Large Language Models

Introduction to Software Engineering

Riccardo Coppola

Software is eating up the world*

* Marc Andreessen
in Wall Sreet Journal

Software and Economy

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Definition of Software

- A collection of
  - Computer programs,
  - Procedures,
  - Rules,
  - Associated documentation
  - Data
- Example:
  - Requirements document,
  - Project plan,
  - Test plan,
  - Test cases,
  - Build scripts,
  - Deployment scripts,
  - User manuals

Software Engineering

- ***Multi person*** *construction of* ***multi version*** *software*

- Multi person:
  - Issues in communication: misunderstandings, language gaps
  - Issues in coordination
- Multi version:
  - Issue in maintenance over many years

Software Types

- Embedded in non-software product
  - Car, washing machine, ..
  - Production line (Industry 4.0)
- Stand alone
  - Office suites, social networks, ..
- Embedded in enterprises
  - Information systems

Software Criticality

- Safety critical: harms people or environment
  - Self-driving car
  - (usually embedded software)
- Mission critical: harms business
  - Banking, finance, retail
  - (usually embedded in enterprises)
- Other levels
  - Depending on the characteristics of the software and how it is used in its environment

Warning

- Software is not free
- Software changes (and it is not easy to change)
- Software is not perfect
- Software is complex

Process and Product

Classical Engineering vs. Software Engineering

- Design the product → Software product
- Design the factory → Software Process
- Manufacture the product → Deployment and Delivery
- Maintain the product → Evolution and Maintenance

Software Product Properties

- Functional properties
  - characteristics that describe the core capabilities and behaviors a software system must exhibit to meet its intended purpose.
  - By accurately defining and implementing these properties, developers ensure that the software provides the intended value.

Software Product Properties

- Non functional properties
  - Usability
    - Effort needed to learn using the product (installation, day to day usage)
    - Satisfaction expressed by the user
    - Existence of functions needed by the user
  - Efficiency
    - For a given function in a given context: response time
    - For a given function / for a complete product:
      - Memory
      - CPU
      - Bandwidth
      - energy used

Software Product Properties

- Non functional properties
  - Reliability / availability
    - Defects visible by end user per time period / Probability of defect over a time period
    - Percentage of time the product is / is not available to end user
  - Maintainability
    - Effort (person hours) needed to add /modify / cancel a software function
    - Effort to fix a defect
    - Effort to deploy on a different platform (DB, OS, ..)

Software Product Properties

- Non functional properties
  - Security
    - Protection from malicious access
    - Access only to authorized users
    - Sharing of data
  - Safety
    - Absence of harm to persons
    - Absence of hazardous situations for persons
  - Dependability
    - Safety + security + reliability

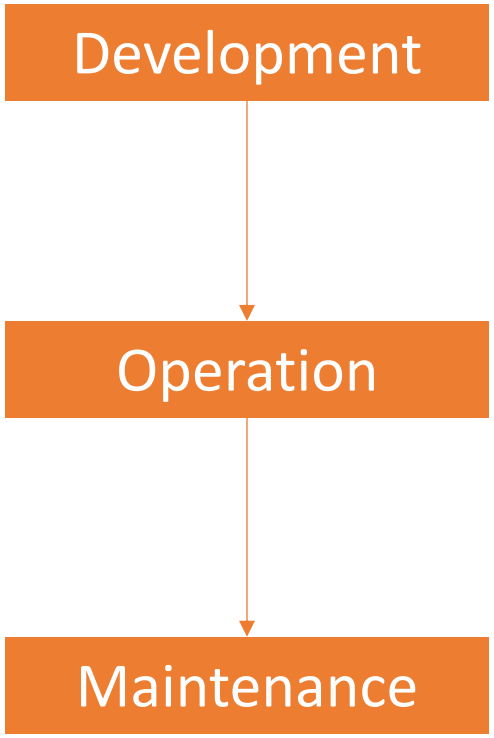
Software Product Properties

- Non functional properties
  - Are difficult to engineer
  - Are often forgotten
  - Make the difference between competing products

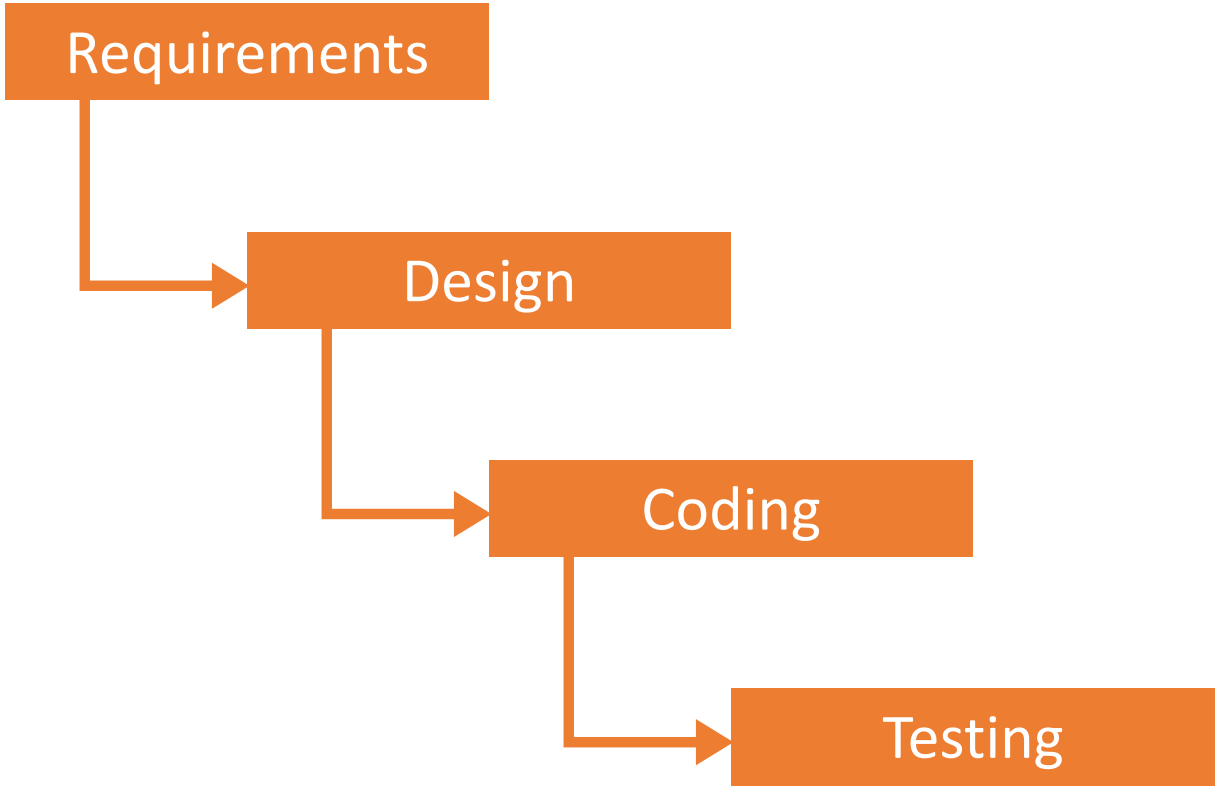
Taxonomy of non-functional requirements

Source: ISO 25010

The traditional software process



Software process – development phase



Software Process Properties

- Cost
  - Currency (€, $, …)
- Effort
  - Person hours
- Punctuality
  - Promised delivery date vs actual delivery date
- Conformance (to standards, norms)

Software Engineering Laws

[Endres Rombach, 2005]

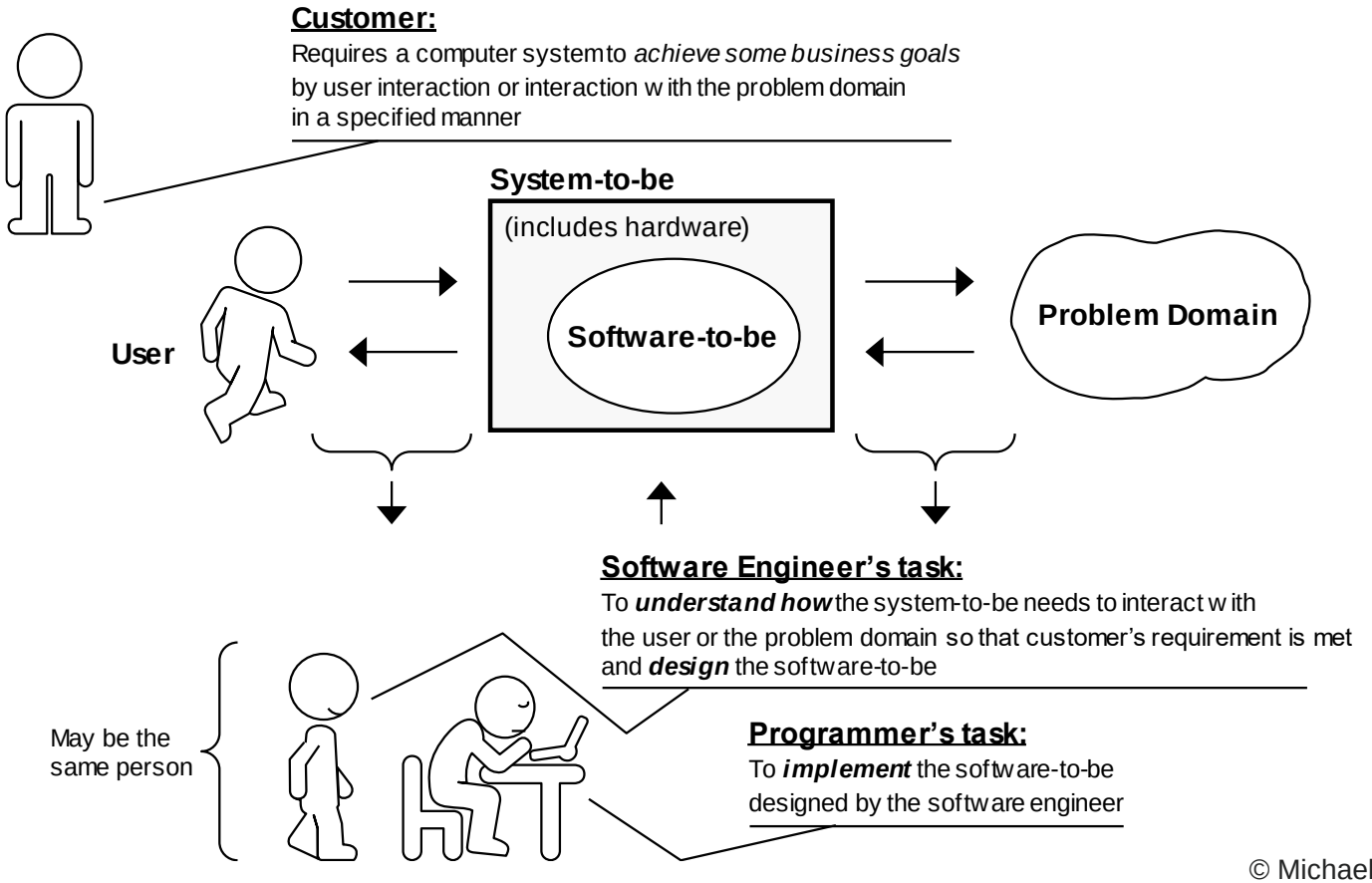
- Requirements deficiencies are the prime source of project failures
- Requirements and design cause the majority of defects
- Defects from requirements and design are the more expensive to fix

