Software Engineering

Lydie du Bousquet

Lydie.du-bousquet@imag.fr

In collaboration with J.-M. Favre, I. Parissis, Ph. Lalanda, Y. Ledru

Short introduction

- Lydie du Bousquet
  - Professor at UGA
  - *Software engineering, validation, test*

- Frédéric Lang
  - Researcher at INRIA
  - *Compositional Verification, New Generation Formal Description Techniques*

This class

- Introduction to software engineering as a tool box

Choose a tool to drill something


Mitre Saw


Sheet Sander


Orbital Sander with dust collection bag


Rechargeable Drill


Power Screwdriver


Circular Saw


Power Nailer/Stapler


Power Plane


Jigsaw


Power Drill


Palm Sander


Router


Oscillating Multi-Tool With Blade


Multi Purpose Saw


Circular Saw Blade

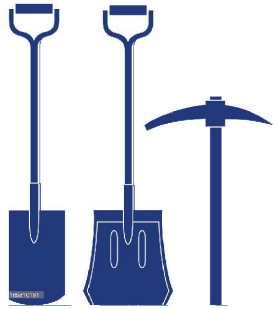

Lathe


Table saw


Drill Stand


Drill Press

Wait! What should I drill? What size?



Before using a tool, you should :

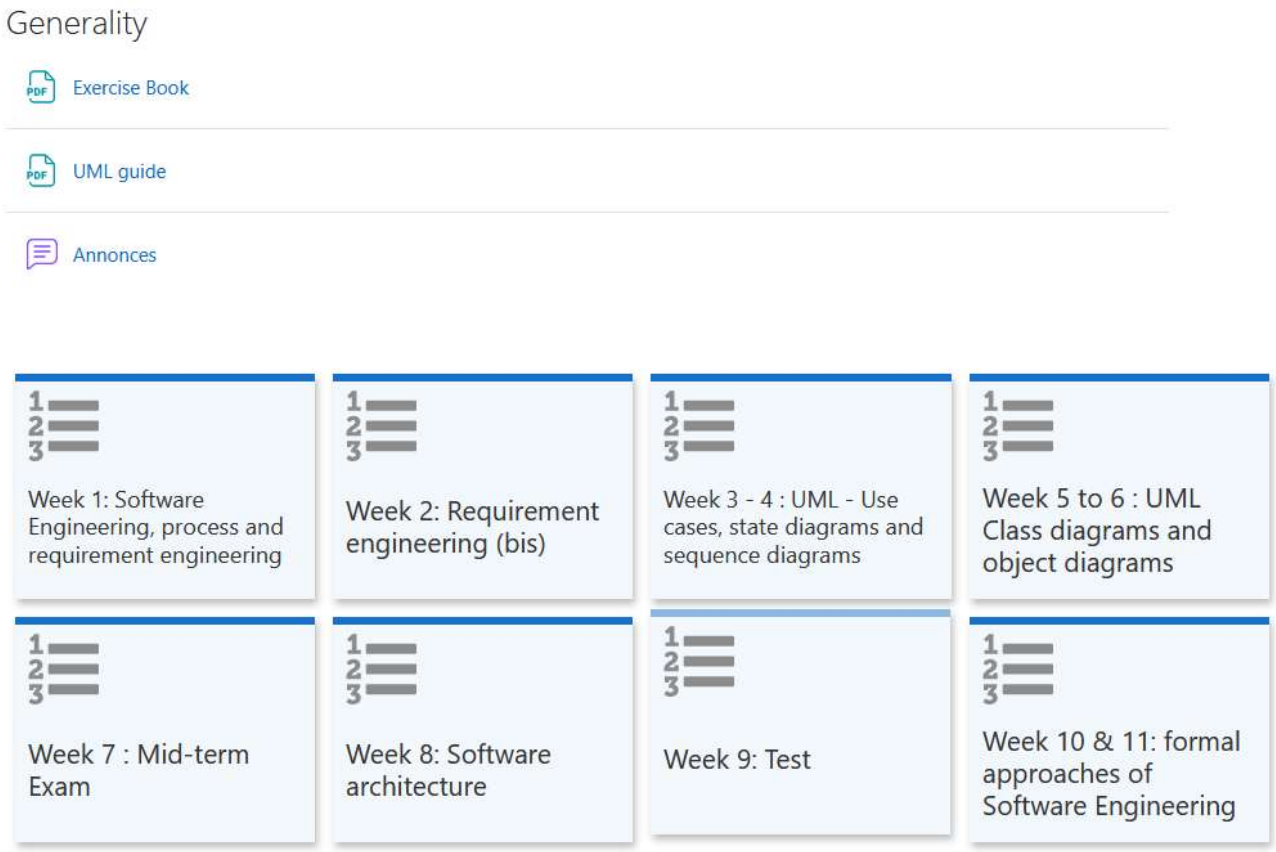
- Know the **characteristics of the tools**
- Learn to **use** them
- Be able to **choose** the appropriate one(s)

