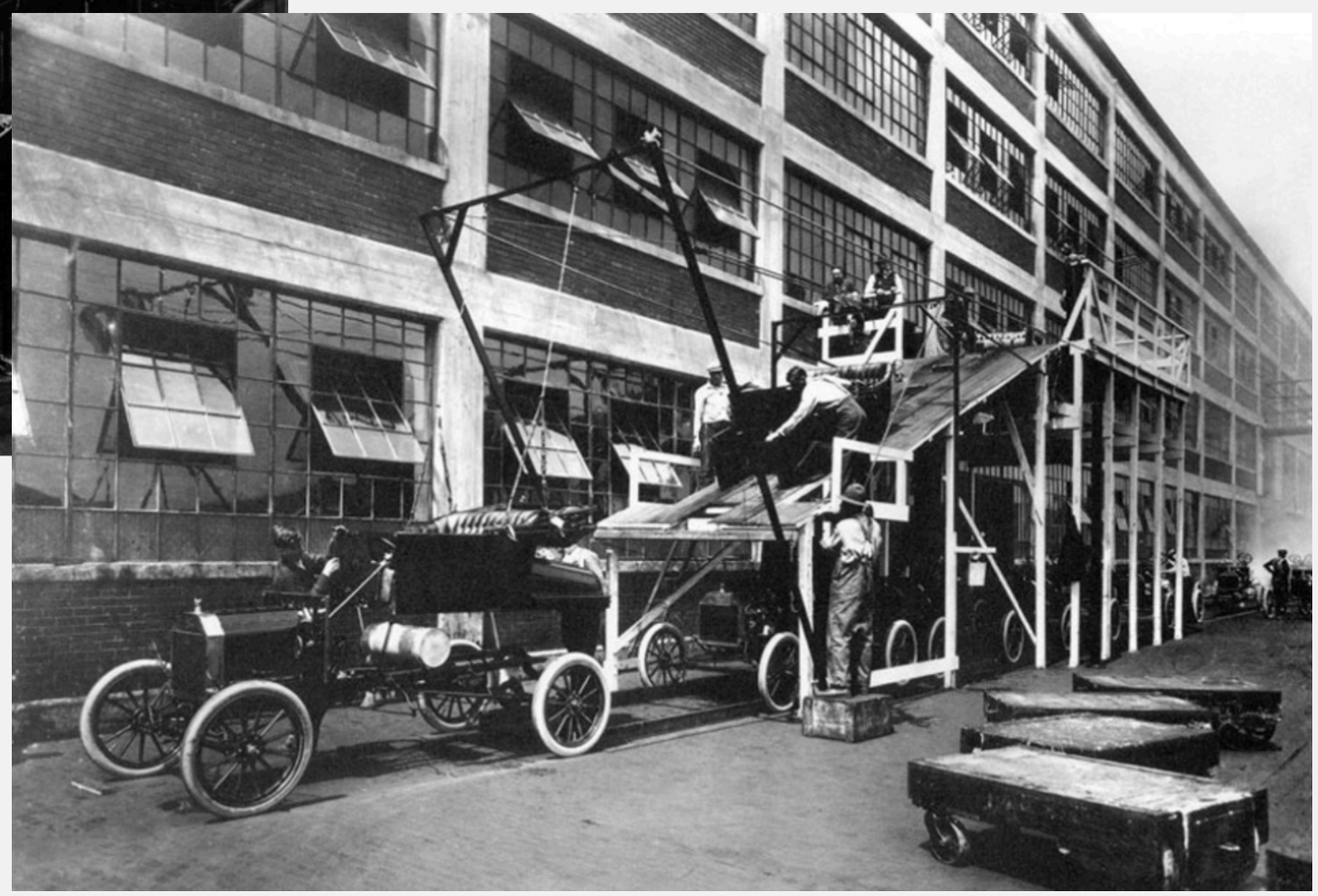
Processes

Waterfall was the OG Software Process



Waterfall diagram CC-BY 3.0 **Paulsmith99** at **en.wikipedia**

Akin to Processes Pioneered in Auto Manufacturing by Ford

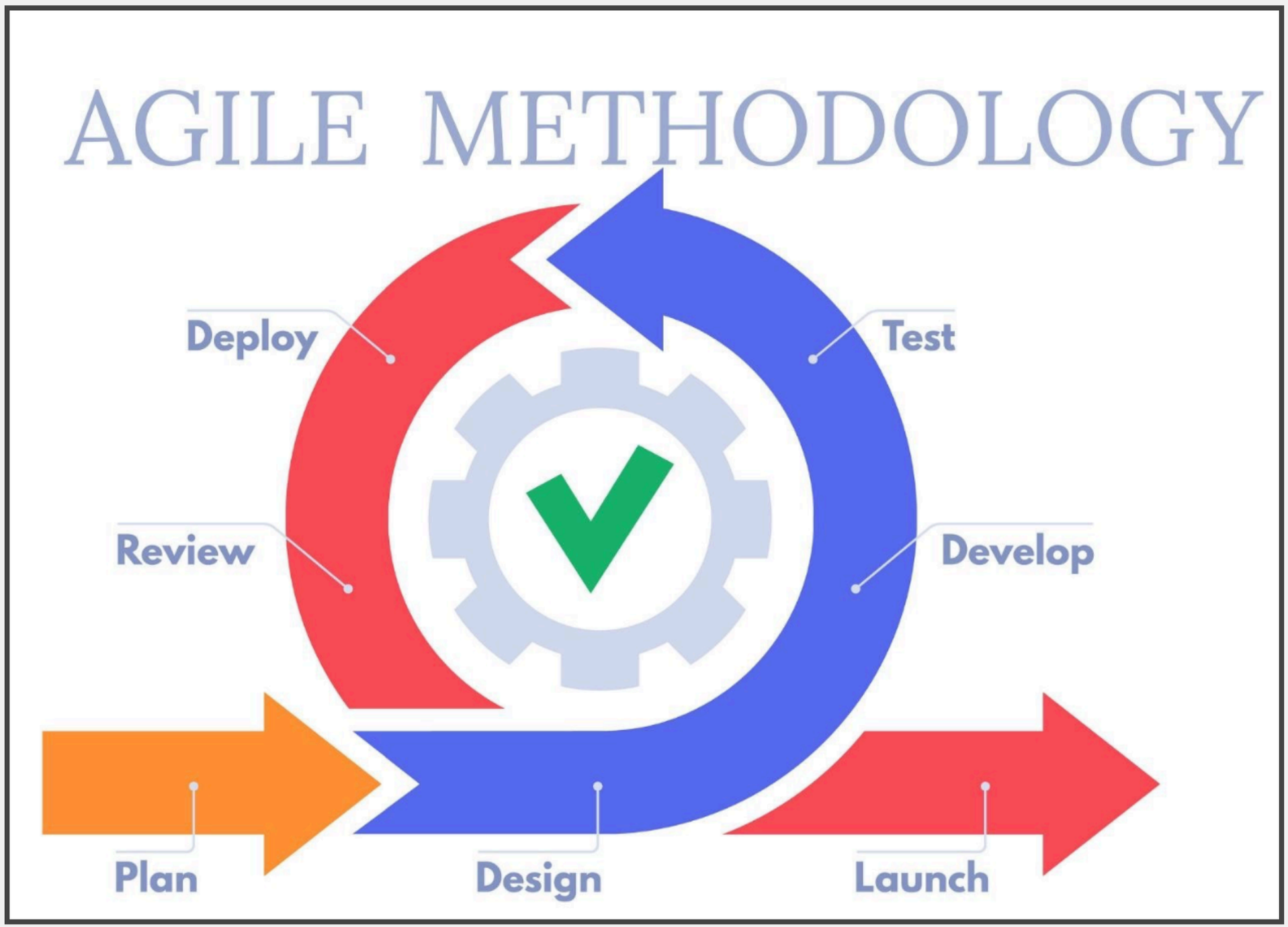


LEAN Production Adapts to Variable Demand

Taiichi Ohno

- Toyota Production System (TPS)
  - Build only what is needed, only when it is needed.
  - Use the “pull” system to avoid overproduction (Kanban)
  - Stop to fix problems, to get quality right from the start (Jidoka)
    - Workers are multi-skilled and understand the whole process; take ownership
- Lots of recent software buzzwords build on these ideas
  - Just-in-time, DevOps, Shift-Left

