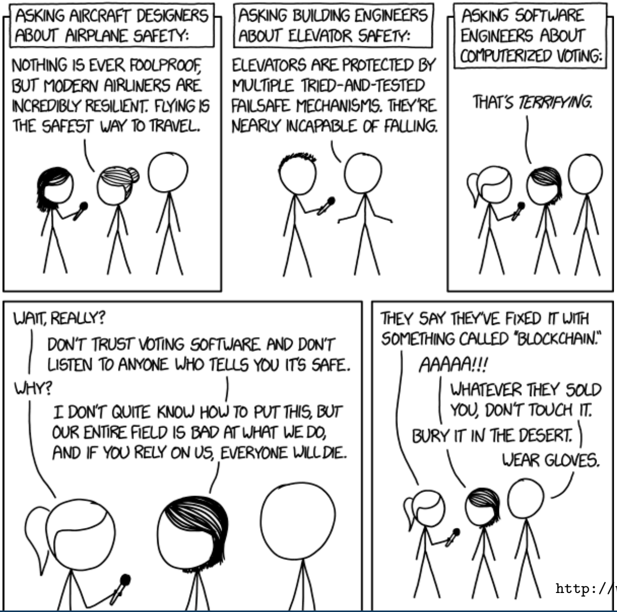
Agile Software Development

7 February, 2022


ASKING AIRCRAFT DESIGNERS ABOUT AIRPLANE SAFETY:
NOTHING IS EVER FOOLPROOF, BUT MODERN AIRLINERS ARE INCREDIBLY RESILIENT. FLYING IS THE SAFEST WAY TO TRAVEL.
ASKING BUILDING ENGINEERS ABOUT ELEVATOR SAFETY:
ELEVATORS ARE PROTECTED BY MULTIPLE TRIED-AND-TESTED FAILSAFE MECHANISMS. THEY'RE NEARLY INCAPABLE OF FALLING.
ASKING SOFTWARE ENGINEERS ABOUT COMPUTERIZED VOTING:
THAT'S *TERRIFYING.*
WAIT, REALLY?
DON'T TRUST VOTING SOFTWARE AND DON'T LISTEN TO ANYONE WHO TELLS YOU IT'S SAFE.
WHY?
I DON'T QUITE KNOW HOW TO PUT THIS, BUT OUR ENTIRE FIELD IS BAD AT WHAT WE DO, AND IF YOU RELY ON US, EVERYONE WILL DIE.
THEY SAY THEY'VE FIXED IT WITH SOMETHING CALLED "BLOCKCHAIN."
AAAAA!!!
WHATEVER THEY SOLD YOU, DON'T TOUCH IT.
BURY IT IN THE DESERT.
WEAR GLOVES.

http://www.xkcd.com

Software Engineering

SE is not “just” programming / coding!

- A formal process to create software.
  - Software: a collection of programs, procedures, rules and associated documentation and data.
- Commercial software:
  - has customers and users;
  - has to meet certain criteria regarding quality, cost and schedule.
- Development of commercial software needs to account for scale and change.
- Need for engineering methods (systematic, disciplined, quantifiable) and project management.

Software Engineering

What makes SE different from other engineering disciplines?

- We are early in history (compared to, e.g., civil engineering).
- Software is an intangible product.
- It is more difficult to see how much of a software product is "done" than how much of a road is "done".
- It may be more difficult to assign responsibility, and blame.
- Complex dependencies present problems for scaling the development, and also the resilience of the product.
- Software is very susceptible to change (or change requests).
    - Often during development, requirements change.
    - It is tempting to *assume* that introducing changes into software is easy.
    - Even finished, there is often need for both corrective and adaptive maintenance.

Overview

Part 1. What is agile development, and why do we want to use it?

Part 2. How to use agile development in practice – Scrum

Failed projects

- [Thomas01] in a UK study on IT projects, 87% failed
- [Jarzombek99] in a Department of Defence study: 75% failed
- [Jarzombek99] 46% of delivered software products was never used (20% needed an extensive rework)
- [CLW01] 90% of code was never deployed
  (80% of what was deployed was never used)

- [Pulse17] 14% of IT projects are deemed failures
  - champions: $\geq 80\%$ of projects are completed in time / budget
  - underperformers: $\leq 60\%$ of projects are completed in time / budget
  - on average: 97 / 1000 dollars wasted

Some reasons why projects fail

- failing developer motivation
- hard to estimate completion time beforehand
- lack of communication
- lack of adaptability
- incorrect requirements
- lack of dedicated testing for parts

A recurring theme: changing requirements

- expected use does not match real use because customer does not fully understand users
- the existence of a feature causes unanticipated new wishes
- requirements that are known but not specified
- *I'll know it when I see it.*

A key principle

Do not fight change.
Embrace it!

