Software Engineering Theory and Practice

School of Computing	
Title	Software Engineering Theory and Practice
Module Coordinator	Steven Ossont
Email	steven.ossont@port.ac.uk
Code	M30819
Moodle	https://moodle.port.ac.uk/course/view.php?id=11429

U30819: Software Engineering Theory and Practice

Agile

https://moodle.port.ac.uk/course/view.php?id=11429

Steven Ossont https://github.com/orgs/M30819-2020/teams/students/discussion

Software Engineering is different from hacking because...

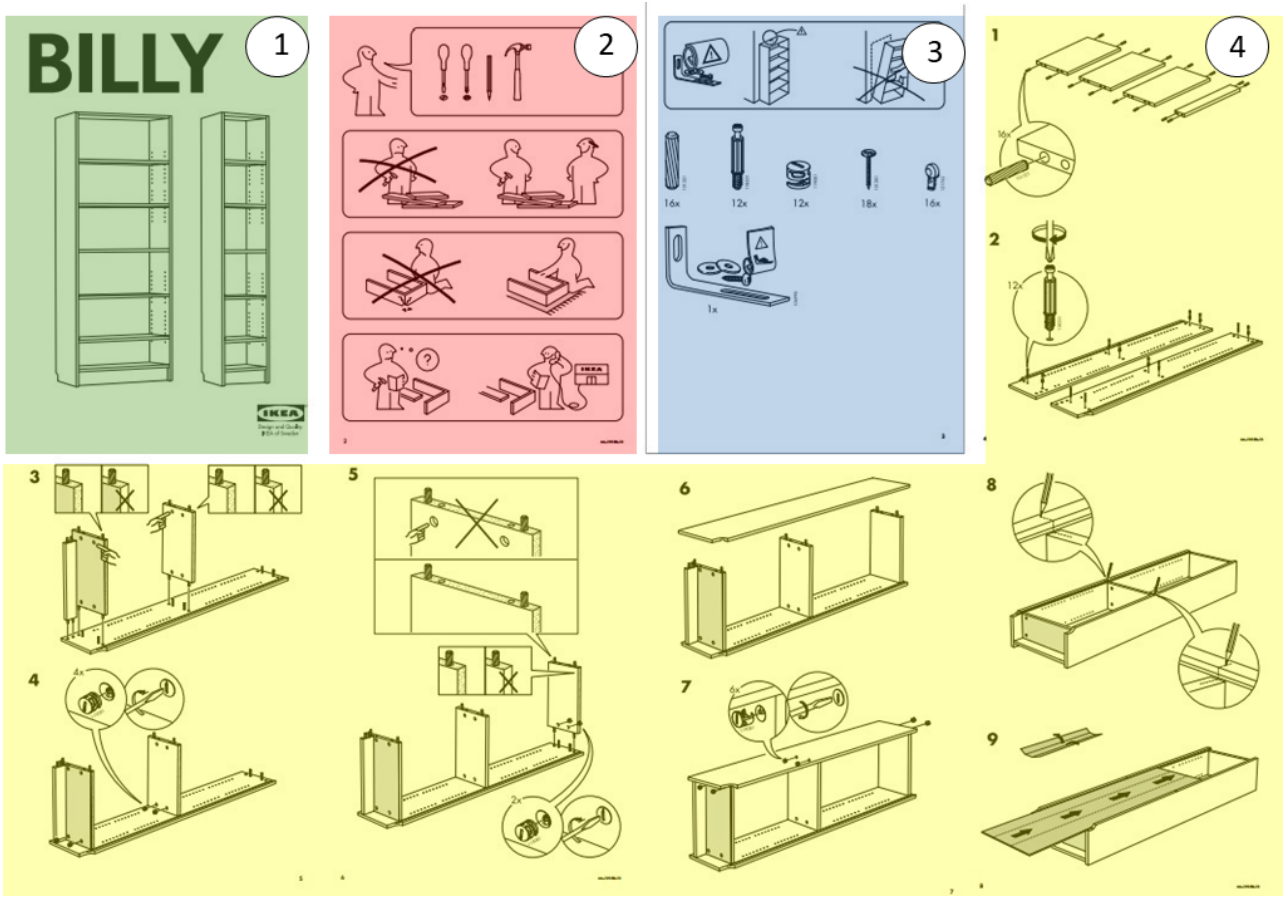
... there is a process!

What is a process and why do you need it?

- A process is followed by **everyone**
- A process is documented
- A process helps with coordination
- A process is repeatable
- A process is efficient and effective
- A process helps achieve complexity

The IKEA example

1. Objective
2. Rules
3. Components
4. **The process**



Software Process

- *"A software process is a set of related activities that leads to the production of a software system"* - Sommerville