Software Engineering Laws

- Modularity, hierarchical structures allow to manage complexity
- Reuse guarantees higher quality and lower cost

Software engineering Laws

- Good designs require deep application domain knowledge



Software Engineering Laws

- Testing can show the presence of defects, not their absence
- A developer is unsuited to test his/her code

Software Engineering Laws

- A system that is used will be changed
- An evolving system will increase its complexity, unless work is done to reduce it
  - Architecture erosion
  - Requirements creep
  - Refactoring

Software Engineering Laws

- The process should be adapted to the project

The Software Process

No Software Process: Cowboy Coding

Cowboy coding is coding where the developer has free rein over the process. The cowboy coder has complete control over the project schedule; the languages, algorithms, tools, and frameworks to use; and the coding style to follow.

Advantages: faster than engineering software

Disadvantages: Lack of testing, quality, maintainability of code

Goals

Produce software

- documents, data, code

with defined, predictable process properties

- cost, duration

and product properties

- functionality, reliability, ..

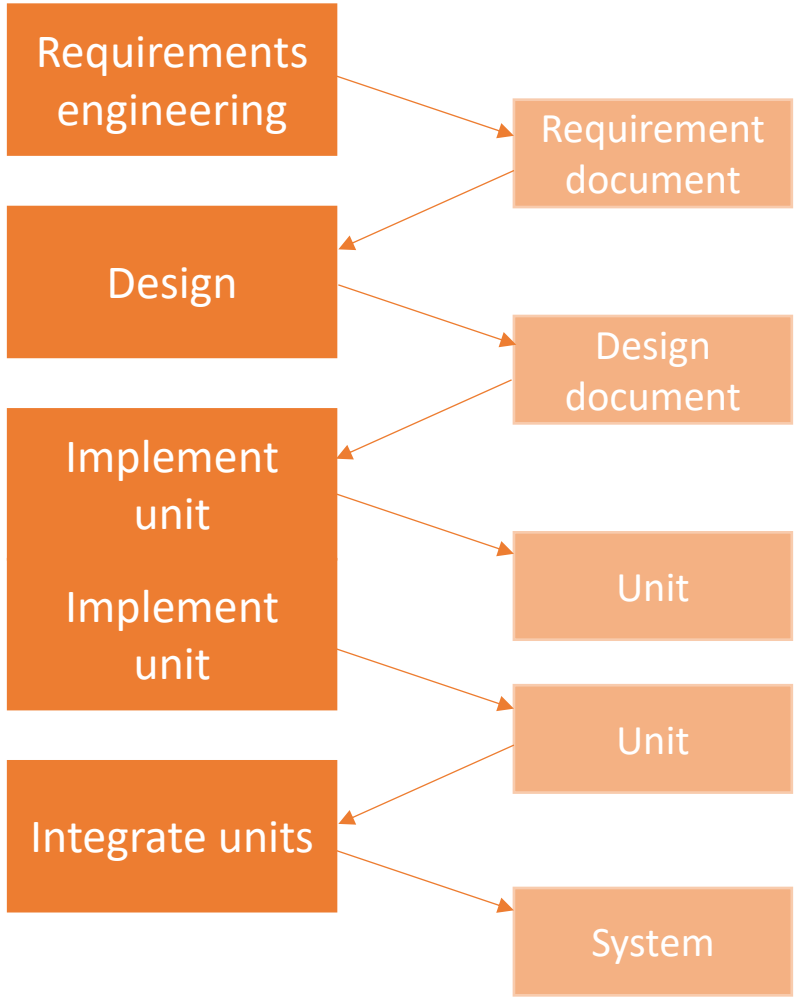
The production activities

- Requirement engineering
  - What the software should do
- Architecture and design
  - Which units and how organized
- Implementation
  - Write source code, (executable code)
- Integrate units

The production activities

- Logically, each activity depends on the previous one(s)
  - To design, one must know the requirements
  - To implement, one must know the design and the requirements
- First approach is to do these activities in sequence
  - See waterfall model later
- In practice feedbacks and recycles must be provided
- Requirements and design are written down in documents

The production activities



The production activities

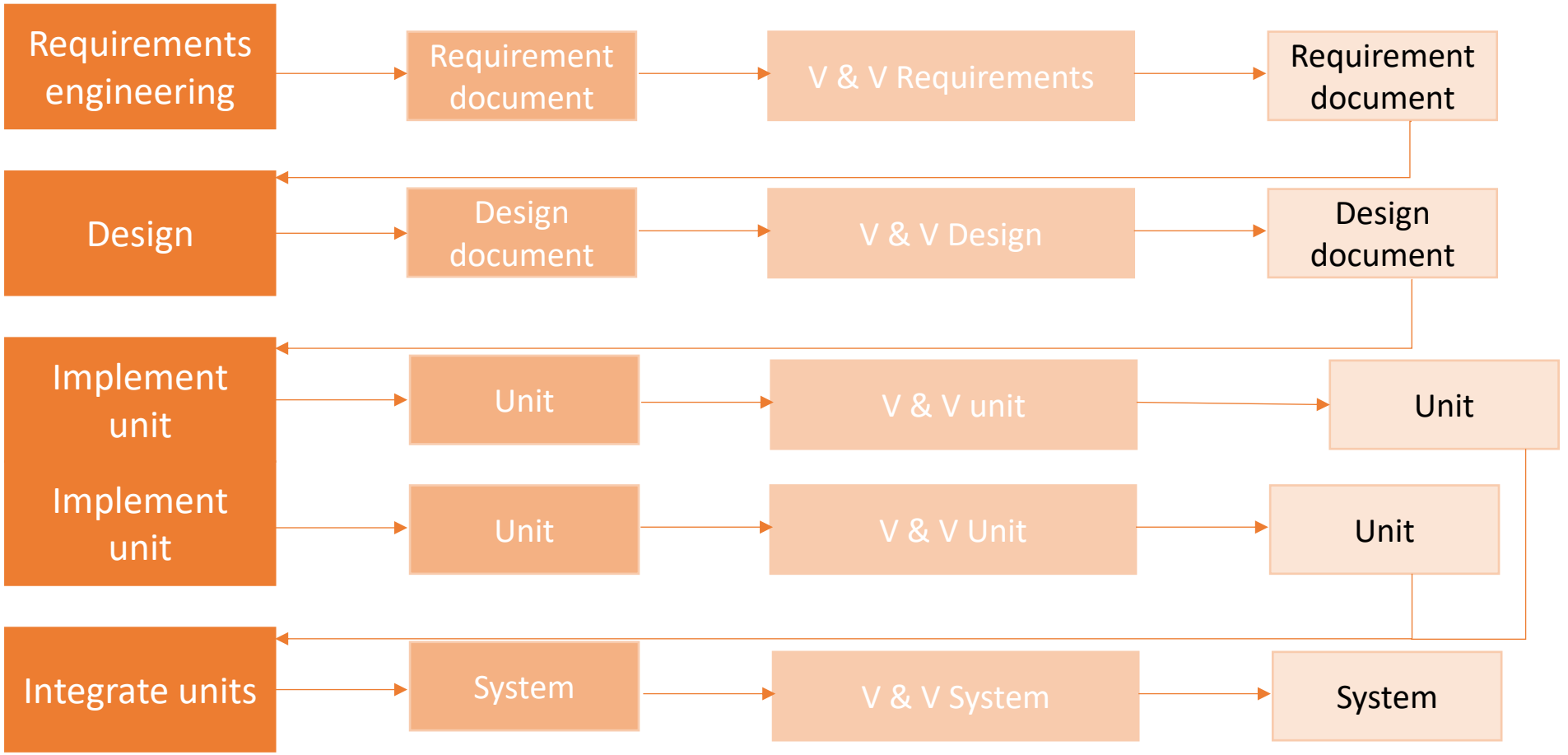
- Ok, we did it
  - Does it work?
  - Is it doing what it should do?

- Or
  - Did we understand the requirements correctly?
  - Did we implement the requirements correctly?