Succesful projects (without SE methods)

- really stubborn coders
- small projects (that later got stuff added to them)
- projects that can be cut into small pieces and tested separately
- feature removal

Gall's Law

A complex system that works is invariably found to have evolved from a simple system that worked.
A complex system designed from scratch never works and cannot be patched up to make it work.
You have to start over with a working simple system

John Gall (1975) Systemantics: How Systems Really Work and How They Fail

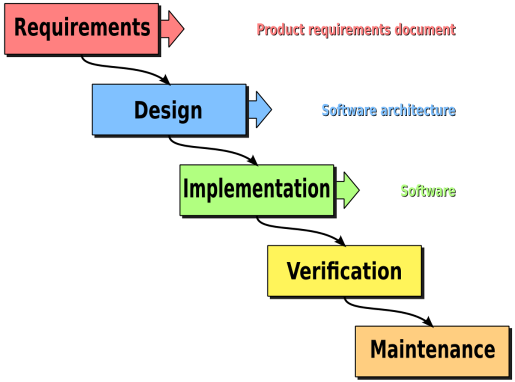
The agile manifesto

We are uncovering better ways of developing software by doing it and helping others do it. 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

That is, while there is value in the items on the right, we value the items on the left more.

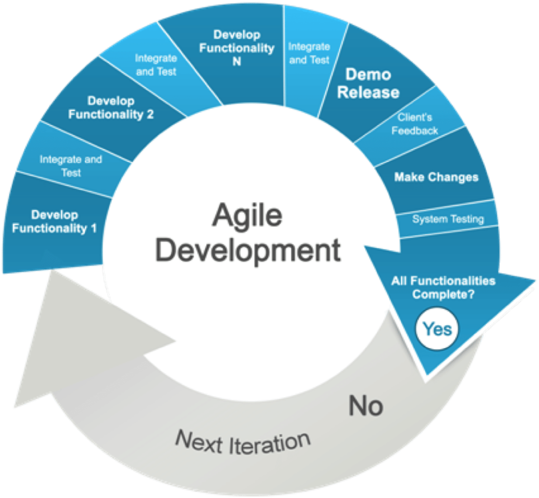
A plan-driven process model: Waterfall



- separation of concerns
- phases have clear objectives
- certification (and payment) of intermediate products

- assumes fixed requirements
- encourages requirements bloating
- final deliverable: all or nothing

Agile development



Agile principles

Agile principles

Satisfy the customer through early and continuous delivery of working software.

Agile principles

Welcome changing requirements. Harness change for the customer's competitive advantage.

Agile principles

Business people and developers must work together throughout the project.

Agile principles

Working software is the primary measure of progress.

Agile principles

Simplicity is essential.

Do as little as possible.

Agile principles

Convey information to and within the team through face-to-face conversation.

Agile principles

Build projects around motivated individuals.
Give them space and support.

Agile principles

The best work emerges from self-organising teams.

Agile principles

At regular intervals the team reflects, then tunes and adjusts their behaviour accordingly.

Agile principles

Development should be sustainable.

Maintain a constant pace.

Agile principles

Continuous attention to
technical excellence and good design.

Core practices

- time-boxed iterations
  - between one and six weeks, set beforehand
  - no requirements may be added or changed during an iteration
  - requirements may be removed if it seems that the deadline will not be met (but only if you really have to)
- incremental delivery
  - risk-driven and client-driven planning
  - get a working product ASAP, build from there
- communication
  - regular communication between customers and developers
  - evolutionary requirements analysis and adaptive planning using results of early work
  - constant in-team communication
- respect for developers
  - teams have input on (take the lead in) planning and organisation
  - maintain a sustainable pace

A common mistake

"Oh yes, we're using agile methods! We've just finished the plan of what we'll do in each iteration..."