There is no ideal/perfect software process

Software is extremely varied

Software Engineering only applies to ALL projects

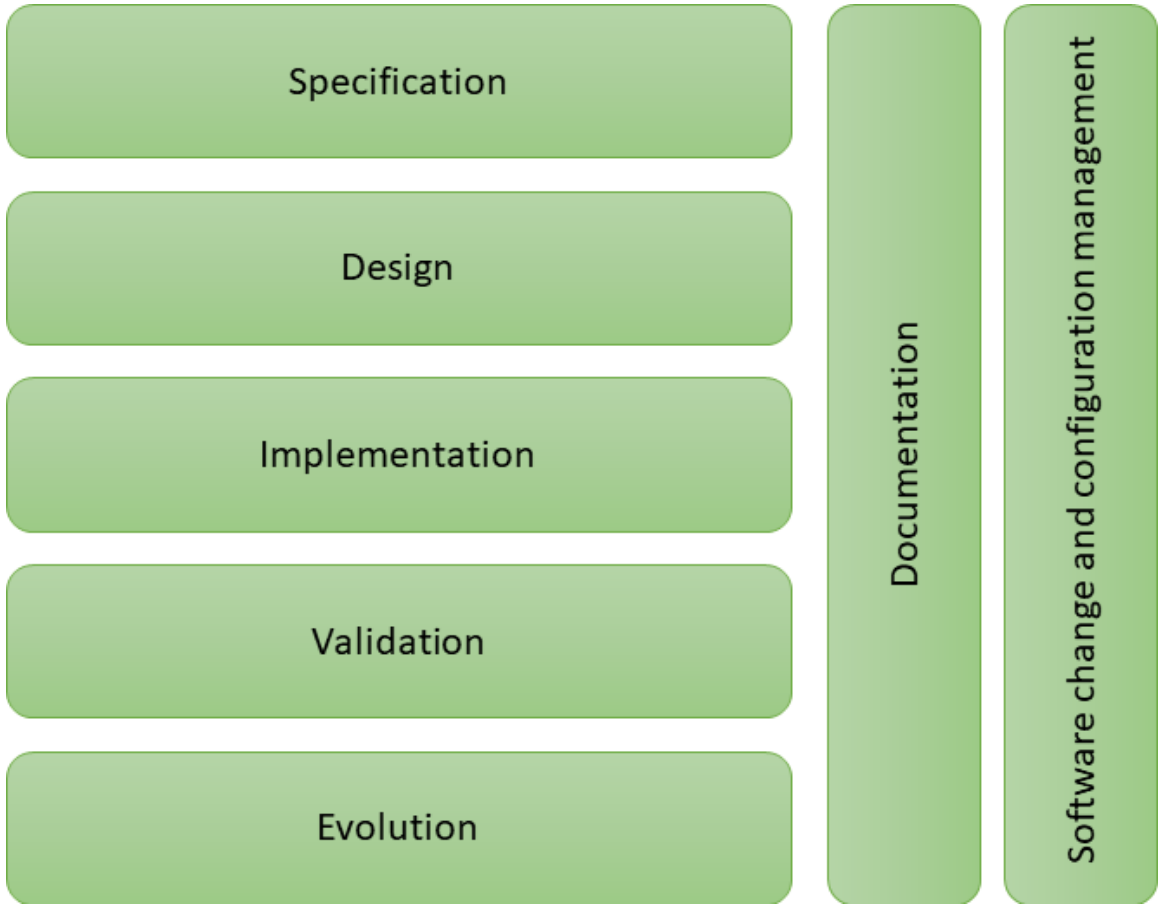
You should use the Software Engineering processes that best suits **your** projects **at all times**

> The SW-Eng philosophy will help you on all of your projects. Even the single use script.

Software Process - Fundamental activities

A Software Engineering process must include these **four** activities:

1. Software specification (or requirements engineering)
2. Software development (design and implementation)
3. Software verification and validation
4. Software evolution (Software maintenance)



Software process models

The waterfall model

Incremental development

Reuse-oriented software engineering

Agile Software Development

- Aim to reduce the overheads in software development processes by limiting documentation
- Allows quick response to changing requirements without excessive rework
- February 2001, 17 developers met in Utah to discuss lightweight software development; they published the **Agile Manifesto**

Manifesto for Agile Software Development

http://agilemanifesto.org

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**.

Individuals and Interactions

- “A good process will not save a project from failure if the team doesn’t have strong players, but a bad process can make even the strongest of players ineffective.”
- More sophisticated and better tools will not automatically help you do better
- “Building the team is more important than building the environment.”

(Robert C. Martin)

Working Software

- “Software without documentation is a disaster. […] However, too much documentation is worse than too little.”
- Always maintain a short, high level rationale and structure document
- “The two documents that are the best at transferring information to new team members are the **code** and the **team**.”

(Robert C. Martin)

Customer Collaboration

- “Successful projects involve customer feedback on a regular and frequent basis.”
- “The best contracts are those that govern the way the development team and the customer will work together” rather than detailing the scope and schedule of the project.

(Robert C. Martin)

Responding to Change

- “When we build plans, we need to make sure that they are flexible and ready to adapt to changes in the business and technology.”
- “Make detailed plans for the next week, rough plans for the next 3 months, and extremely crude plans beyond that”

(Robert C. Martin)

We follow these principles:

http://agilemanifesto.org/principles.html

We follow these principles:

Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.

- Deliver rudimentary system within the first weeks of the start of the project
- Continue to deliver increasing functionality every few weeks
- Customers either review the system or decide to put it in production

We follow these principles:

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

- Keep the architecture flexible so that the impact of changes is minimal

We follow these principles:

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

- Documents and plans are not deliverables
- Working software is a deliverable
- Deliver every couple of weeks

We follow these principles:

Business people and developers must work together daily throughout the project.

We follow these principles:

Build projects around motivated **individuals**. Give them the environment and support they need, and trust them to get the job done.

- People are the most important success factor
- Processes, environment, management need to change to match the people

We follow these principles:

The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

- People talk to one another
- Documents are created/updated on the same schedule as the software

We follow these principles:

Working software is the primary measure of progress.

- Progress is not measured in terms of the phase of the project or the volume of documentation
- Progress is only measured in the amount of software that is currently meeting the customer’s needs

We follow these principles:

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a **constant pace indefinitely**. Continuous attention to **technical excellence** and good design enhances agility.

- Work at a rate that you can maintain (long distance running)
- Maintain the same rhythm for the duration of the project

We follow these principles:

Simplicity -- the art of maximizing the amount of work not done -- is essential.

- Always the simplest path that is consistent with your goals

We follow these principles:

The best architectures, requirements, and designs **emerge** from self-organizing teams.

- Keep the software as clean and as robust as possible as you go along
- Do not write messy code that works in the hope you'll clean it up later
- Agile team members work together on all aspects of the project
- No single team member is solely responsible for any one part of the project

We follow these principles:

At regular intervals, the team reflects on how to become more effective, then **tunes and adjusts** its behavior accordingly.

- Continually adjust the organization of the team, its rules, conventions, etc.

Agile vs Traditional (Waterfall or Spiral) Development

Traditional: Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning.

Agile: High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.

Control

Traditional: Process-centric

Agile: People-centric

Management Style

Traditional: Command-and-control

Agile: Leadership-and-collaboration

Role Assignment

Traditional: Individual—favors specialization

Agile: Self-organizing teams—encourages role interchangeability

Communication

Traditional: Formal

Agile: Informal

Customer’s Role

Traditional: Important

Agile: Critical

Project Cycle

Traditional: Guided by tasks or activities

Agile: Guided by product features

Technology

Traditional: No restriction

Agile: Favors object-oriented technology

Development Model

Traditional: Life cycle model (Waterfall, Spiral...)