The Validation & Verification activities

- These activities are usually called V & V activities
- Control that the requirements are correct
  - Externally: did we understand what the customer/user wants?
  - Internally: is the document consistent?
- Control that the design is correct
  - Externally: is the design capable of supporting the requirements
  - Internally: is the design consistent?
- Control that the code is correct
  - Externally: is the code capable of supporting the requirements and the design?
  - Internally: is the code consistent (syntactic checks)

The Validation & Verification activities



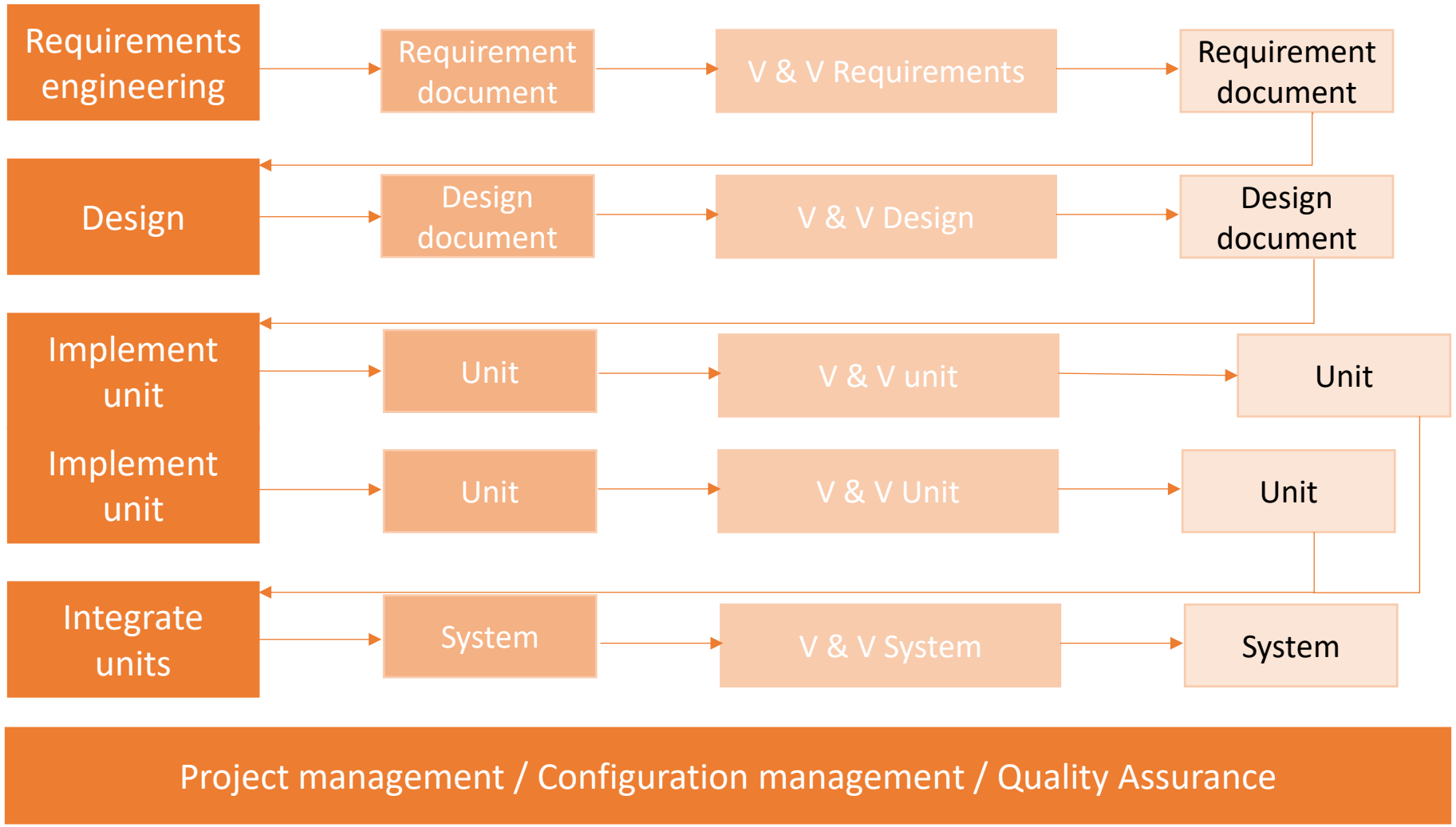
The management activities

- Well, seems a lot of work
  - Who does what, when?
  - With what resources?
  - How much will it cost, when will we finish?
  - Where are the documents and units? Who can modify what?
  - Are we doing it state of the art?

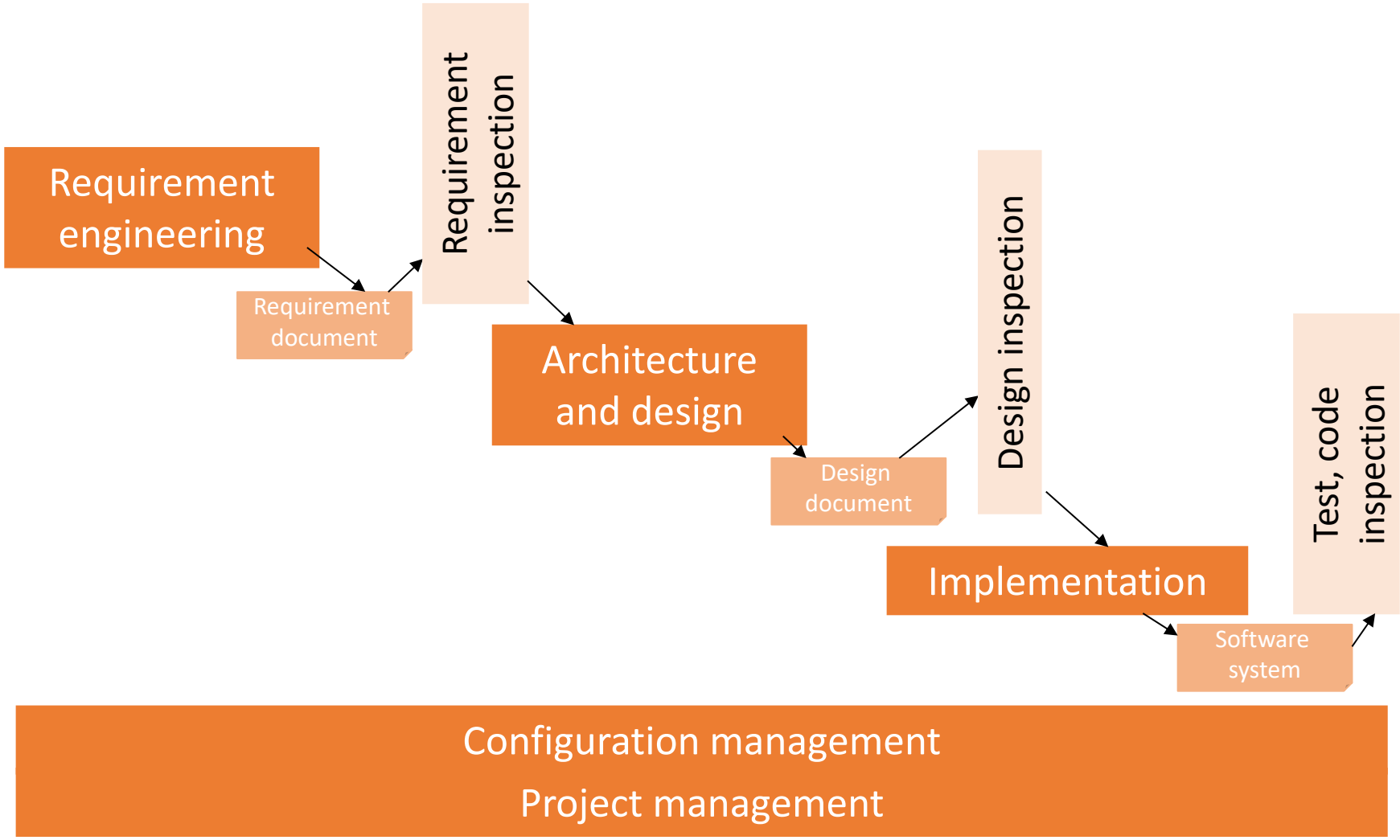
The management activities

- Project management
  - Assign work and monitor progress
  - Estimate and control budget
- Configuration management
  - Identify, store documents and units
  - Keep track of relationships and history
- Quality assurance
  - Define quality goals
  - Define how work will be done
  - Control results

The whole picture



The whole picture



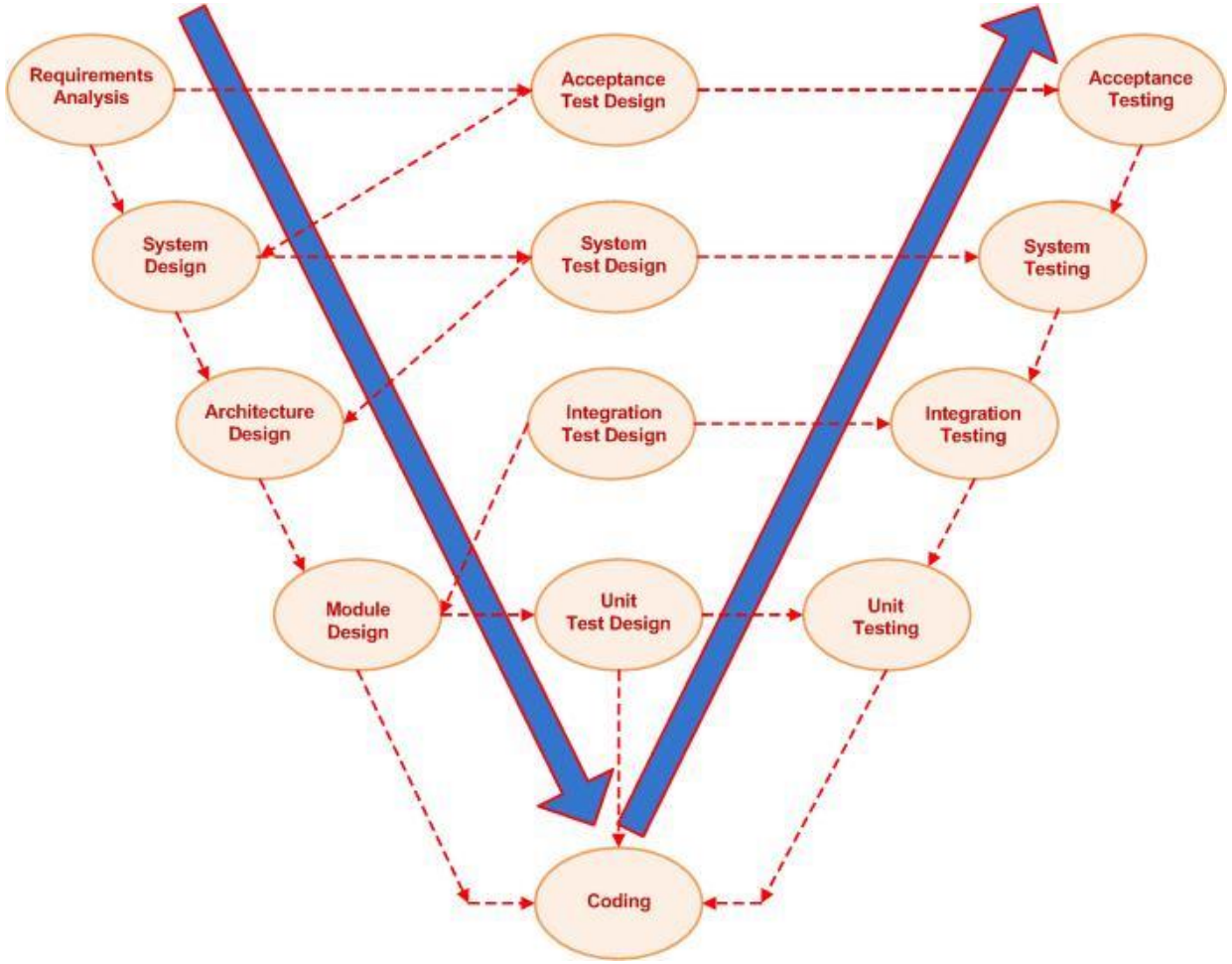
The Waterfall Model

- A Linear, sequential approach to the software development lifecycle.