An agile anecdote

[9 January]

Celt: Fishing is huge, [my manager] and I went a little mad with the feature creep and now it's a monster. So I'm trying to slim the requirements down.
Celt: But I am not sure what the right design is. Should I use effects, or do everything from the fishing rod?
Celt: The old idea didn't work well. I don't want to get a few months down the track and run into problems again.
Me: If is is even possible that you figure out something is wrong months down the line, you're doing it wrong...
Me: You make a single fish, a single rod, and a command to get the fish with the rod. Then see what's next.

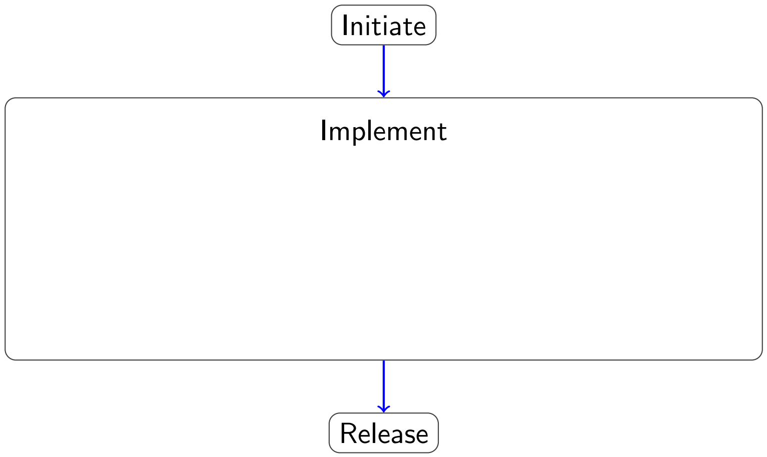
An agile anecdote

[20 February]

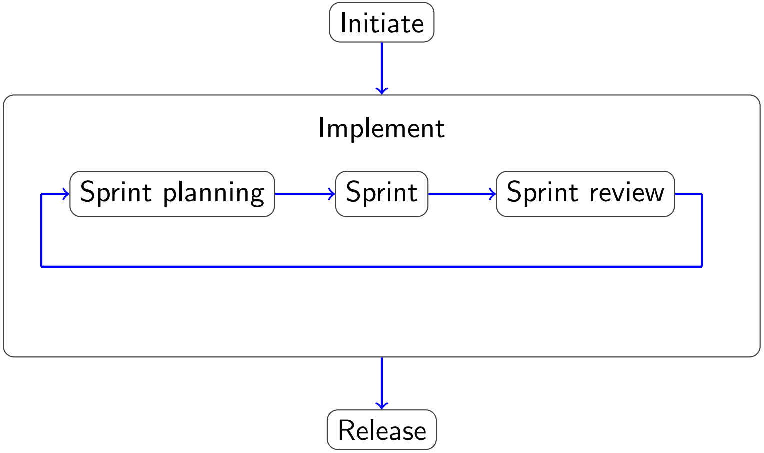
You: So how's the fishing going? Are you still going with the agile rewrite, or are you fiddling with the old system?
Celt: Agile, baby!
Celt: Well, it's constantly functioning now. It works from start to finish, and I'm pretty happy with how it is.
Celt: So everything I'm doing now is fleshing it out and making parts of it more interesting.