This is the objective of the class!
(for software engineering tools)

Class program during the semestrer

- Software engineering, development process
- Requirement engineering
- Modeling with UML
- Modeling with formal methods
- Validation by test

Moodle space



Evaluation

- One **mid-term exam** **(0.3)**
  - Exercises to check that the notions are mastered
  - Basic exercices but not so simple
  - Related to dev. process, req. eng. and UML
- One **final exam** **(0.7)**
  - More complex problems
  - Related to all the chapters

Class program of this week

1(a) Software engineering and dev. process

1(b) Requirement engineering

Schedule

- What is Software Engineering?
- What are the activities during the development?
- How are they organized?
- Which process should you choose?

What is software Engineering?

Exercise 0, Q1

What is software Engineering?

- Engineering?
  - derived from the Latin
  - ***ingenium***, meaning "cleverness"
  - ***ingeniare***, meaning "to contrive, devise" (find a solution, build)
- Software Engineering
  - discipline that is concerned with **all aspects of software production**

Why Software Engineering?

- Provide systematic methods, tools
- To achieve
  - predictability
  - precision
  - mitigated risk
  - professionalism

Schedule

- What is Software Engineering?
- What are the activities during the development?
- How are they organized?
- Which process should you choose?

What are the activities during a software project?

Exercise 0, Q2

Classical activities

- Requirements Analysis
- Specification
- Software architecture
- Design
- Implementation
- Testing
- Documentation
- Installation, deployment
- Training and Support
- Maintenance

Development process

- Also known as
  - development methodology
  - software development life cycle,
  - software process
- Process followed during the development of a software product
  - organization of the tasks or activities that take place during the development
  - several models of development process

What development process do you know ?

How the previous activities are organized?

Classical development processes

- Code and fix
- Waterfall development
- V-shaped Model
- Prototyping
- Incremental/iterative development
- Agile development, SCRUM, KANBAN, …
- Spiral