Agile: The evolutionary-delivery model

Agile Methods

There are software processes that **implement** the agile principles, for example:

- Scrum
- Extreme programming (XP)
- Feature drive development
- Kanban
- Lean

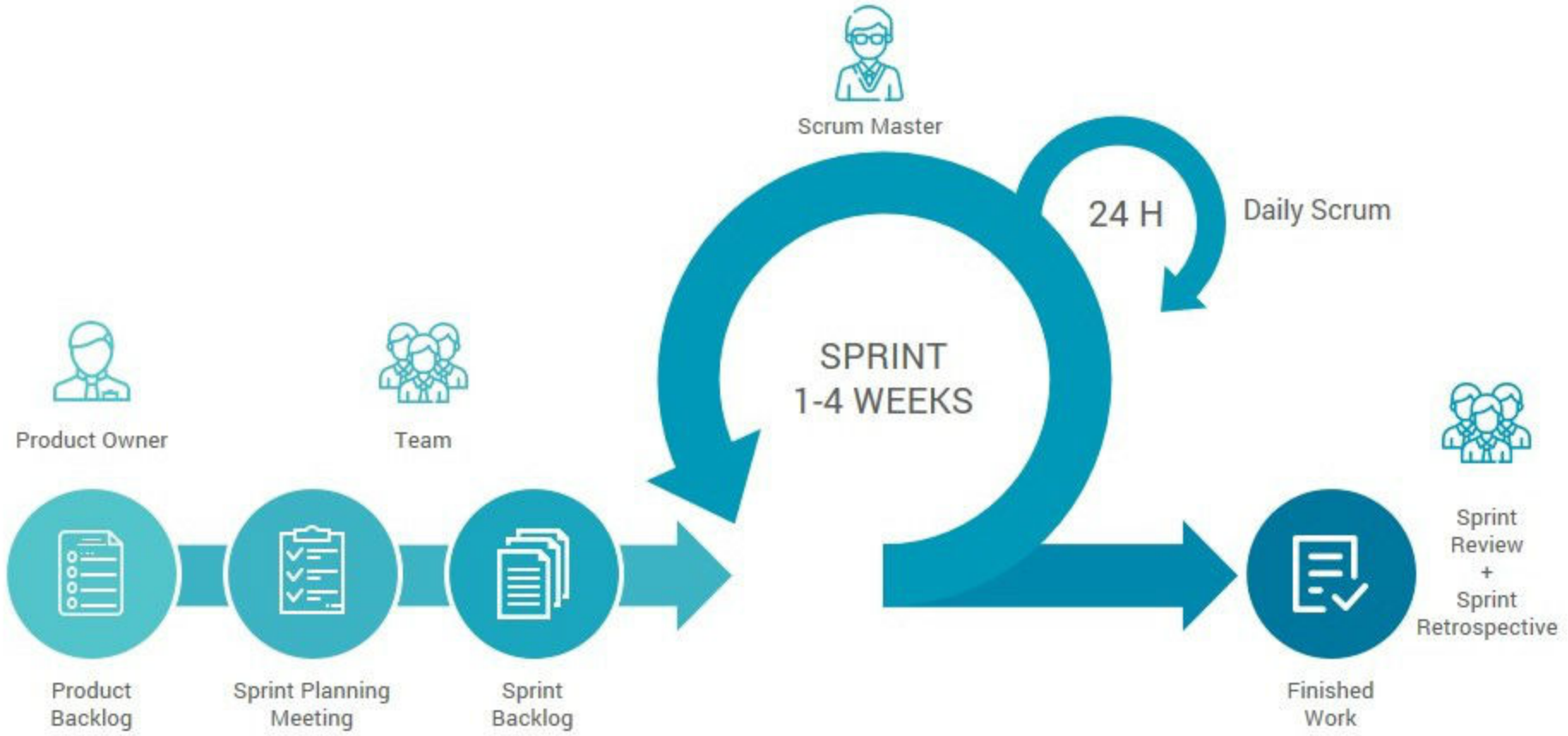
Scrum

Scrum is a process framework that has been used to manage work on complex products, it is not a process, technique, or definitive method. It is a **framework** within which you can employ various processes and techniques.

Scrum makes clear the relative efficacy of your product management and work techniques so that you can continuously improve the product, the team, and the working environment.

> Scrum employs an iterative, incremental approach to optimize predictability and control risk. https://www.youtube.com/watch?v=ITOWKUjr8VA

Scrum Overview



Roles in Scrum - Product Owner (PO)

- Represents the customer
- Prioritizes the requirements

Product Backlog

- Managed by the Product Owner (PO)
- Master list of all functionalities desired in the product
- Ordered by the Product Owner (PO) to best achieve product goals
- Visible to everyone