- Advantages:
  - Uses a clear structure
  - Determines the end goals early
  - Transfers information well

- Disadvantages:
  - Makes change difficult
  - Excludes the client and/or end user
  - Delays testing until after completion

The V-Model


Requirements Analysis
Acceptance Test Design
Acceptance Testing
System Design
System Test Design
System Testing
Architecture Design
Integration Test Design
Integration Testing
Module Design
Unit Test Design
Unit Testing
Coding

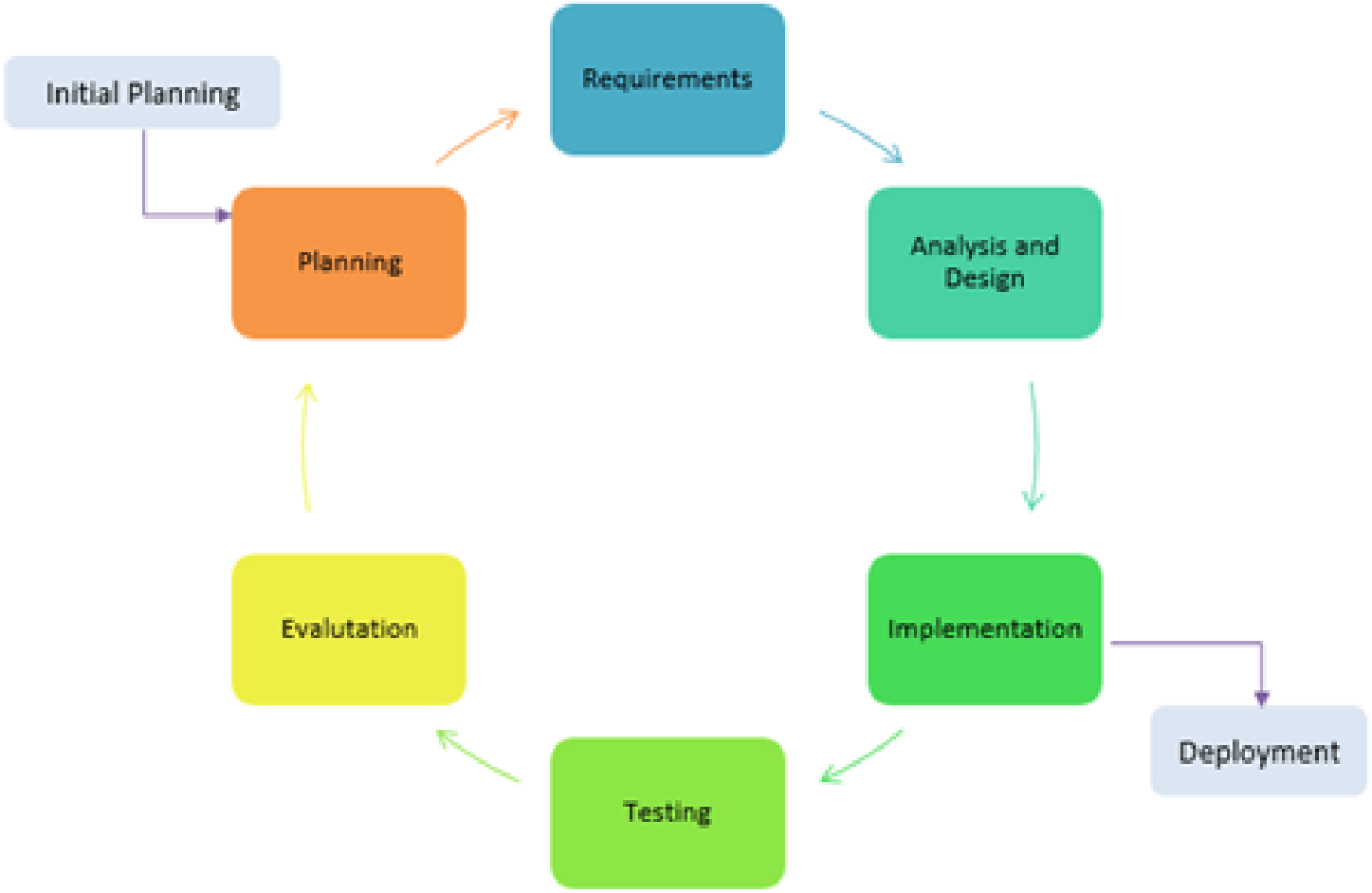
The V-Model

- V-Model is focused on Verification & Validation and software. The V-Model mandates – for every stage in the development cycle – an associated testing phase is considered
- The testing activities start immediately in any phase (the tests are first prepared, then executed)

- Advantages:
  - More control and more quality of the software

- Disadvantages:
  - More expensive than waterfall
  - Still, design only happens once

The Iterative model



The Iterative Model

- An iterative life cycle model does not start with a full specification of requirements. In this model, the development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements.
- Moreover, in iterative model, the iterative process starts with a simple implementation of a small set of the software requirements, which iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed. Each release of Iterative Model is developed in a specific and fixed time period, which is called iteration.

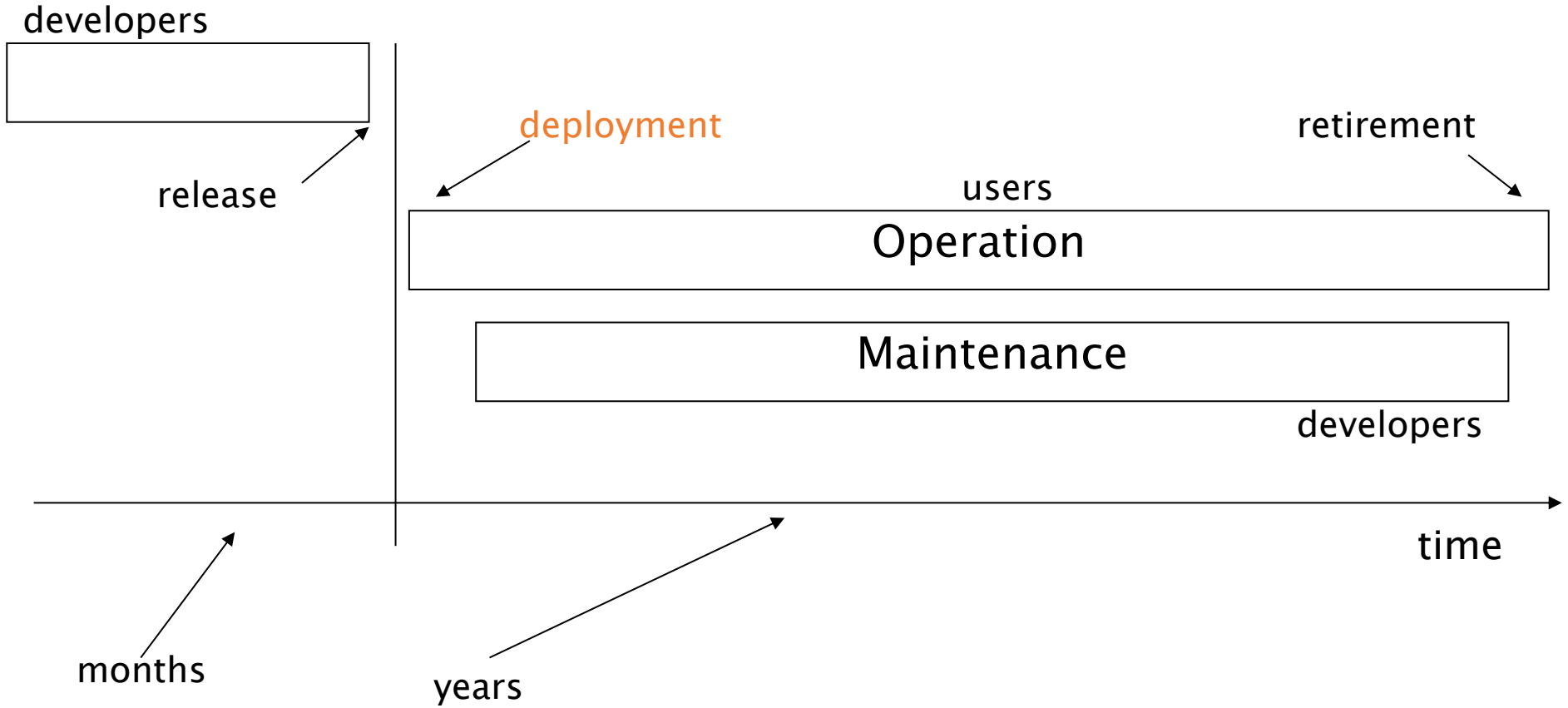
- Advantages:
  - It is easily adaptable to the ever changing needs of the project as well as the client

- Disadvantages:
  - It is not suitable for smaller projects
  - Defining increments may require definition of the complete system