Classical development processes

A software engineer should

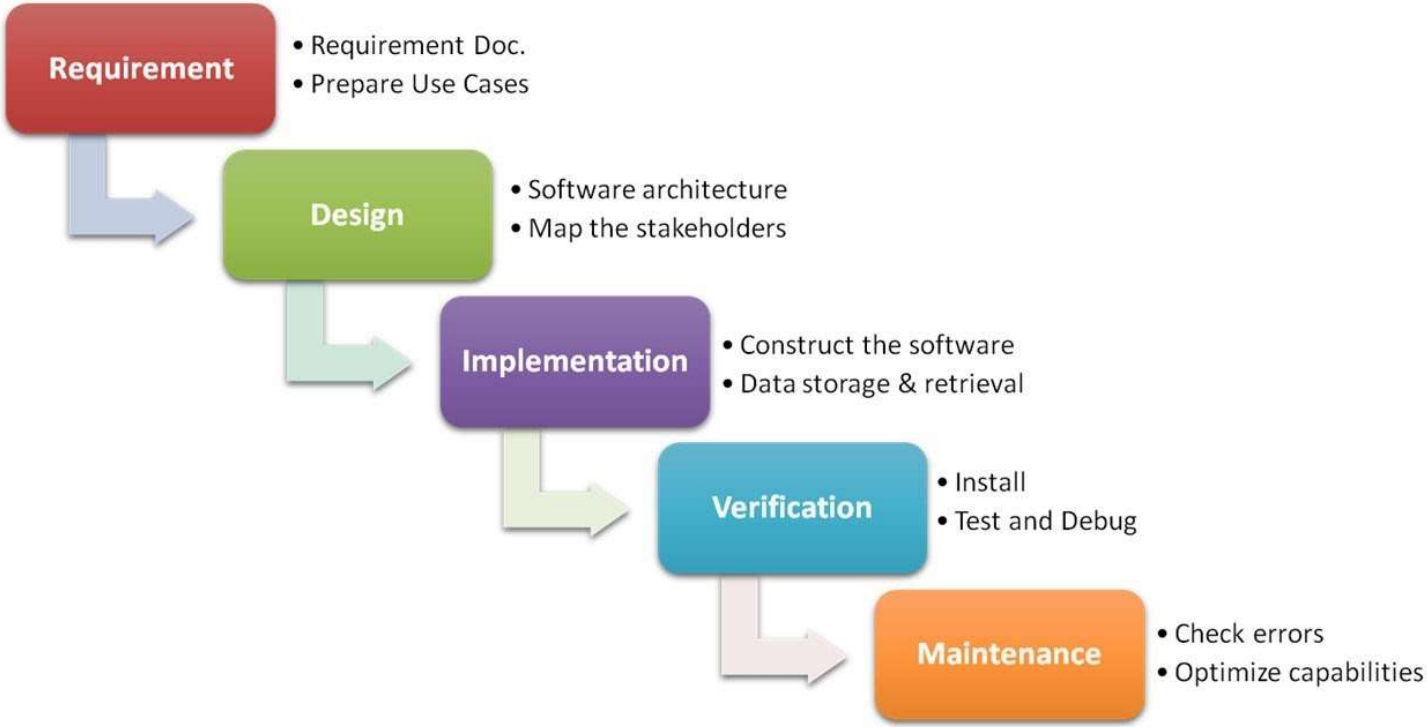
- Be able to recognize a development process
- Know the step order
- Know the advantages and the drawbacks
- Choose an appropriate process w.r.t. the situation

Code and fix

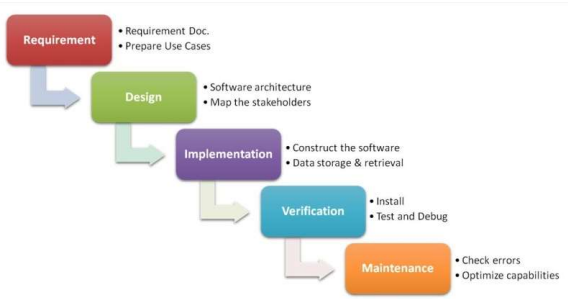
- Without much of a design in the way, programmers immediately begin producing code.
- At some point, testing begins (often late), unavoidable bugs must then be fixed before the product can be shipped.