SCRUM

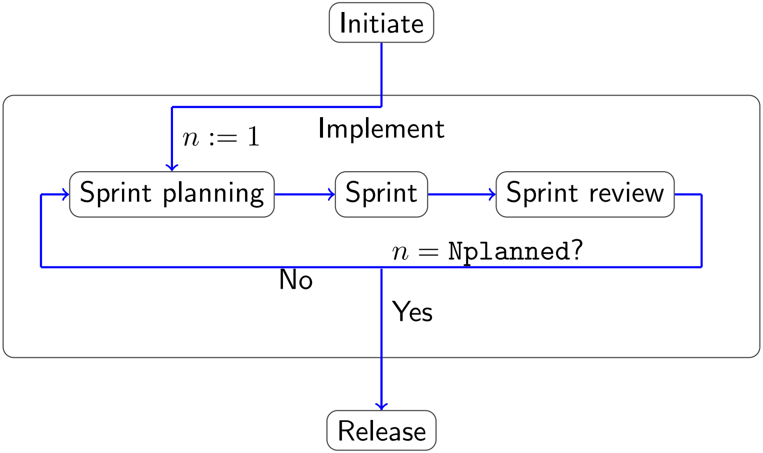
The Scrum lifecycle



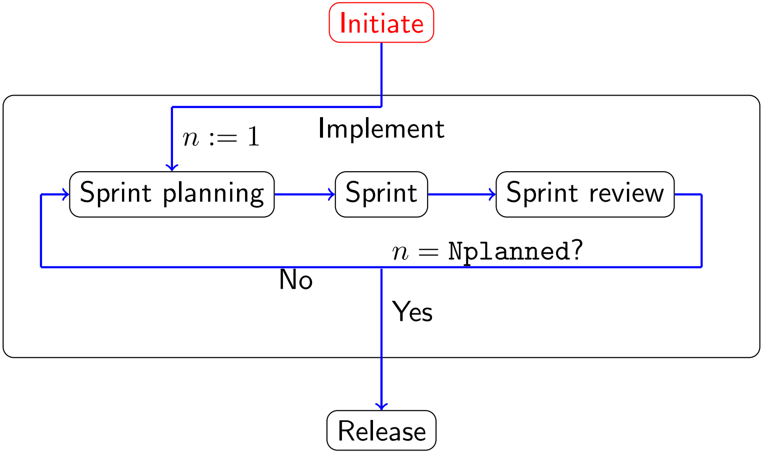
The Scrum lifecycle



The Scrum lifecycle



The Scrum lifecycle



Scrum Initiation

- create project vision
- create initial product backlog: prioritised requirement list
- create initial user stories
- designate Product Owner and Scrum Master
    - Product Owner: product visionary (often: key stakeholder)
    - Scrum Master: removing blocks, making decisions
- assemble development team (3-9 people, not necessarily just programmers!)
- make agreements on development process (tool use, definition of done, code reviews, etc.)
- create initial time estimates

Running example: create project vision

Goal: a website for publishing and reading stories
Vision: Provide an easy way for authors to make their stories available and earn money with them. Provide an easy way for readers to find and read good stories.
Vision statement following a template:

- **For** ⟨target customer⟩
- **who** ⟨statement of need/opportunity⟩,
- **the** ⟨product name⟩ **is a** ⟨product category⟩
- **that** ⟨key benefit/reason to buy⟩.
- **Unlike** ⟨main competitive alternative⟩,
- **our product** ⟨statement of primary differentiation⟩

(Note: this is only an example; you can use your own template or no template at all. A template only serves as a guideline.)

Running example: create project vision

Goal: a website for publishing and reading stories

Vision: Provide an easy way for authors to make their stories available and earn money with them. Provide an easy way for readers to find and read good stories.

Vision statement following a template:

- **For** authors
- **who** wish to make their work available,
- **the** FictionPublishingSite **is an** online browser-based publication tool
- **that** allows users to immediately make their story available to the broad public.
- **Unlike** traditional publication methods,
- **our product** does not require formal approval by an editor, but instead works with feedback from actual readers.

Running example: create initial backlog and user stories

- uploading stories or chapters
- reading stories online
- downloading e-books
- searching for stories
- discussion forums per story
- a rating system on several characteristics

Running example: create initial backlog and user stories

- uploading stories or chapters
  - As a ⟨role⟩ I can ⟨what⟩ so that ⟨why⟩.
- reading stories online
- downloading e-books
- searching for stories
- discussion forums per story
- a rating system on several characteristics

Running example: create initial backlog and user stories

- uploading stories or chapters
  - As an author I can upload a story from my computer so that it is in my account, to edit or publish.
  - As an author I can upload a single chapter from my computer so that it is in my account.
- reading stories online
- downloading e-books
- searching for stories
- discussion forums per story
- a rating system on several characteristics

Running example: create initial backlog and user stories

- creating accounts
  - As an author I can create an account so that my stories are saved under my name.
- uploading stories or chapters
  - As an author I can upload a story from my computer so that it is in my account, to edit or publish.
  - As an author I can upload a single chapter from my computer so that it is in my account.
- reading stories online
- downloading e-books
- searching for stories
- discussion forums per story
- a rating system on several characteristics

Running example: create initial backlog and user stories

- creating accounts
  - As an author I can create an account so that my stories are saved under my name.
- uploading stories or chapters
  - As an author I can upload a word/latex/html document from my computer so that it is in my account.
- reading stories online
- downloading e-books
- searching for stories
- discussion forums per story
- a rating system on several characteristics