Now, Most Teams use some form of Agile Methods



Now, Most Teams use some form of Agile Methods

Agile software development

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

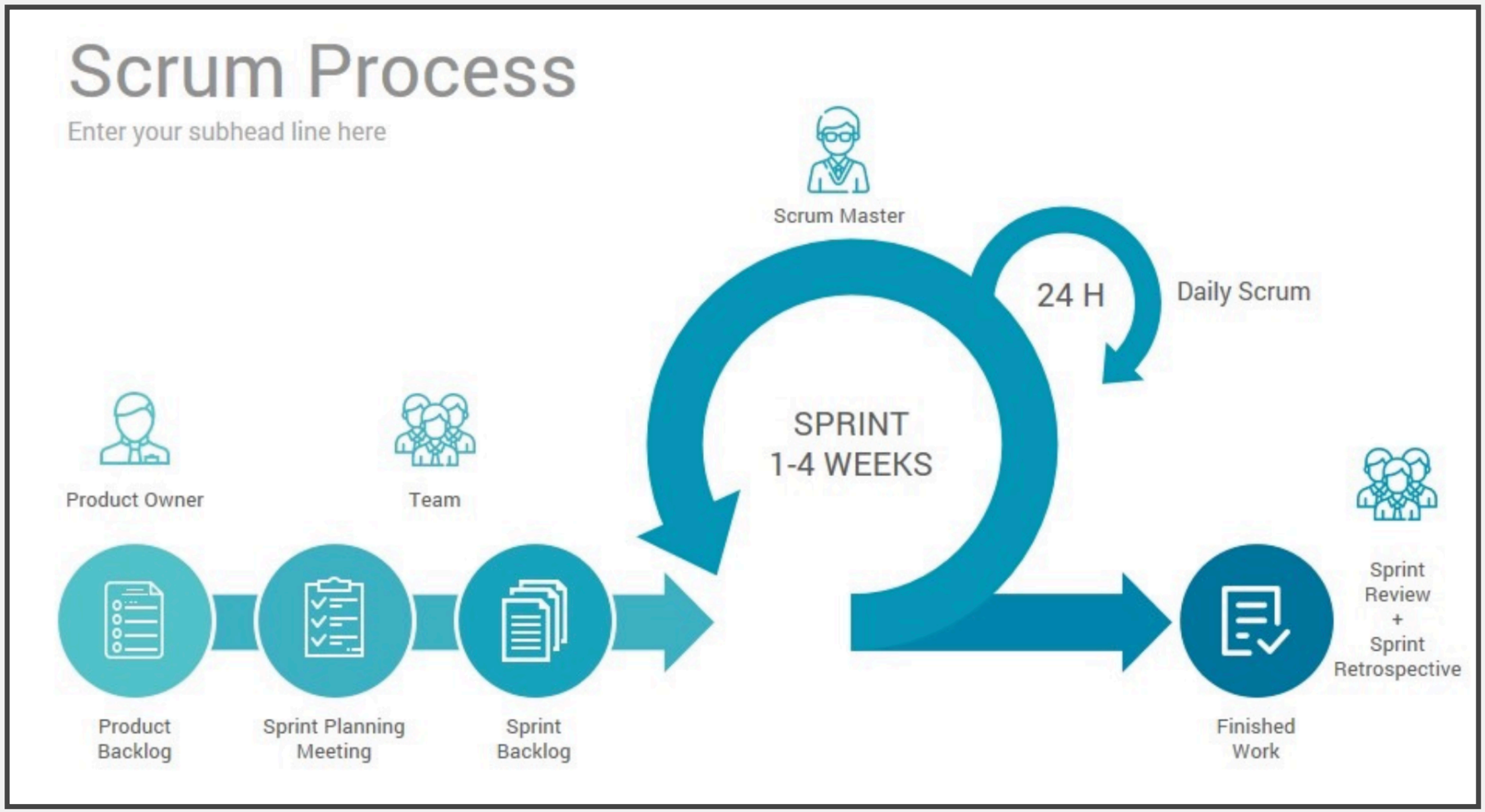
Manifesto for Agile Software Development (2001)

Core Concepts in Agile

Core concepts	Facets of agility in the literature
(1) Incremental design and iterative development	*Anticipating change by working iteratively – in short, delivery cycles – and thereby reducing the scope of the product to small increments to create opportunities for inspection; Creating change through incremental software design in response to change from what has been learned*
(2) Inspect and adapt cycles	*Anticipating change by instituting ceremonies for inspecting and adapting (i.e., learning from and creating change in response to discovered changes) the product increment (e.g., simplifying – "just enough" – design, testing software frequently) and the development process (e.g., updating work statuses, reevaluating team processes, reprioritizing requirements)*
(3) Working cooperatively/ Collaboratively/In close communication	*Anticipating change through recognising and predicting changes in one's environment; Creating change as a team by working together to respond to change from what has been learned collectively*
(4) Continuous customer involvement	*In addition to the cell above, centralising user requirements changes by working together with the customer to collectively identify and respond to change early through close customer involvement*