Beyond code development

- Development is only the first part of the game
  - Operate the software
    - Deployment, operation
  - Modify the software
    - Maintenance
  - End up
    - retirement

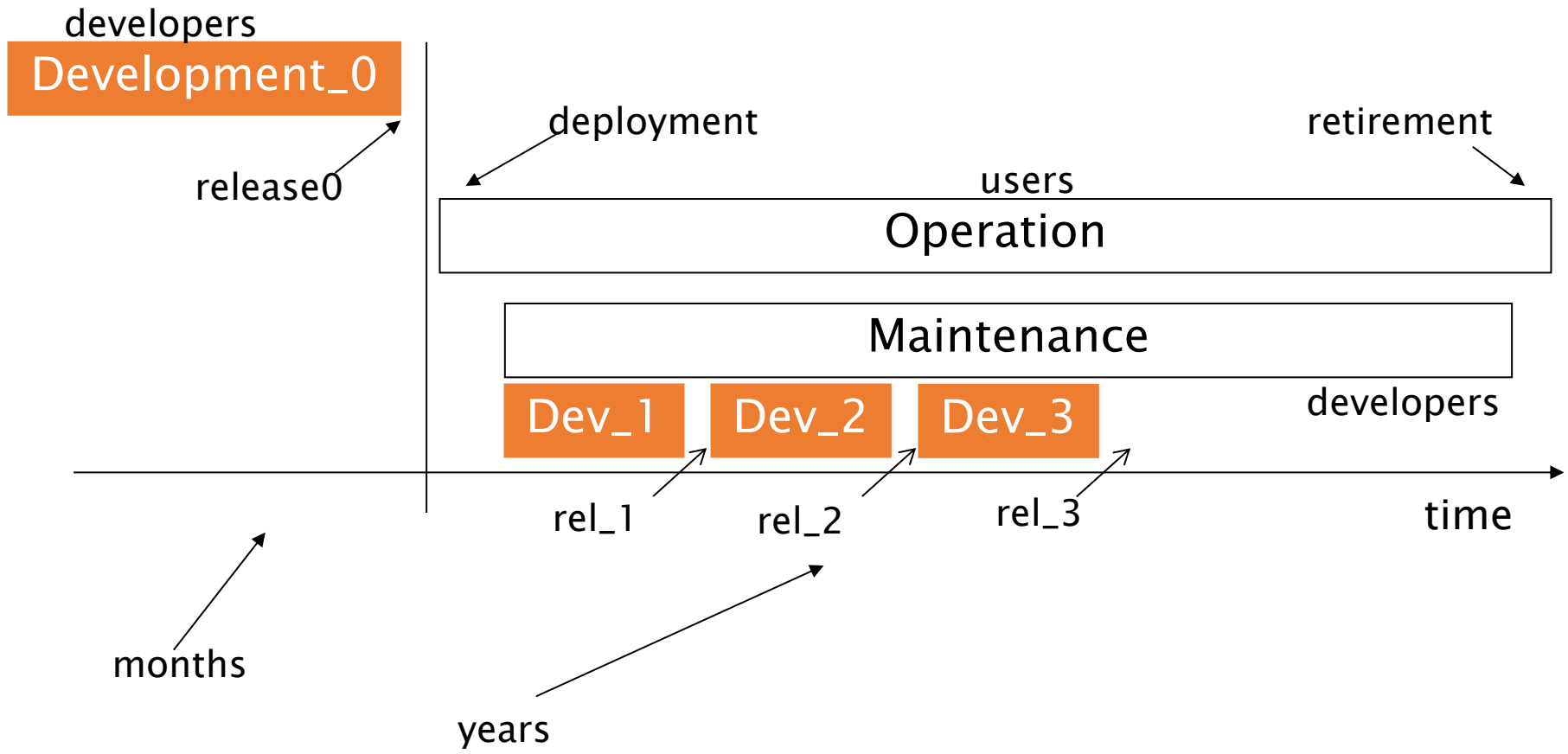
Beyond code development



Maintenance

- Can be seen as a sequence of developments
- First development usually longer
- Next developments constrained by previous ones and related choices
  - If dev_0 chooses Java, next developments are in Java
  - If dev_0 chooses client server model, next developments keep C/S

Maintenance



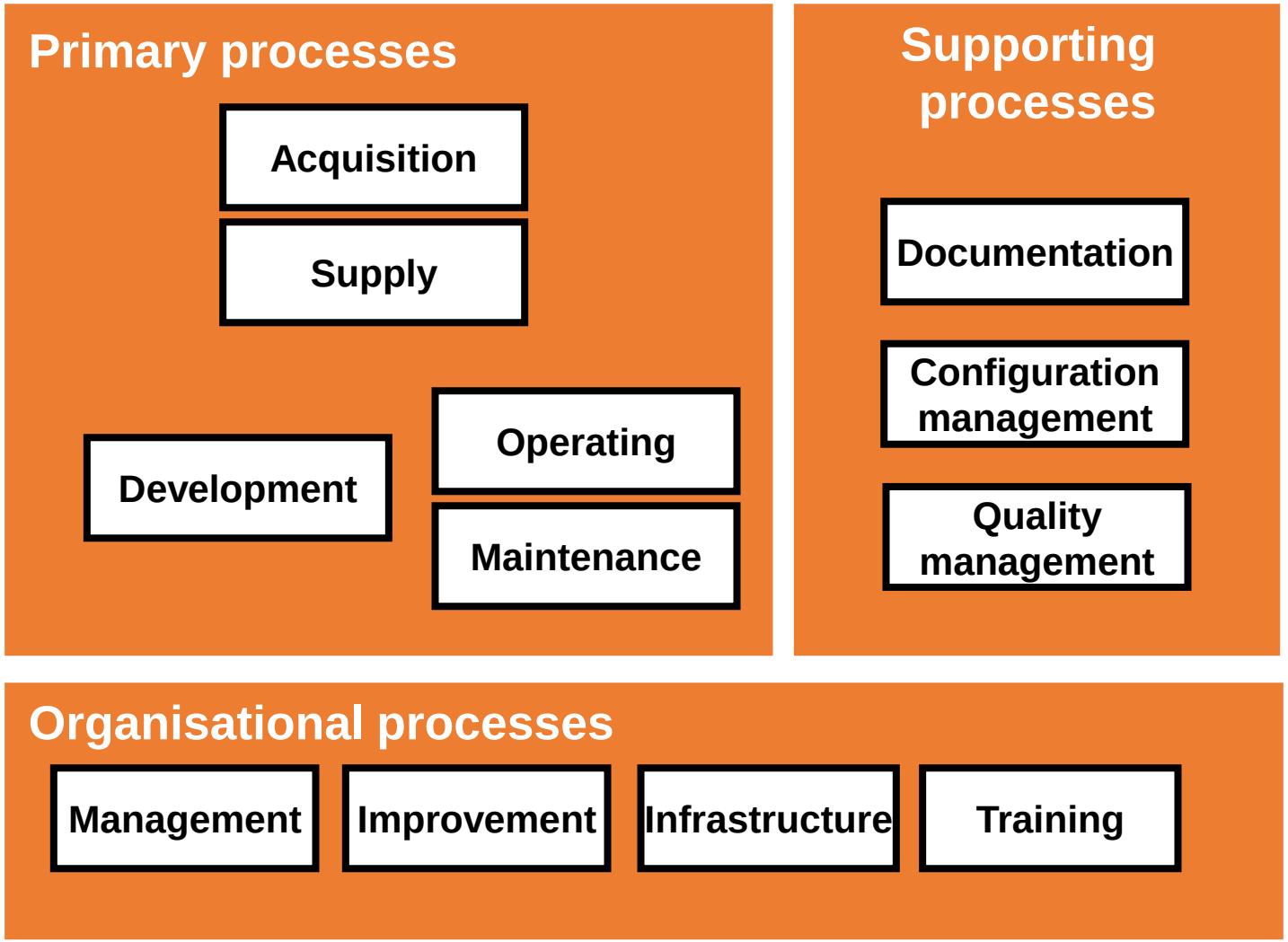
Maintenance

- Development and maintenance do the same activities (requirement, design, etc)
  - But in maintenance an activity is constrained by what has been done before
  - After years, the constraints are so many that changes become impossible