Waterfall development

- **sequential** development approach,
  in which development is seen as flowing steadily downwards through several phases

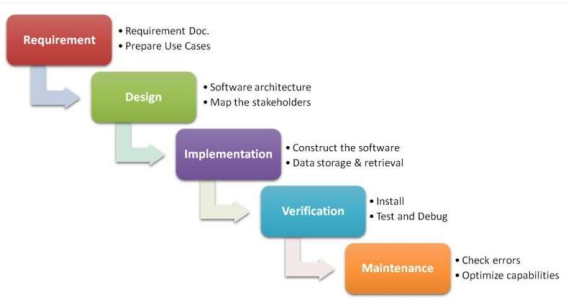


Waterfall development



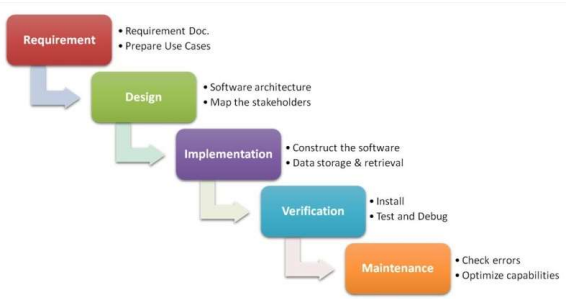
- The idea behind:
  - **structured approach**:
    the current phase should be finished **before** starting a new one
  - identifiable **milestones**:
    each phase is documented and validated

Waterfall development



- **Advantages**
  - Each phase has specific **deliverables**
  - **Verification** at each stage for early detection of errors / misunderstanding
  - Simple
- **Disadvantages**
  - Assumes that the requirements are **frozen**
  - Very **difficult to go back** to any stage after it finished
  - Little flexibility and adjusting scope
  - Executable is only available at the end