Running example: create initial backlog and user stories

- creating accounts
  - As an author I can create an account so that my stories are saved under my name.
- uploading stories or chapters
  - As an author I can upload a word/latex/html document from my computer so that it is in my account.
- reading stories online
- downloading e-books
- searching for stories
- discussion forums per story
- a rating system on several characteristics
- user accounts
  - As an author I can edit the text of my story directly online, so that I can easily correct small mistakes.
  - As an author I can split and merge chapters so that I have full control over what to publish when.
  - As an author I can publish one or more chapters of a story, so

Running example: create initial backlog and user stories

- uploading stories or chapters
  - As a ⟨role⟩ I can ⟨what⟩ so that ⟨why⟩
- reading stories online
- downloading e-books
- searching for stories
- discussion forums per story
- a rating system on several characteristics

Note: format for user stories not mandatory; just a guideline to push your thoughts in the right direction.

Running example: create prioritised product backlog

• minimal account creation	1 day
• minimal writing and editing chapters in the browser	1 week
• figuring out the legalities of copyright and donations	2 days
• publishing chapters and full stories	3 days
• searching for stories by keywords and genre	1 day
• rating a story on language, plot, humour, etc.	1 week
• filtering story search based on ratings, length, newness	1 day
• extended account creation	2 weeks
• donation button hooked up to paypal	1 week
• . . .	

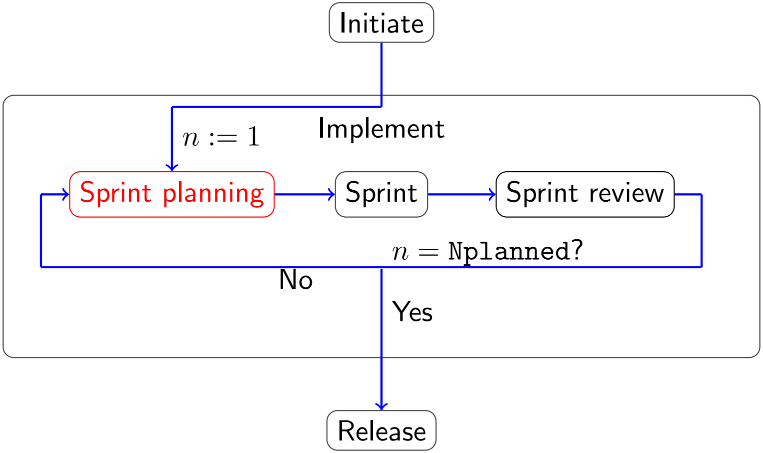
Running example: Definition of Done

Definition of Done: a feature is finished if:

- the code is complete
- the code is entirely unit-tested
- the code has been reviewed and accepted by at least one team member
- the feature has been acceptance-tested
- a manager has signed off on it

Note: this is just an example; teams decide for themselves what their DoD is.

The Scrum lifecycle



Sprint Planning

- refine and reprioritise product backlog
- improve relevant user stories
- (re-)estimate times for features in product backlog
- create Sprint backlog of tasks (4–16 hours)
- estimate time for each task using in-team discussions and (e.g.) planning poker
- estimated time may not exceed available time!

Planning poker

- all players are given a deck of cards:
  1, 2, 3, 5, 8, 13, 21, 34 . . . and ∞, COFFEE
- when estimating a task, cards are played face-down
- cards are turned over at the same time, and discussion ensues
- repeat until a consensus or compromise is reached

Running example: create initial Sprint backlog

- designing and setting up the database
- setting up the server
- “create account” webpage with one textfield
- contacting legal to figure out copyright laws
- story creation (overall data, without chapters)
- chapter creation without any lay-out features
- marking chapters in a story as published
- presenting published chapter as HTML for publication
- present story as a whole (with buttons, index)
- . . .

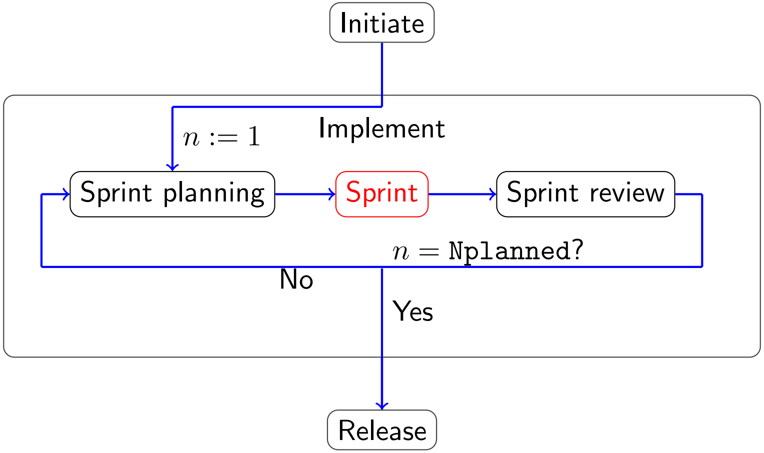
Running example: create initial Sprint backlog