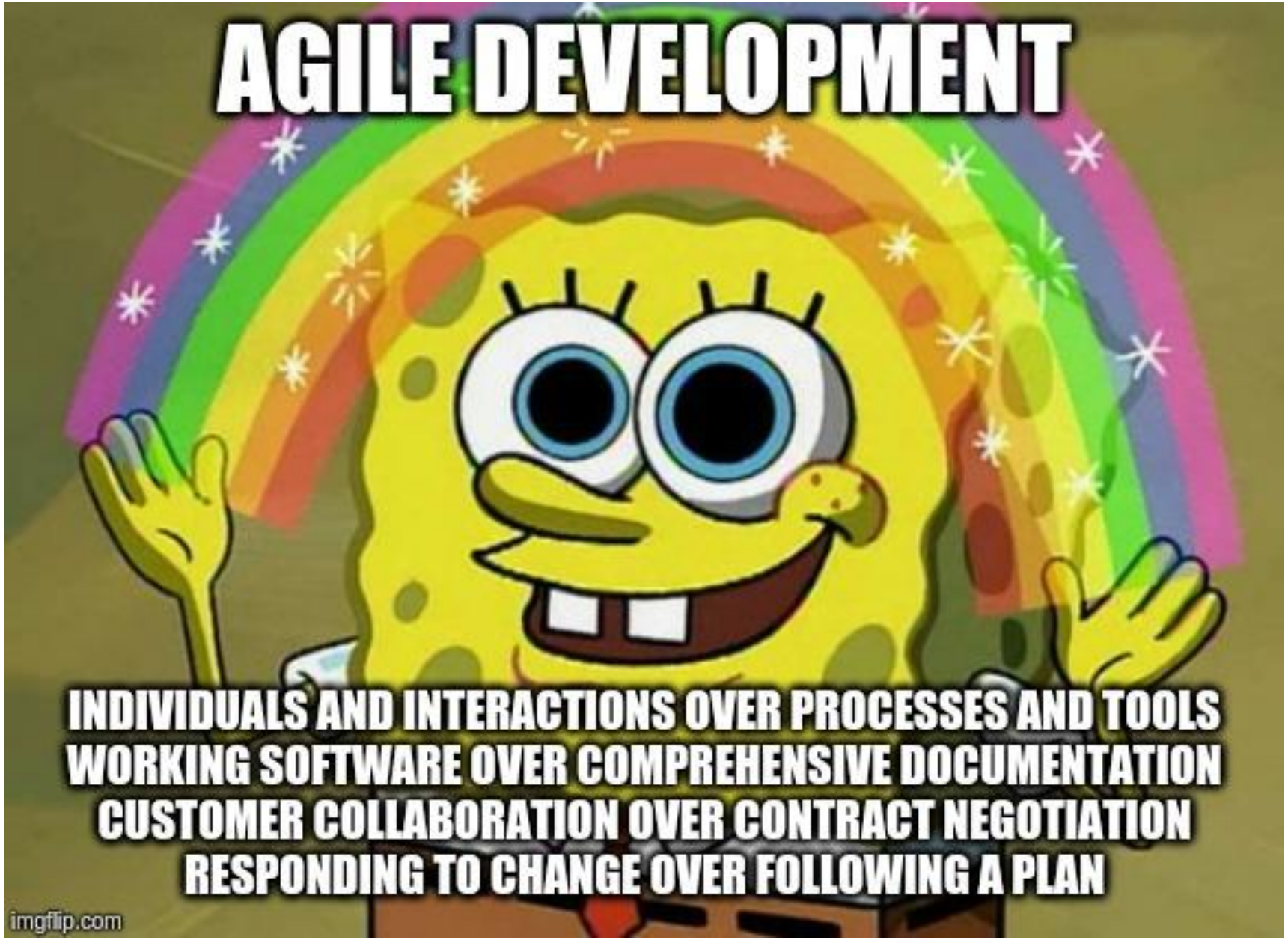
Maintenance

- Development_0
  - Req_0 developed from scratch
  - Design_0 developed from req_0
  - Impl_0 developed from design_0
- Development_1
  - Req_1 from Req_0 (and Des_0, Impl_0)
  - Des_1 from Req_1
  - Impl_1 from Des_1

ISO / IEC 12207



Agile Software Engineering



(Beck et al., 2001)


AGILE DEVELOPMENT
BECAUSE WHO NEEDS A PLAN

The Agile Manifesto - Principles

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

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

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

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

The Agile Manifesto - Principles

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

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

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

The Agile Manifesto - Principles

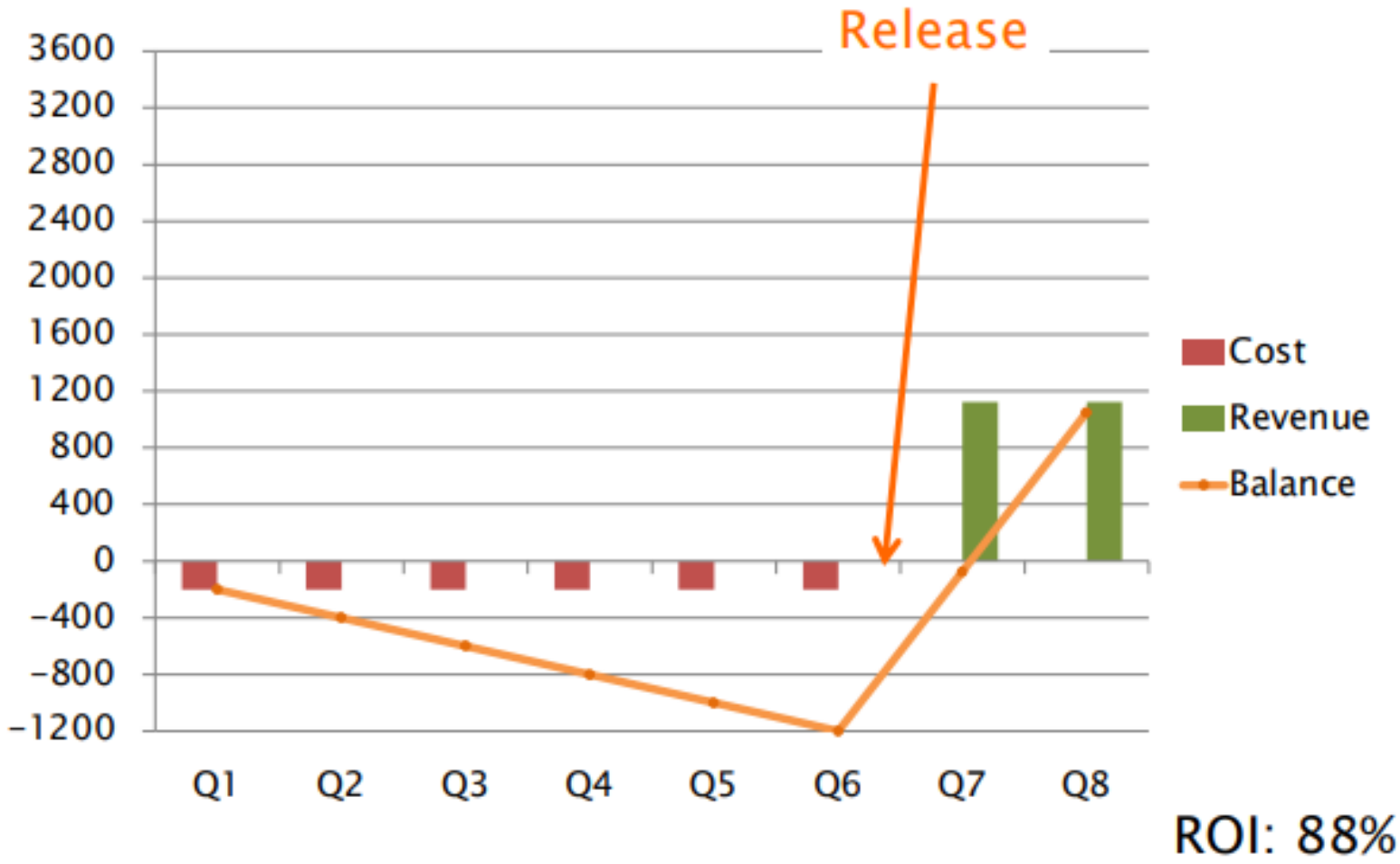
9. Continuous attention to technical excellence and good design enhances agility.

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

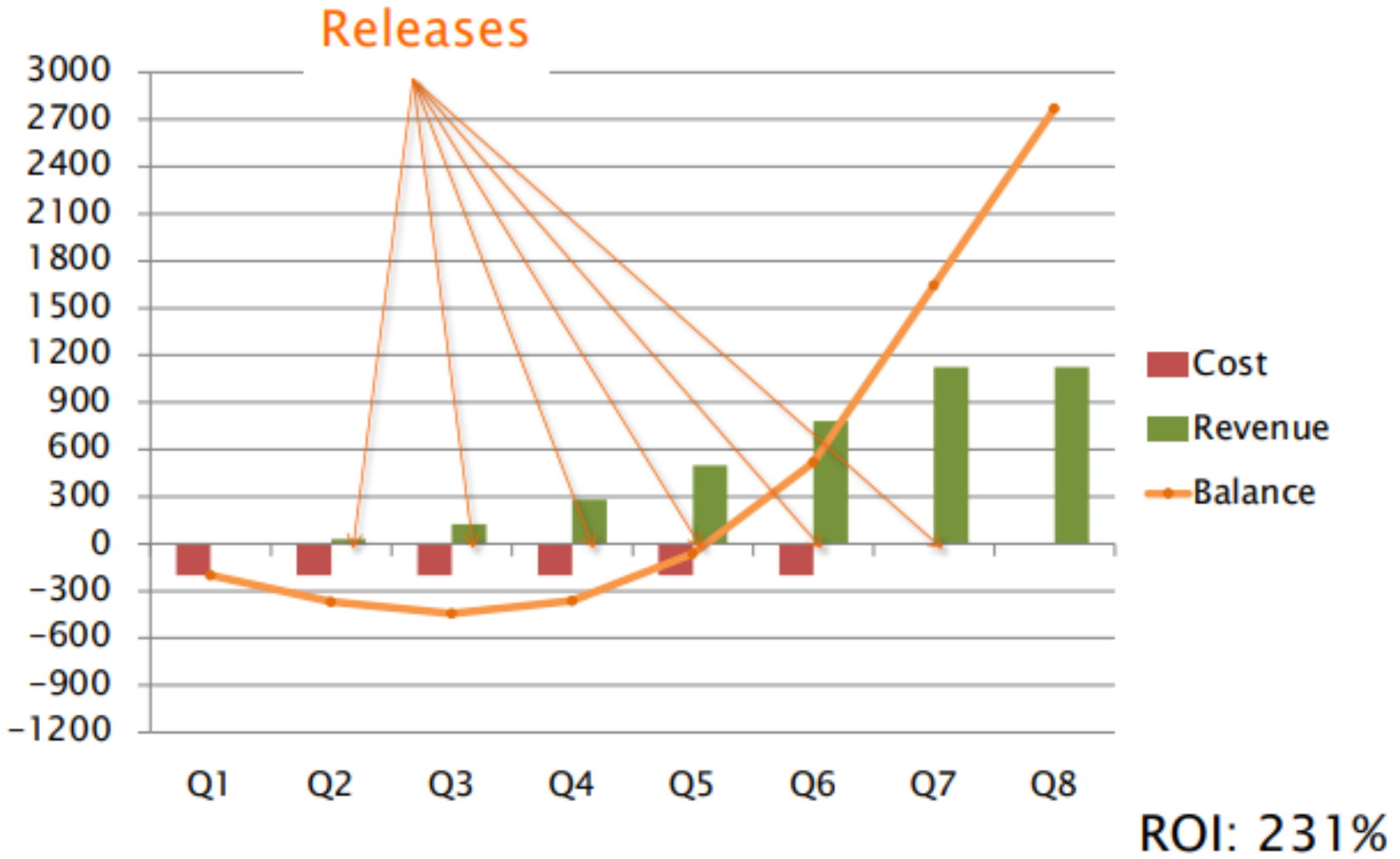
11. The best architectures, requirements, and designs emerge from self-organizing teams.