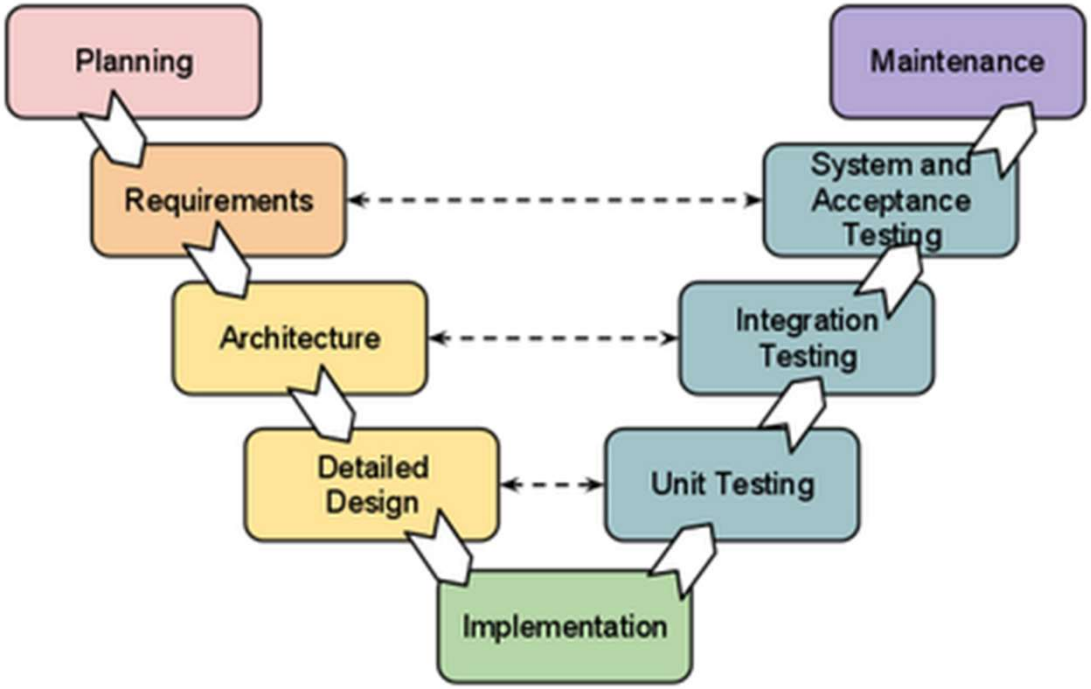
Waterfall development



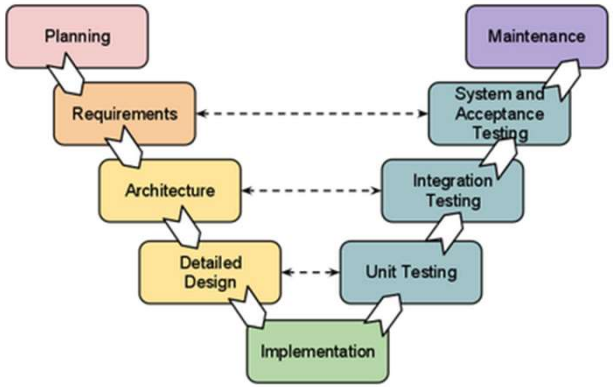
- **Can be used**
  - Projects with well-known requirements
  - Middle-size
- **Criticisms**
  - (Too) many documents
  - Clients may not know exactly what their requirements
  - Requirements may change

V-shaped Model

- Like the waterfall model, it is a **sequential** path of execution of processes
- **Testing** of the product is planned in parallel with a corresponding phase of development



V-shaped process



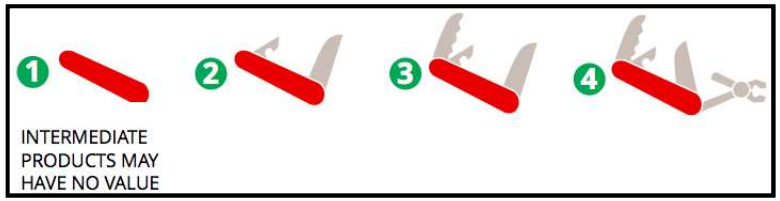
- The **idea** behind
  - Speed up and Improve validation (testing, most of the time)
- **Advantages**
  - Simple and easy to use.
  - Tests are prepared before/in parallel of coding.
- **Disadvantages** = same as water-fall process
  - If any changes happen in midway, then the test documents along with requirement documents have to be updated.
- When to use the V-model ?
  - For projects where requirements are easily understood
  - When validation is a key point,
    when you have an independent validation team

Incremental/iterative development

- Objective:
  - Being able provide an adequate final product
  - Reduce risks by breaking a project into smaller parts and providing more ease-of-change during the development
- Different possibilities:
  - A series of mini-Waterfalls are performed.
    All phases of the Waterfall are completed for a small part,
    before proceeding to the next increment,
  - **Overall requirements** are defined **before** proceeding to evolutionary, mini-Waterfall development of individual increments of a system, or
  - Requirements analysis, architecture and system core are defined, followed by developing successively the different parts (= increments).

Incremental development

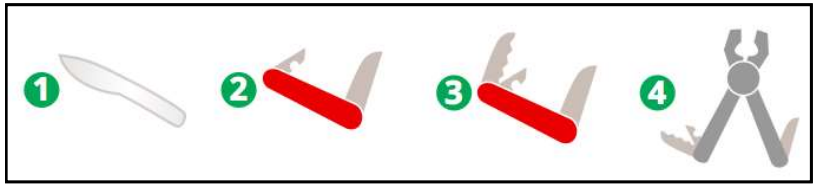
- **Principles**:
  - Requirements are globally collected,
  - Development is sliced into parts
  - Parts are developed at various times and integrated to obtain the releases
  - Each increment adds more software value – e.g. adding package



- **Advantages**
  - First release can be delivered sooner than in a waterfall process
  - Major parts can be developed first, the order of others can be modify
  - Users can provide feedbacks to adjust requirements for each the release
  - Easier to test and easier to manage risks than waterfall
- **Disadvantages**
  - Architecture generally chosen at the beginning => needs a complete definition of the whole system before it can be broken down into parts
  - Total cost is higher than waterfall

Illustration from https://productcoalition.com/product-development-using-agile-methodology-446c01ecd510