Scrum

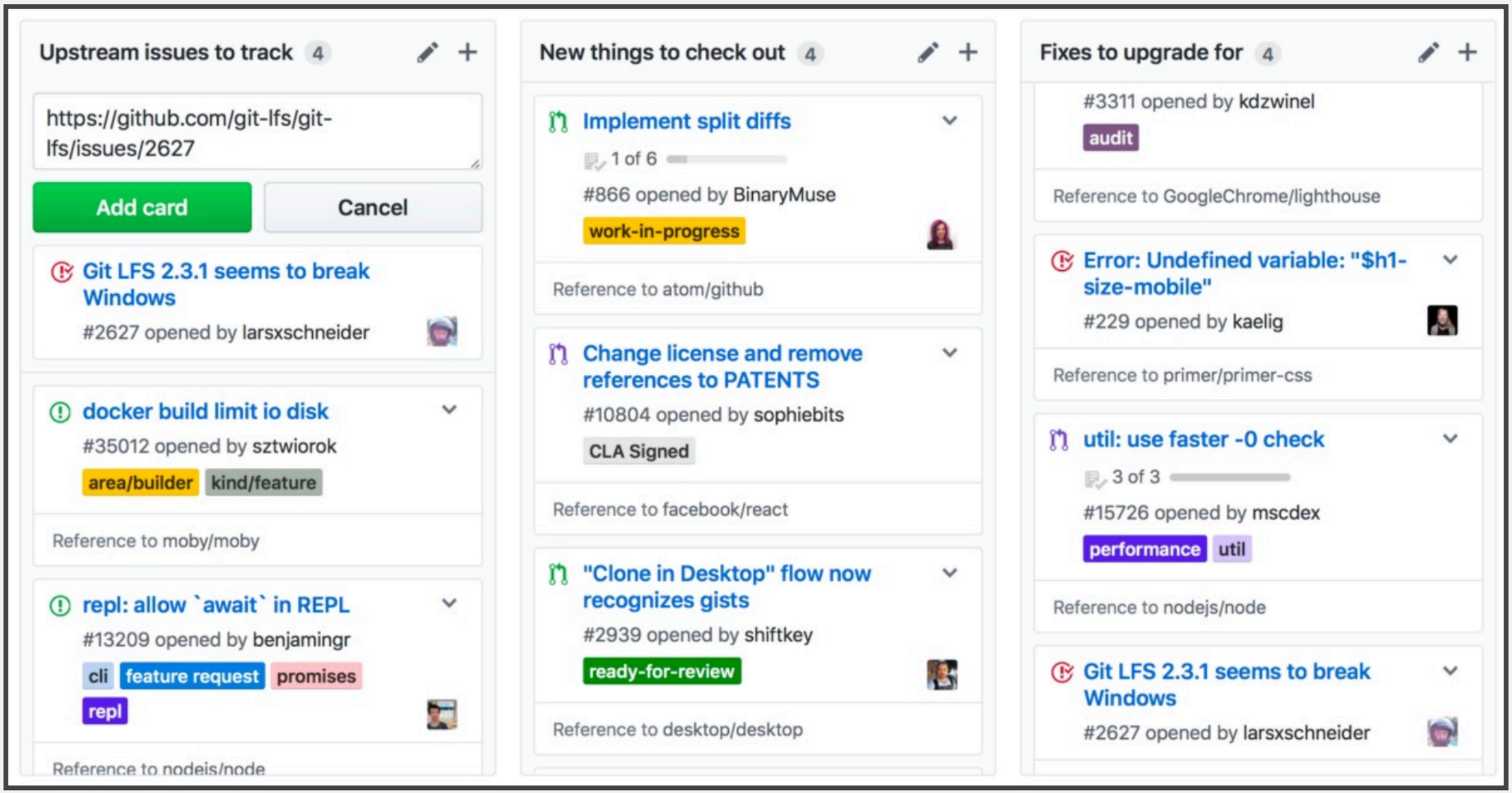
Elements of Scrum



Backlogs

- The product backlog is all the features for the product
- The sprint backlog is all the features that will be worked on for that sprint. These should be broken down into discrete tasks:
  - Fine-grained
  - Estimated
  - Assigned to individual team members
  - Acceptance criteria should be defined
- User Stories are often used