> User stories can be used fill the Product Backlog

Roles in Scrum - Scrum master

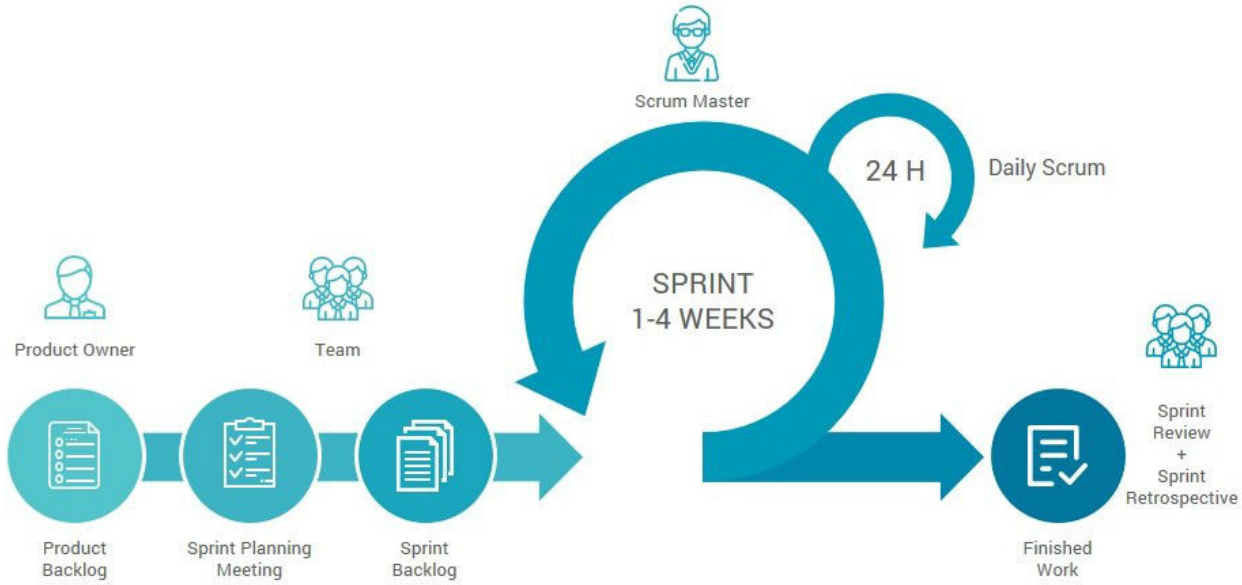
- Arranges daily meetings
- Tracks the backlog of work to be done
- Records decisions
- Measures progress against the backlog
- Communicates with customers

Roles in Scrum - Team

- The developers

Sprint Planning Meeting

- Attended by all (Product Owner, Scrum Master, Team)
- Set Sprint objective/goal
- Backlog is negotiated
- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?



Sprint Goal

- The Sprint Goal is an objective set for the Sprint that can be met through the implementation of Product Backlog.

Daily Scrum

- Held **every** day (time restricted: e.g. 15min)
- Inspect progress toward the Sprint Goal
- Progress on completing the work in the Sprint Backlog.

For example: (for every team member)

- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?
- The team participate, the Scrum Master is responsible to ensure it takes place.

> You can follow this with detailed discussions

Finished work (an increment)

- The Increment is the sum of all the Product Backlog items completed during a Sprint
- Increment must be **Done**, which means it must be in useable condition
- The increment is a step toward a vision or goal
- Product Owner decides to release it or not.

Sprint Review Meeting

- Held at the end of each sprint
- Team demonstrates the product increment to the Product Owner
- The project is assessed against the sprint goal
  - determined during the Sprint Planning meeting

> This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

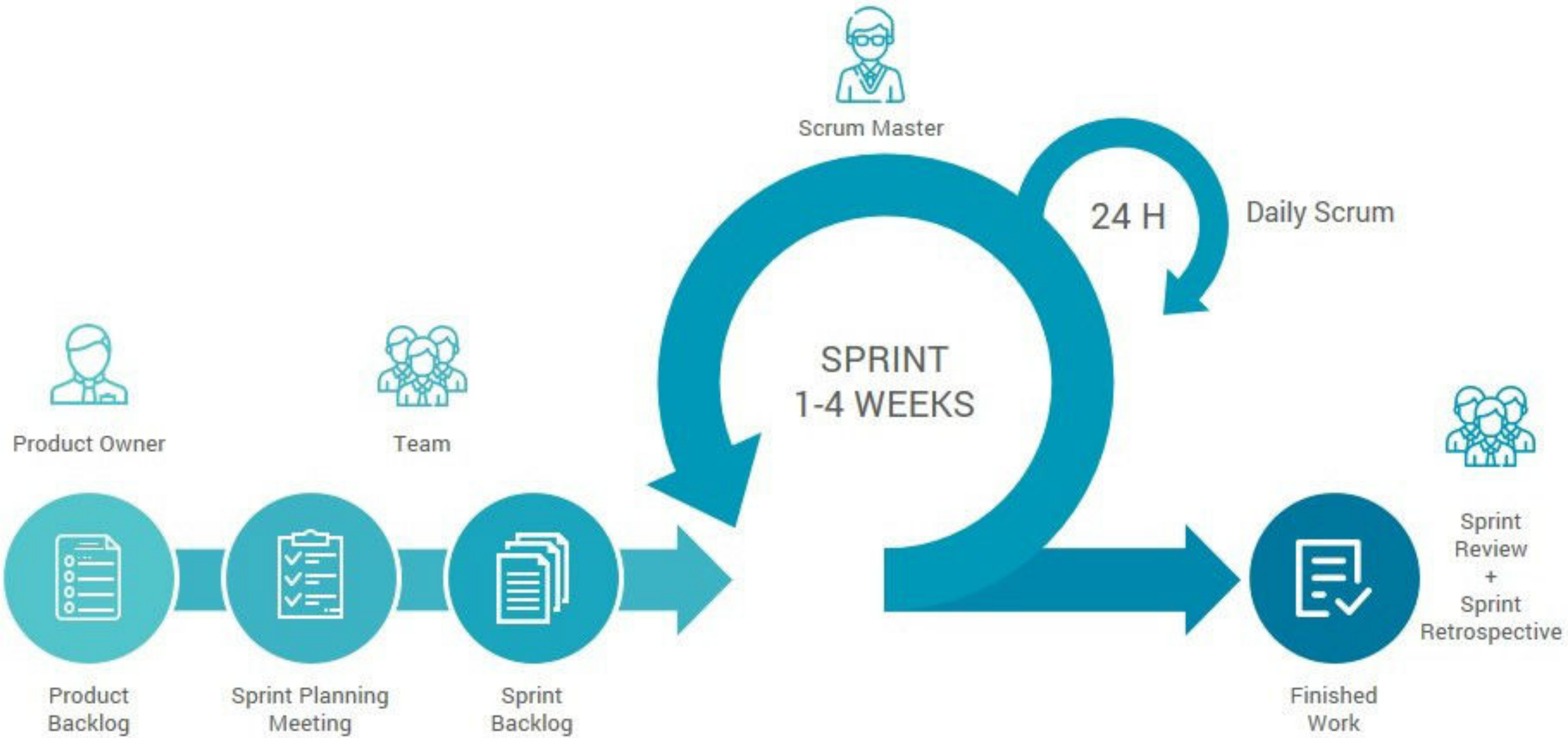
The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.