• designing and setting up the database	8 hours
• setting up the server	5 hours
• “create account” webpage with one textfield	1 hours
• contacting legal to figure out copyright laws	13 hours
• story creation (overall data, without chapters)	3 hours
• chapter creation without any lay-out features	2 hours
• marking chapters in a story as published	5 hours
• presenting published chapter as HTML for publication	5 hours
• present story as a whole (with buttons, index)	21 hours
• . . .	

Running example: create initial Sprint backlog

• designing and setting up the database	8 hours
• setting up the server	5 hours
• “create account” webpage with one textfield	1 hours
• contacting legal to figure out copyright laws	13 hours
• story creation (overall data, without chapters)	3 hours
• chapter creation without any lay-out features	2 hours
• marking chapters in a story as published	5 hours
• presenting published chapter as HTML for publication	5 hours
• present story as a whole (with buttons, index)	21 hours
• . . .	

The Scrum lifecycle



Scrum Sprint

- usually 60 days (but here: 3 weeks)
- work is done in a common project room (or: digital room)
- the team has the authority and resources to find their own way
- daily Scrum meeting, attended by the full team
- daily build, integrating all code
- Sprint backlog continuously updated
- Sprint refinement: refining other items on the product backlog

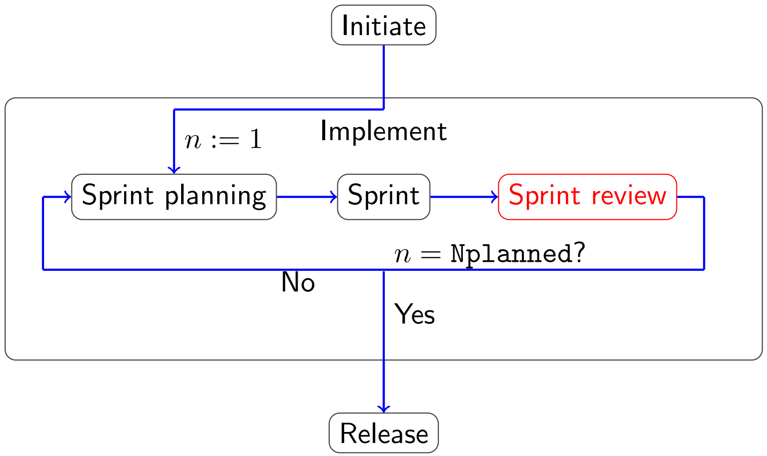
Stand-up Meeting

- short meeting: about 2 minutes per person
- people are standing up!
- everyone answers the following questions:
  - What have you done since the last meeting?
  - What will you do before the next meeting?
  - What is getting in the way of meeting the Sprint goals?
  - (MAYBE) Any missed tasks to add to the Sprint backlog?
  - (MAYBE) Have you learned or decided anything new?
- reported blocks should be removed before the next meeting
- necessary decisions taken in one hour
- note: outsiders (e.g., CEO, customers) may attend, but not speak (unless asked for clarification)

Running example: stand-up meeting

- **SM:** What have you done since last meeting?
- **P:** I have designed the database
- **SM:** What will you do before the next meeting?
- **P:** Implementing the database, and putting some default users in
- **SM:** What is getting in the way of meeting the Sprint goals?
- **P:** The database is different from what I learned. I may need some help in figuring out the right SQL queries.
- **SM:** Okay, let's update the planned time. How long do you think this will take you?
- ...
- **Overall:** discussion possible, but try to stay below 2–3 minutes each
- Scrum Master is responsible for keeping the meeting on track!