Kanban Boards



Scrum Meetings

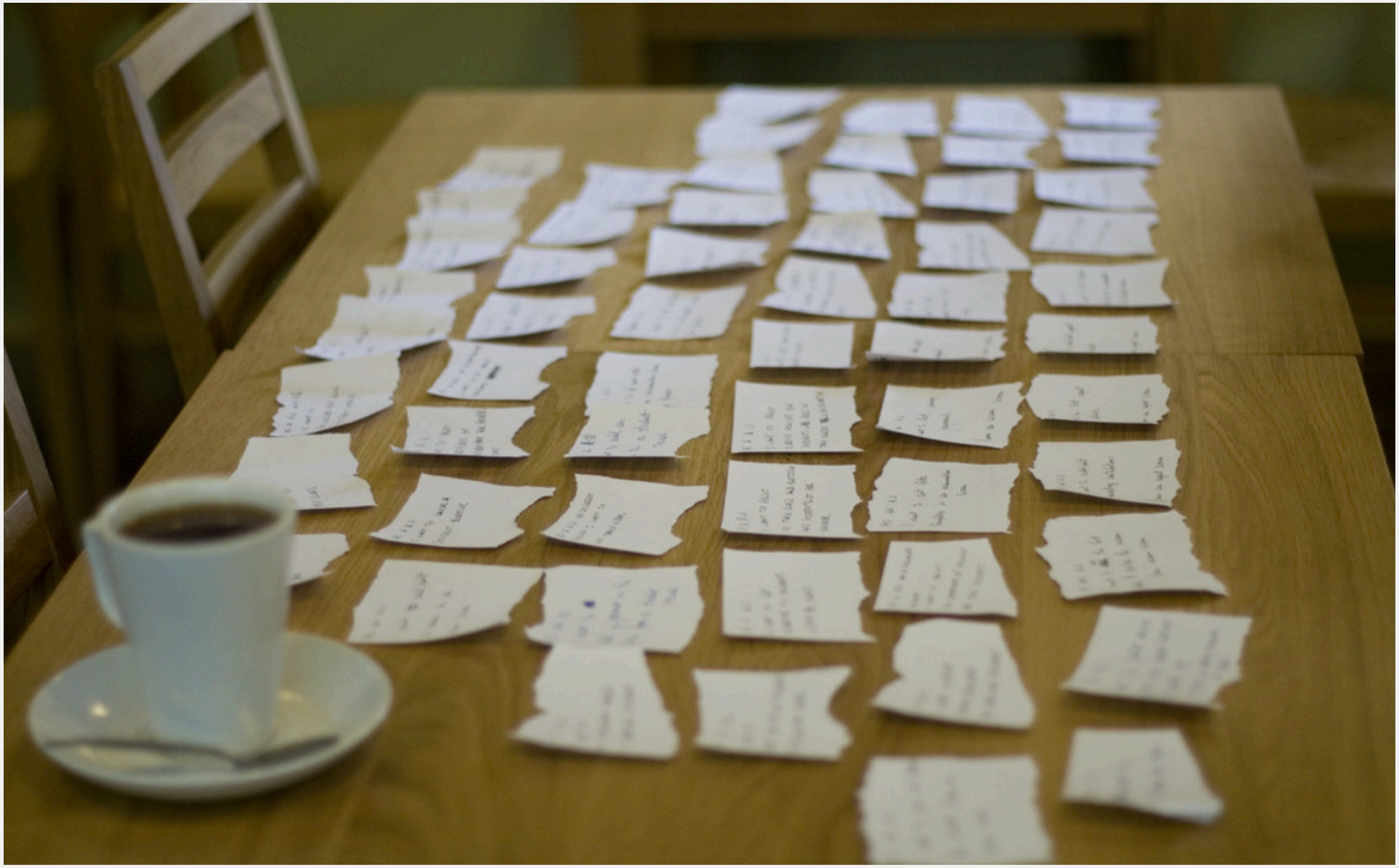
- Sprint Planning Meeting
  - Entire Team decides together what to tackle for that sprint

- Daily Scrum Meeting
  - Quick Meeting to touch base on :
    - What have I done?
    - What am I doing next?
    - What am I stuck on/need help?