Sprint Retrospective

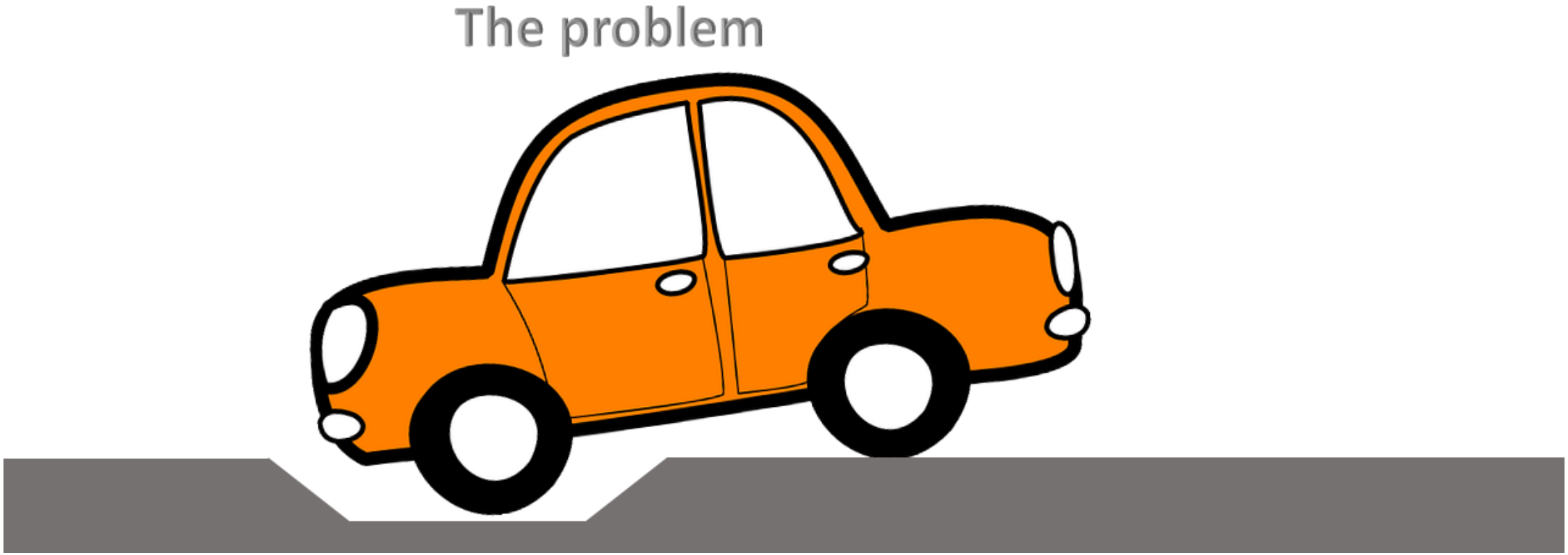
A positive meeting held at the end of each sprint

- Inspect how the last Sprint went with regards to people, relationships, process, and tools;
- Identify and order the major items that went well and potential improvements; and,
- Create a plan for implementing improvements to the way the Scrum Team does its work.

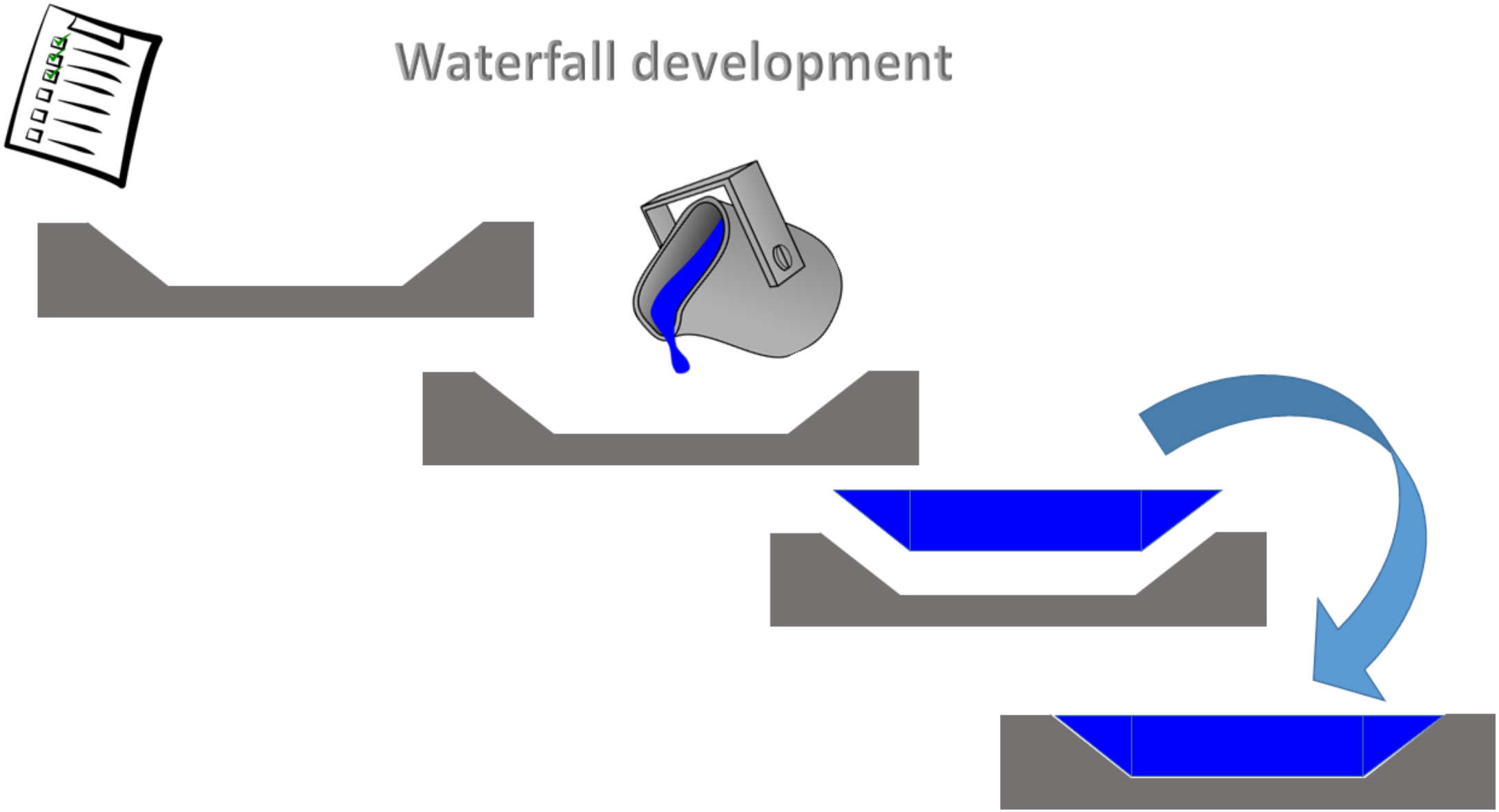
> The Scrum Team should have identified improvements that it will implement in the next Sprint.