The Scrum lifecycle



Sprint review

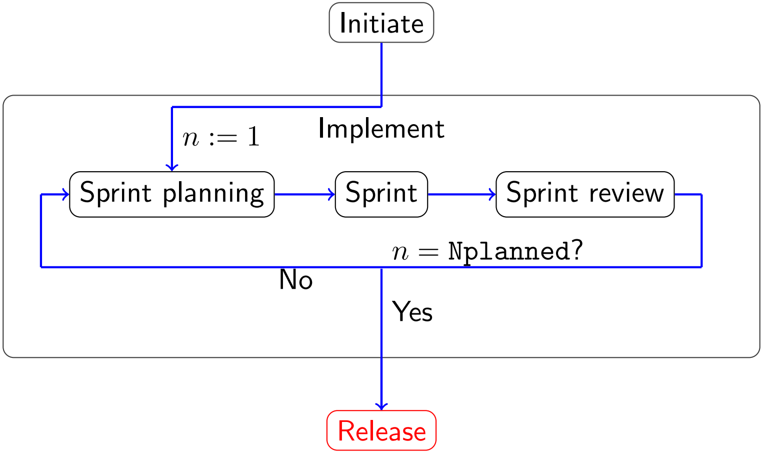
- meeting attended by team, Scrum Master, Product Owner and stakeholders
- discuss what has been accomplished!
- should include a product demo, no powerpoint!
- inform stakeholders of the system functions, choices, strengths, weaknesses, and future trouble spots
- feedback and brainstorming on directions, but no commitments

Sprint retrospective

A short meeting where the team discusses:

- What went well in the sprint?
- What could be improved?
- What do we commit to improve in the next sprint?

The Scrum lifecycle



A different agile approach: eXtreme Programming

- emphasis on oral communication:
    - minimal requirements documentation; just story cards
    - everyone in one room, including at least one customer
    - pair programming with regularly mixed pairs
    - entire team responsible for all code
- evolutionary delivery through small, frequent releases
- test-driven development:
    - automatic acceptance tests for all features
    - unit tests for most code
    - first write the test, then write the code that makes it succeed
    - continuous integration on a dedicated machine that runs all tests
- frequent refactoring

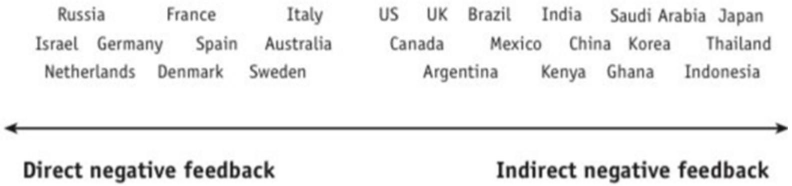
Methodologies change

From a friend:

In college, I was trained as a top-down, structured software developer. . . in Fortran. Waterfall development cycle was next. Object Oriented if it existed, was just starting to take shape at Bell Laboratories. Then down the road, drifted in to "Patterns". And of course, "Anti-Patterns". Now "agile". Oh, and CAS (computer aided software) design is in that methodology mix too.

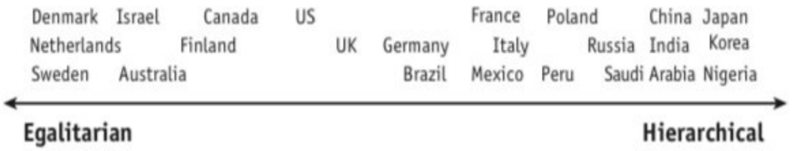
Some final notes on culture

Some final notes on culture



Source: *The Culture Map: Breaking Through the Invisible Boundaries of Global Business*

Some final notes on culture



Source: *The Culture Map: Breaking Through the Invisible Boundaries of Global Business*

Some final notes on culture

- People from different cultures have different ways of communicating.
- Be aware of your own cultural habits and assumptions.
- If you aren't sure how something is meant, ask.
- Have a discussion about your team approach to:
  - feedback
  - management
  - sharing ideas
  - resolving conflicts
  - building the team
  - timing and deadlines

Resources

- [Thomas01] Thomas, M. 2001. “IT Projects Sink or Swim.” *British Computer Society Review*
- [Jarzombek99] Jarzombek, J. 1999. *The 5th annual JAWS S3 Proceedings*
- [CLW01] Cohen, D., Larson, G. and Ware, B. 2001. “Improving Software Investments through Requirements Validation.” *IEEE 26th Software Engineering Workshop*
- [Pulse17] Global Management Survey. 2017. “Success Rates Rise – Transforming the high cost of low performance.” *PMI’s Pulse of the Profession*