Iterative development



- **Principles**
  - Build a very first version,
  - Get some feedback and refine it to make better,
  - Keep doing that until the product is good
- **Advantages**
  - Reduce the rarely used features, maximize the frequently used features
  - Usable product at any time
  - Usually used in Agile processes (see after)
- **Disadvantages**
  - Increased pressure on User engagement
  - Each phase of an iteration is rigid with no overlaps
  - Requires higher level of technical excellence (than the other processes)
  - Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

Illustration from https://productcoalition.com/product-development-using-agile-methodology-446c01ecd510

Incremental vs **iterative** development

- The word **increment** fundamentally means **add onto**
- The word **iterate** fundamentally means **re-do**

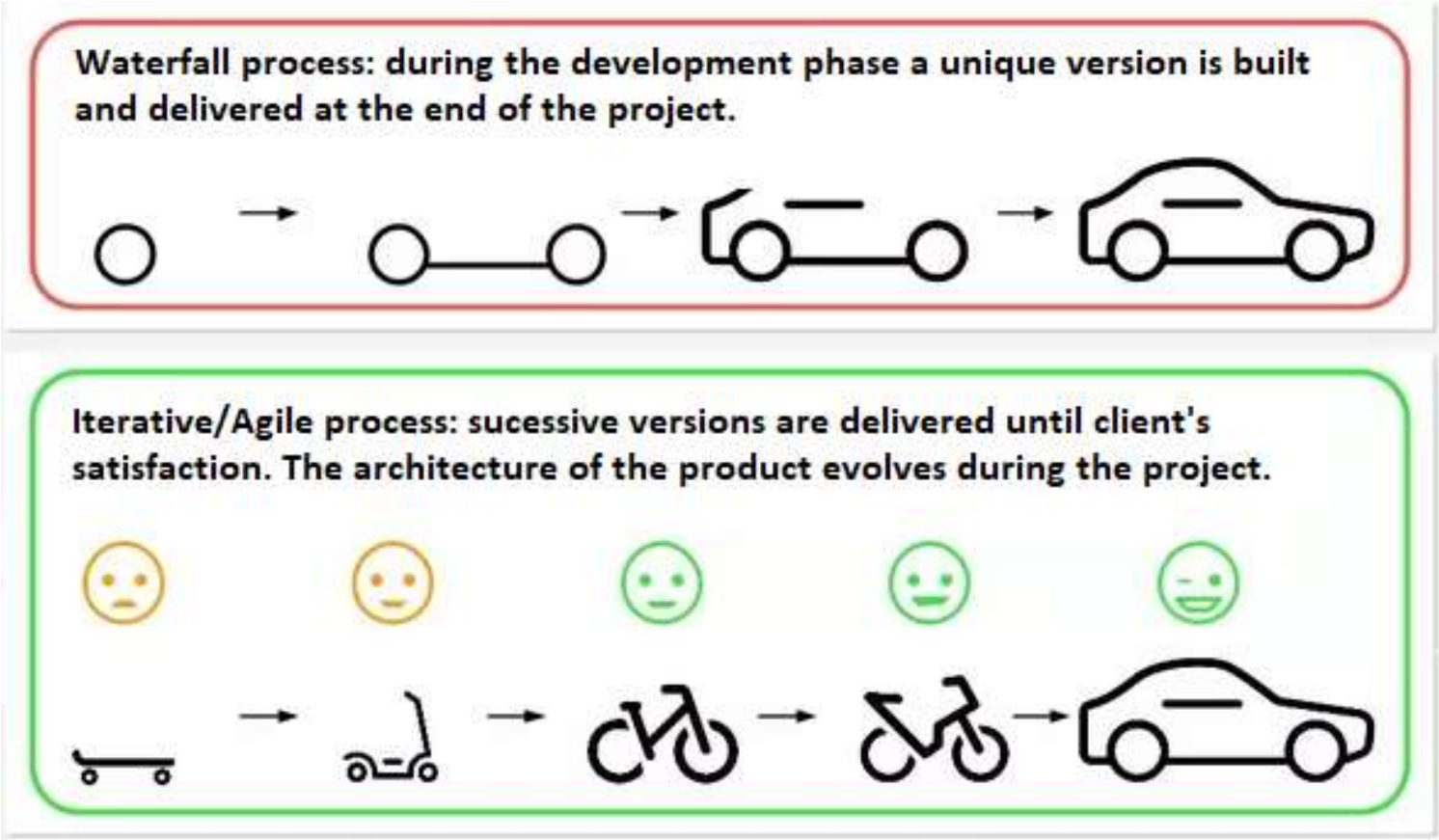
- For **incremental**,
  - you need to have well defined requirements
  - Between each increment, you can adjust part of the requirements
  - Several increments can be produced in parallel