Scrum Master
24 H
Daily Scrum
SPRINT
1-4 WEEKS
Product Owner
Team
Sprint
Review
+
Sprint
Retrospective
Product
Backlog
Sprint Planning
Meeting
Sprint
Backlog
Finished
Work

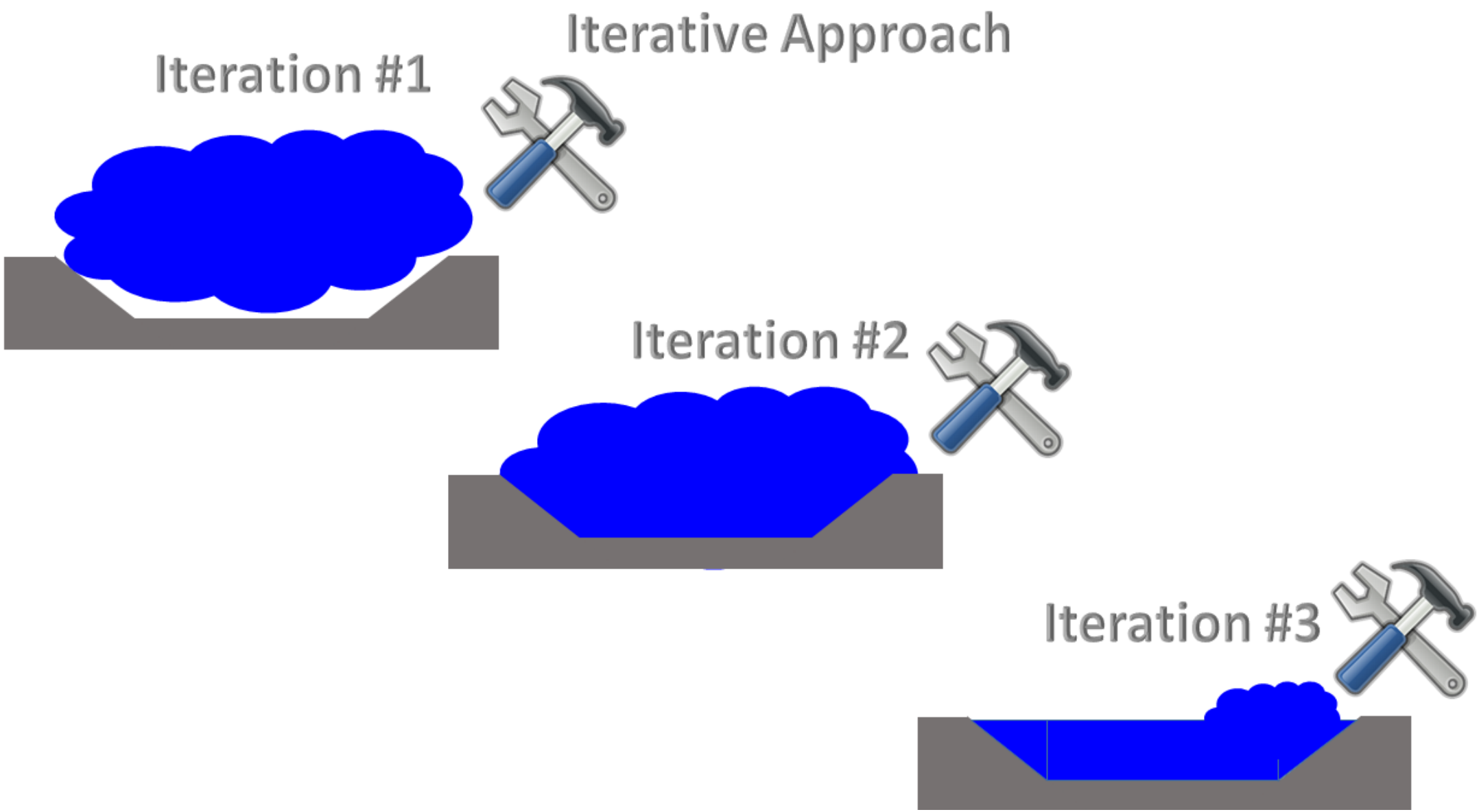
An example



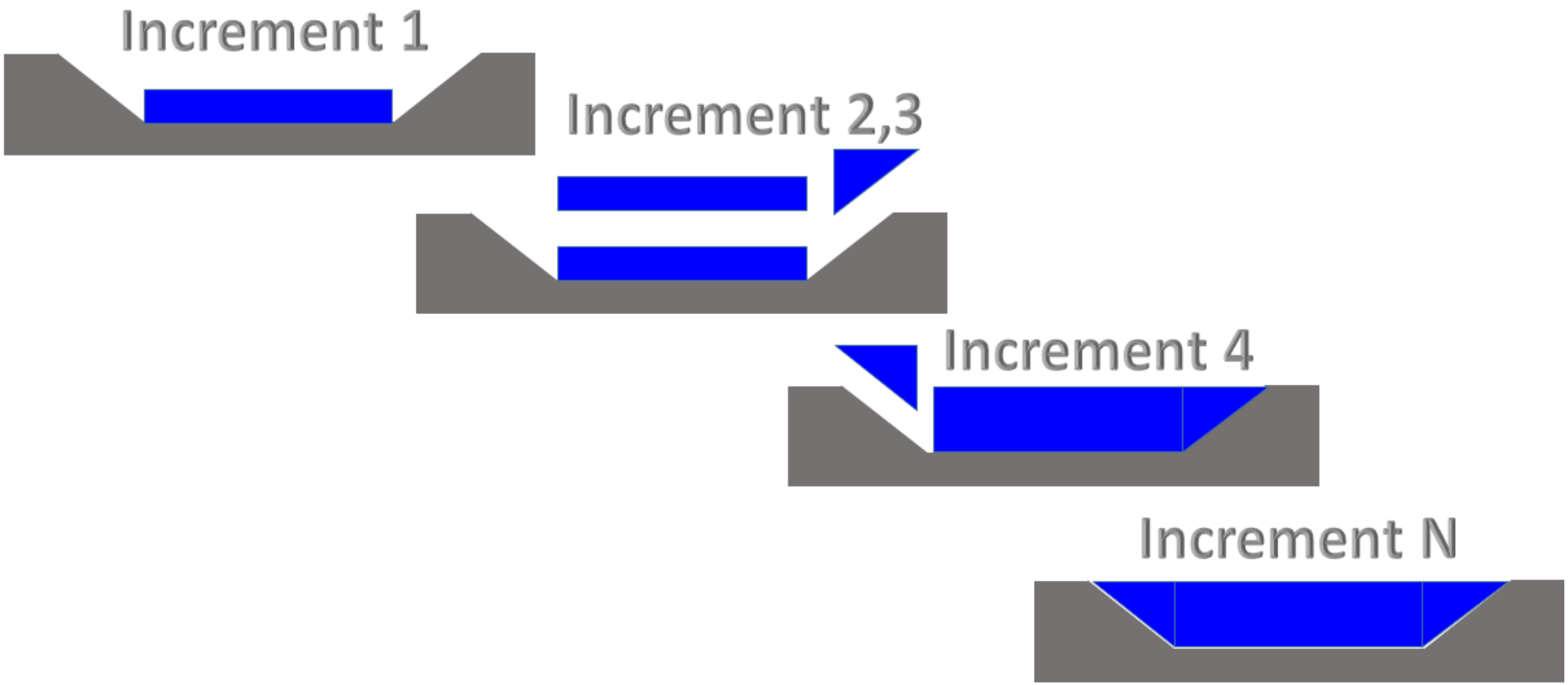
Waterfall development



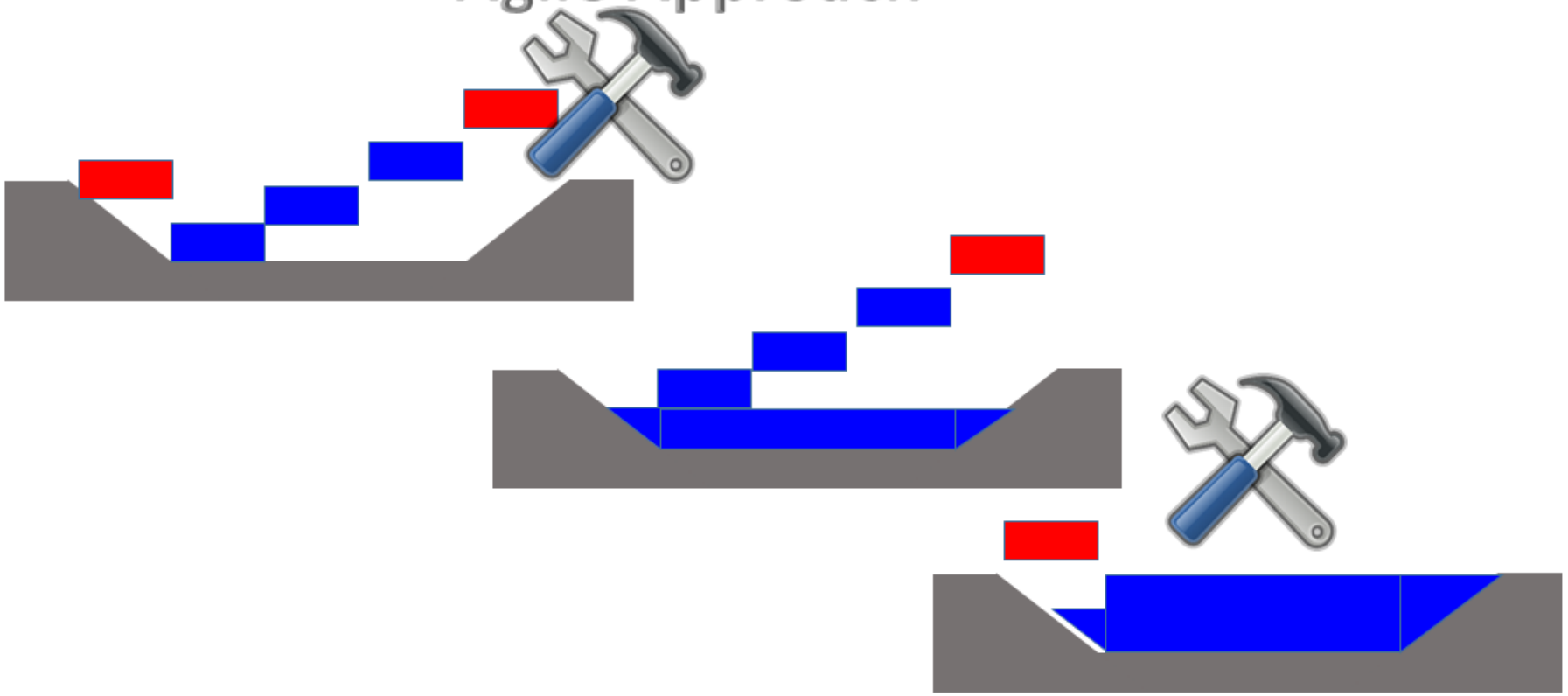
Iterative Approach



Incremental Approach



Agile Approach



Increments and Iterations

Extreme Programming (XP)

Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

https://www.agilealliance.org/glossary/xp Overview https://www.youtube.com/watch?v=hbFOwqYIOcU

When should you consider it?