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

...in practice? (€$)


Release
3600
3200
2800
2400
2000
1600
1200
800
400
0
-400
-800
-1200
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Cost
Revenue
Balance
ROI: 88%

...in practice? (€$)

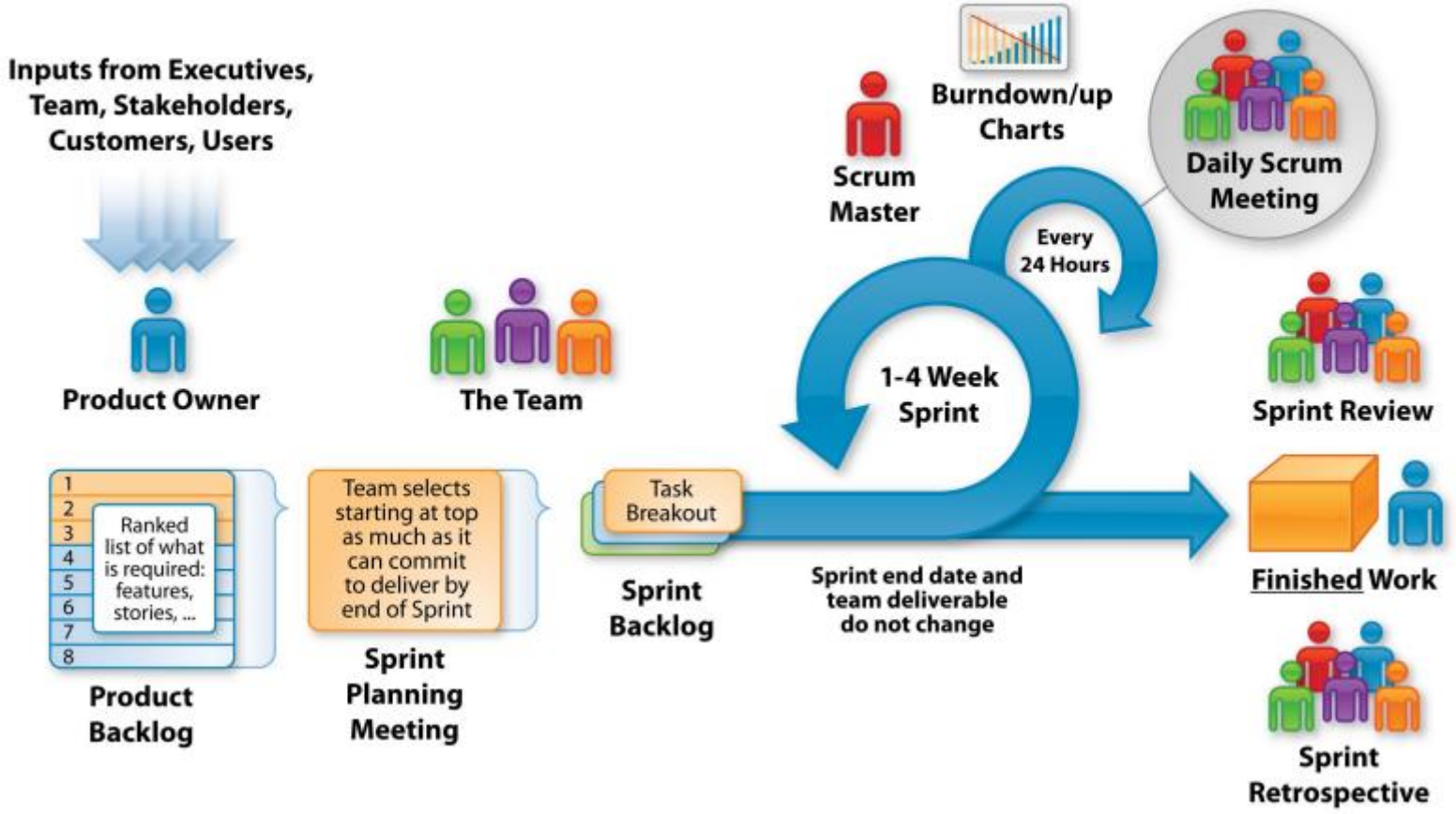


ROI: 231%

From manifesto to practice

- Tight collaboration between developers and stakeholder over the entire course of the project
- Self-organizing teams
- Software SCRUM is the most used methodology
- Other methodologies like Kanban and eXtreme Programming are less used

SCRUM

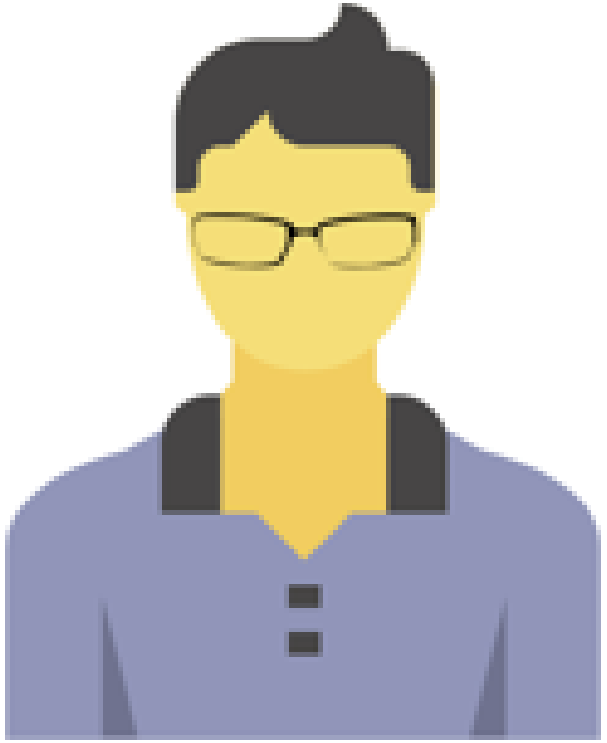

Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective

SCRUM: Essential elements

- User stories
- Estimation
- Planning
- Tight Feedback Loops
- Reflection

SCRUM Roles

- The Product Owner
  - Controls the priority order of items in the team's backlog
  - Works closely with the stakeholders in order to deliver the maximum business value (deciding what needs to be built and when)
  - Makes sure that the needs of the customers and the end-user are understood by the team
  - Keeps a vision of the product (who the product is built for, why they need it, how they will use it)

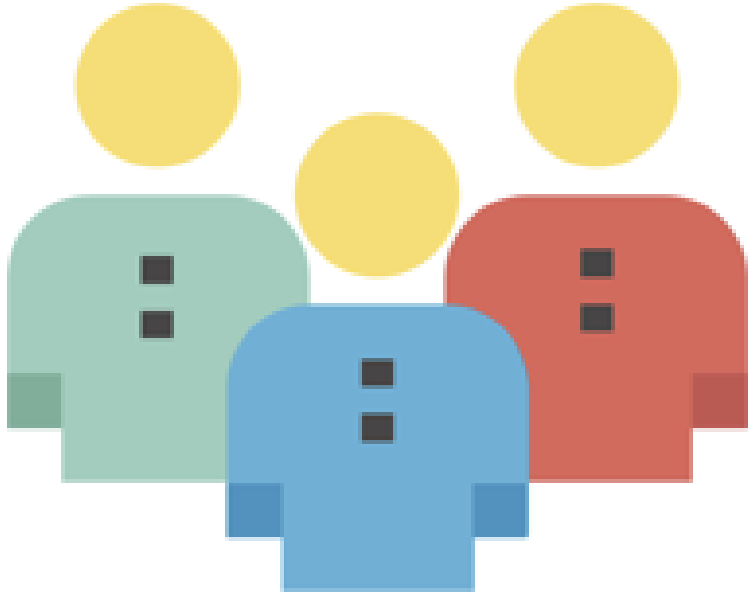


SCRUM Roles

- The Scrum Master
  - Acts as a coach: the goal is to produce a self-organizing team
  - They are a facilitator, not a boss
  - Main task is to remove both external or internal impediments for the team

SCRUM Roles

- Team members
  - They cooperate to achieve the final goal
  - The team members self-organize, in terms of tools used, techniques and task assignment
  - The team members estimate the effort required to implement the features
  - The teams are typically sized 5 to 9



SCRUM Sprint

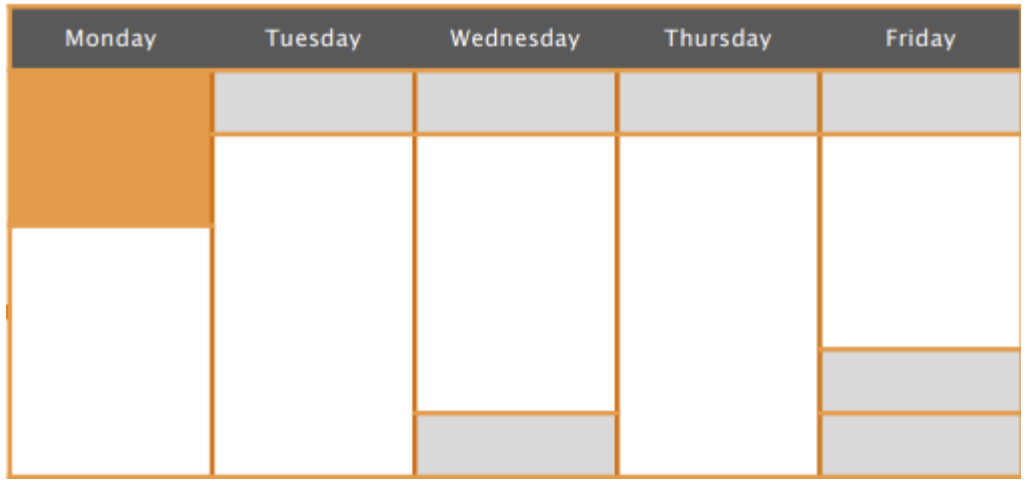
- The Sprint is the basic iteration in the Scrum approach
- It produces a piece of working software to be demonstrated and reviewed at the end of the sprint
- It is from 1 to 4 weeks long