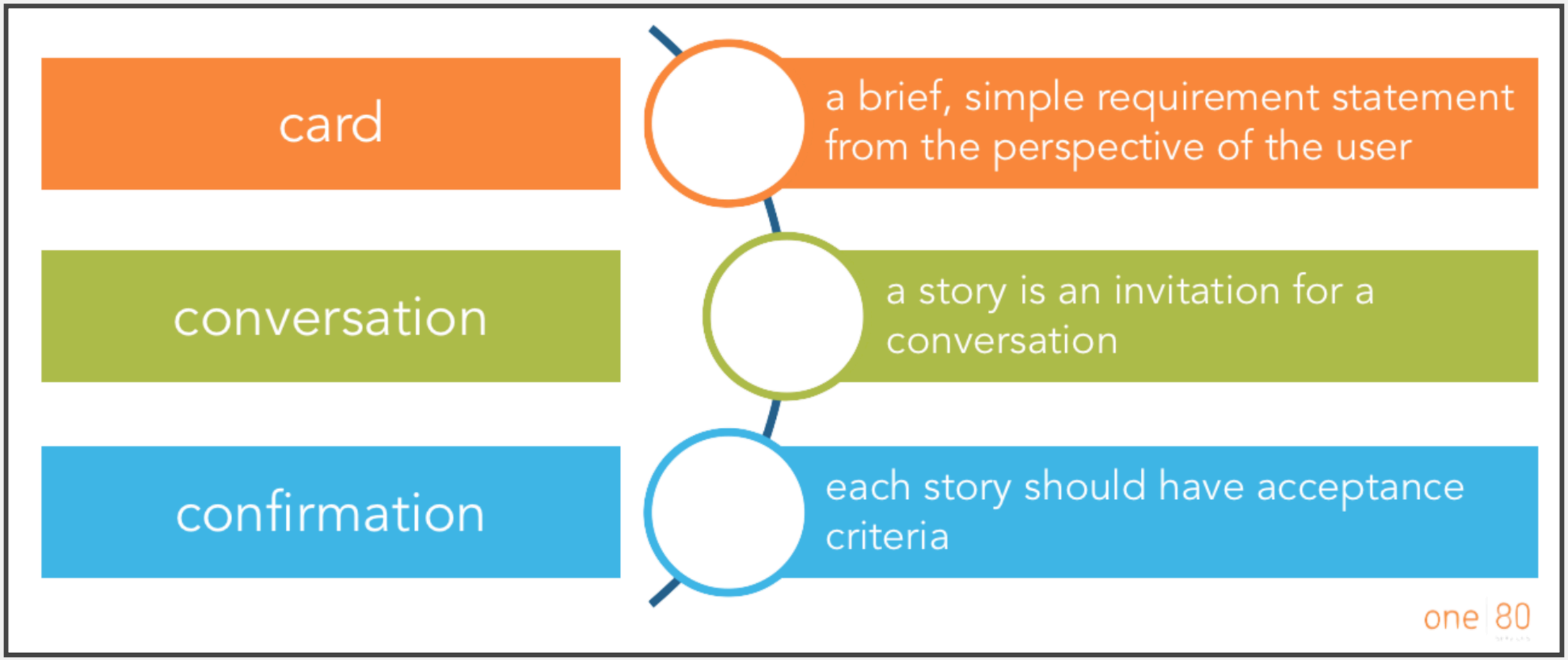
- Sprint Retrospective
  - Review sprint process

- Sprint Review Meeting
  - Review Product

User Stories



User Stories



Card

- “As a [role], I want [function], so that [value]”

Card

- What must a developer do to implement this user story?

Confirmation

- How can we tell that the user story has been achieved
- It's easy to tell when the developer finished the code.
- But, how do you tell that the customer is happy?

How to Evaluate a User Story

Independent

- Schedule in any order.
- Not overlapping in concept.
- Not always possible.

Negotiable

- Details to be negotiated during development.
- A good story captures the essence, not the details.

Valuable

- This story needs to have value to someone (hopefully the customer).
- Especially relevant to splitting up issues.

Estimable

- Helps keep the size small.
- Ensure we negotiated correctly.
- “Plans are nothing, planning is everything” - Dwight D. Eisenhower

Small

- Can be written on a 3x5 card.
- At most two person-weeks of work.
- Too big === unable to estimate

Testable

- Ensures understanding of task
- We know when we can mark task “Done”

- Unable to test === I do not understand it