- Dynamically changing software requirements
- Risks caused by fixed time projects using new technology
- Small, **co-located** extended development team
- The technology you are using allows for **automated** unit and functional **tests**

> Customers, managers, and developers all work closely with one another

Customer

- The person/group that defines and prioritizes features
- The group of business analysts, QA specialists, and marketing specialists working in the same company
- The user representative commissioned by the body of users
- The paying customer

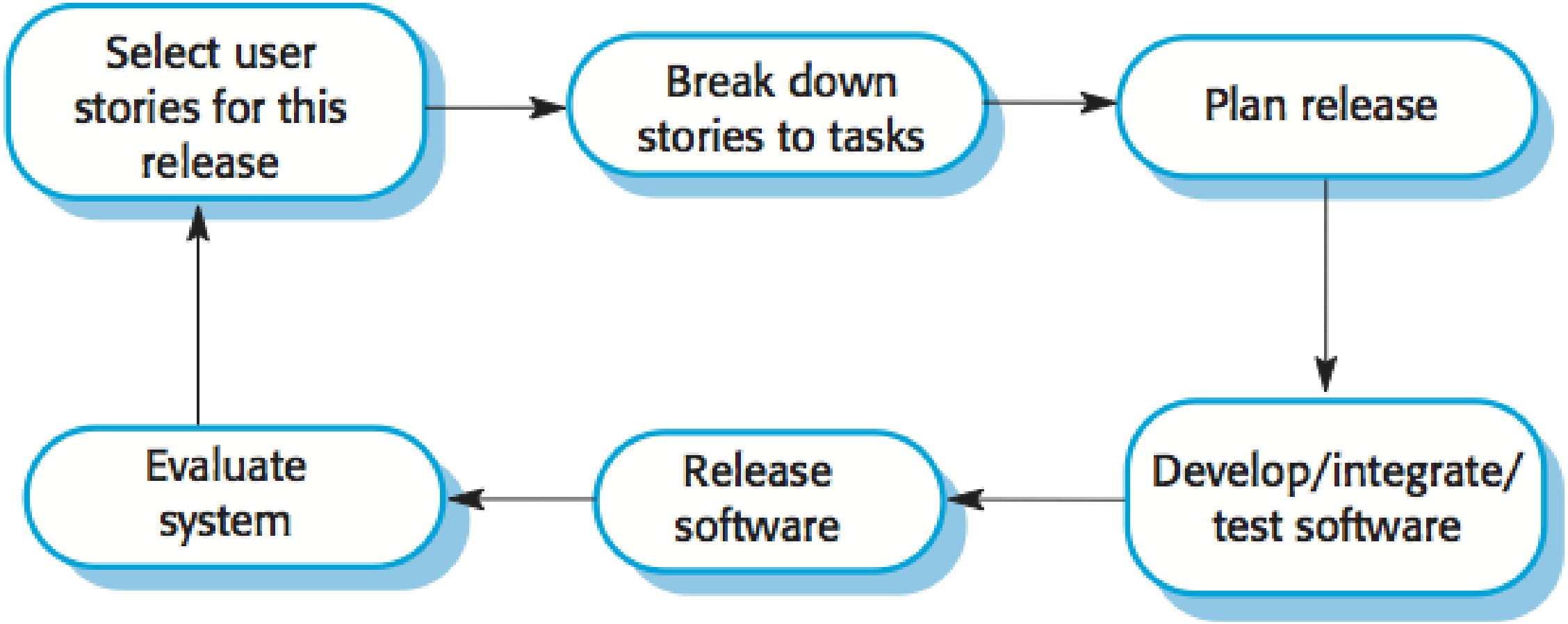
User Stories

- Get the sense of the details of the requirements by talking them over with the customer
- Do not capture the details, but write a few words on an card that will remind both the developer and the customer of the conversation and an estimate (ongoing conversation)
- User stories: tokens of a conversation about a requirement

Development Process

- Developers set the budget for the release (based on previous releases)
- Customer selects a number of stories for the release to fit the budget
- Developers break the stories into tasks
- Customer decides the order in which the stories will be implemented

Development process



Small Releases

- Deliver working software every two weeks
- The system is demonstrated after each development cycle to get feedback

Acceptance Tests and TDD

- Capture details about the user stories
- Written immediately preceding or concurrently with the implementation of the story
- Written as scripts that can be run automatically
- Verify that the system behaves as the customers have specified
- Written by QA specialists or testers during the iteration
- The true requirements document of the project - they describe the details operation of the stories they are implementing

Acceptance Tests

- If an acceptance test passes
  - added to passing acceptance tests and never allowed to fail again
- Passing acceptance tests are run every time the system is built
  - usually several times per day
- If an acceptance test fails
  - the entire build is declared a failure
- Once a requirement is implemented, is it never broken

Continuous Integration

- Code integration run several times per day

Refactoring

- Transformations of the code that improve the structure of the system without changing its behavior
- Tests are run after each transformation
- Done continuously, every hour or every half hour

Example of Refactoring

- Removing duplicate code
- Renaming attributes and methods
- Replacing inline code with calls to methods included in a program library

Pair Programming

- Pairs of programmers working together at the same workstation
  - One coder types the code
  - One coder watches the code being typed, searching for errors and improvements
- Roles change often
- Resulting code is authored by both coders

Pair Programming

- Pair membership changes often
- Every programmer works in two different pairs every day
- Every member of the team should work with every other member of the team during the course of an iteration
- Every member of the team should work on every aspect of an iteration

Collective Ownership

- No programmer is individually responsible for any one particular component or technology
- Nobody has more authority than anybody else

> Read this: https://www.agilealliance.org/glossary/xp

When to Use Agile Method?

- Small or medium-sized software products
- Small or medium-sized teams
- Customer committed to be involved in the development process
- Limited number of external rules and regulations affecting the software
- Requirements change often
- Culture that thrives in chaos

When Not to Use Agile Method?

- Difficult to keep the interest of customers who are involved in the process
- The intense involvements characterizing agile methods may not work for the development team
- Strict contracts required upfront

Questions?