SCRUM Sprint

Monday	Tuesday	Wednesday	Thursday	Friday
Sprint Planning (2h)	Scrum (15m)	Scrum (15m)	Scrum (15m)	Scrum (15m)
				Sprint Review (1/2 h)
		Story Time (1h)		Retrospective (1h)

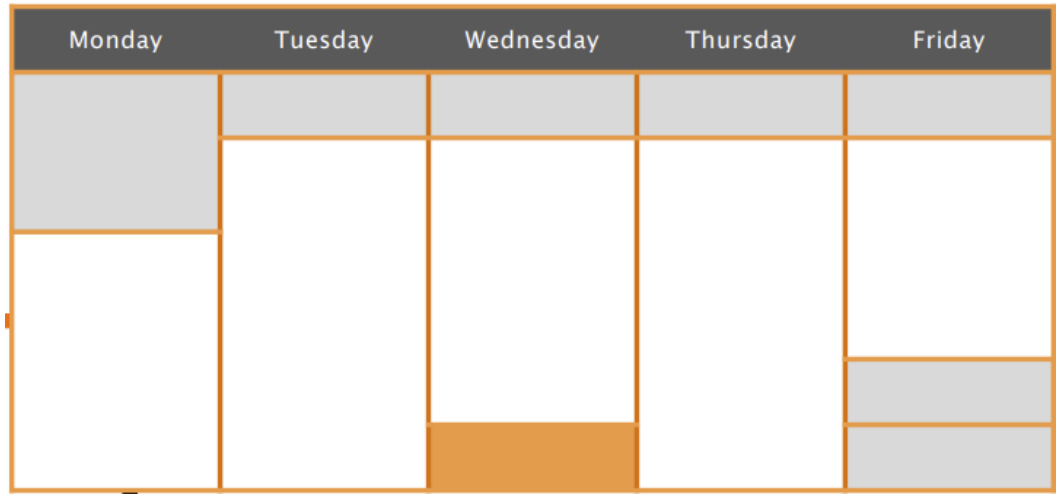
Sprint Planning

- What will we do?
  - Set of committed stories
  - Product owner proposes story
  - Team members decide whether commit
- How will we do it?
  - Decompose stories into tasks
  - May trigger renegotiation of stories
  - Max effort per task: half-day
- Creation of the sprint backlog:
  - List of stories with related task
  - Estimations: task hours, task points, task count



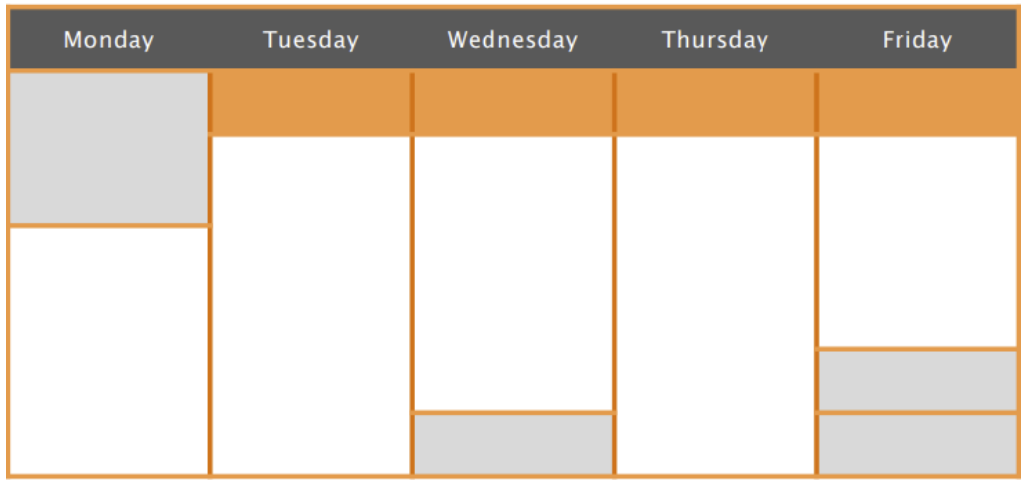
Story Time

- Backlog refinement
  - Upcoming stories are reviewed
  - Sizing (estimating) future stories
  - Clarification of requirements
  - Splitting stories
- Goal:
  - Start next sprint with a set of small,
  - well-understood, well-sized stories



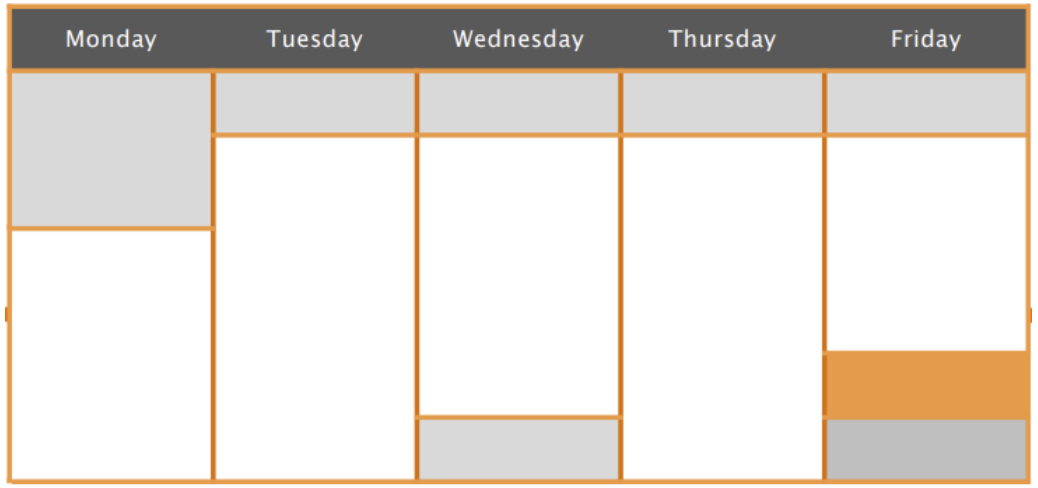
Daily Scrum

- Performed daily at any time suitable
- Only members of the development team
- It must be very brief: standing, only basic updates, max 15 mins
- Points out:
  - What's done
  - What will be done
  - Current problems and obstacles



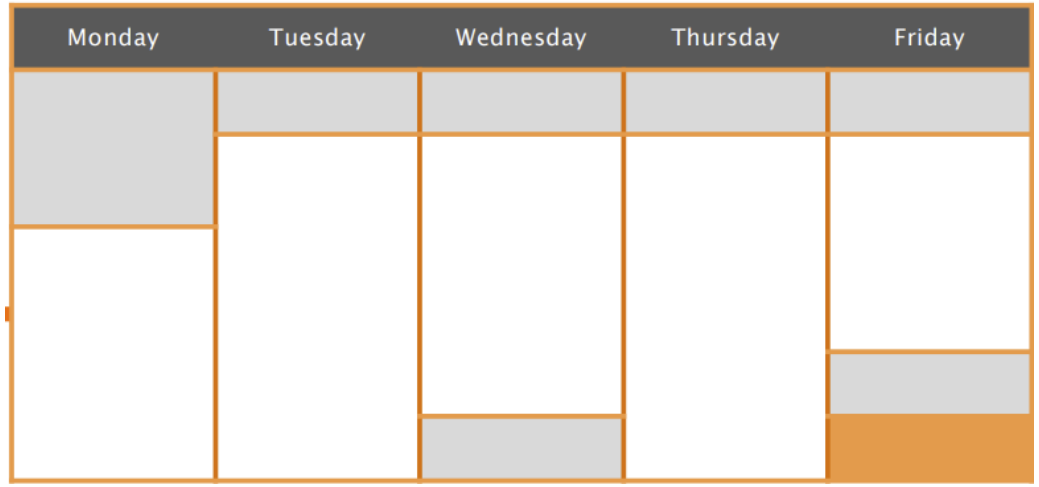
Sprint Review

- Show off some piece of working software to the stakeholders
- Report on incomplete stories
- Record the reactions of the stakeholders
  - They will be the basis for the product owner's future decisions



Sprint Retrospective

- Gather data about what happened, and the time-line and artefacts
- Generates insights about the causes of issues (no blaming!)
- Decide what to do next time
- Focus is on lessons learned



The Product Backlog

- It contains anything that will consume team resources
- In general, it contains PBIs (Product Backlog Items)
- Often, PBIs are **user stories**
- All the PBIs are ordered and assigned a scored in terms of relevance (a relative scoring)

The Product Backlog

Story	Story points
As an *unregistered user* **I want** *to create a new account* **So that** *I can buy items*	3
As a *customer* **I want** *to use my Card* **So that** *I can buy the items*	8
As a *user* **I want** *to add items* *to my wish list*	5
	TOT 16

Forecasted velocity: 12 pt per sprint
Sprint length: 2 weeks

What makes a good PBI: INVEST

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

What is a user story

- Essential description of a desired functionality
- The acceptance criteria are written by the PO and must be easy to turn into automated tests

As a *<actor type>*

I want *<to do something>*

So that *<some value is created>*

Stories and Epics

- A story is a manageable piece of requirements
  - (following the INVEST principle)
- The Epic is a story that we find out it is too large
  - … we will refine it by splitting it into stories

What ends in the Sprint Backlog

- In the sprint backlog we put committed stories (and the tasks for each story)
- Additional tasks, e.g.
  - Team improvement
  - Research work
  - Performance and security requirements
  - Bug fixing
- It is frozen at sprint planning and not changed until next sprint

Overview of artifacts, roles, activities

		Participants				Artifacts & Activities	
		Product owner	Scrum Master	Devel. Team	Stakeholder	Used Artifacts	Activities
Events	Sprint Planning	X	X	X	(opt)	Product backlog Sprint backlog	Definition of a sprint goal Selection of user stories Definition of tasks
	Daily Scrum	(opt)	(opt)	X	(opt)	Sprint backlog Burn-Down/up chart	Update each other
	Story time	(opt)	X	X	(opt)	Product backlog	Backlog refinement Estimation of future stories
	Sprint review	X	X	X	X	Software Sprint backlog	Show piece of working software Collect feedback
	Sprint Retrospective	(opt)	X	X	(opt)	What is deemed useful/necessary	Examination of the last sprint in order to identify possible improvements