- For **iterative**,
  - You start with a set of requirements in order to produce a first delivery
  - At each iteration, you collect new requirements and you make a new version

Incremental vs iterative development

Illustration from http://www.golali.co/waterfall-vs-iterative-model/ and from https://agilenotion.com/agile-categoriesiterative-incremental-evolutionary/

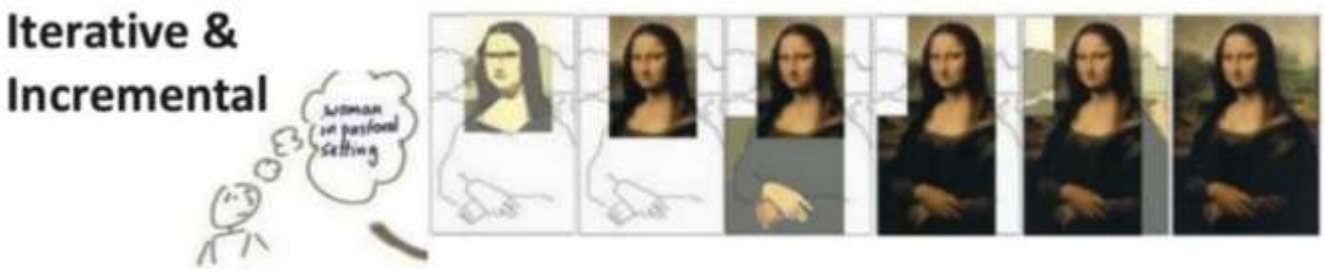
Waterfall vs Iterative



Adapted from an image of http://www.golali.co/waterfall-vs-iterative-model/

Agile development processes

- Mainly an iterative development processes
  - Rapid cycles (1 to 3 weeks)
  - Small release
- Examples of agile processes
  - XP (Extreme Programming)
  - Scrum
  - …



Agile processes: 12 principles

- **Customer satisfaction** by early and continuous delivery of useful software
- Welcome **changing requirements**, even in late development
- Working software is delivered frequently (weeks rather than months)
- Close, **daily cooperation** between business people and developers
- Projects are built around motivated individuals, who should be trusted
- **Face-to-face conversation** is the best form of communication (co-location)
- **Working software** is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to **technical excellence** and good design
- **Simplicity** (maximizing the amount of work not done) is essential
- Self-organizing teams
- Regular **adaptation** to changing circumstance

Agile software development practices

- Backlogs (Product and Sprint)
- Behavior-driven development (BDD)
- Continuous integration (CI)
- Daily stand-up / Daily Scrum
- Iterative and incremental development (IID)
- Pair programming
- Planning poker
- Refactoring
- Retrospective
- Story-driven modeling
- Test-driven development (TDD)

Agile processes:

- **Advantages**
  - **Customer satisfaction** by rapid, continuous delivery of useful software
  - **People and interactions** are emphasized rather than process and tools
  - Customers, developers and testers **constantly interact** with each other
  - Working software is **delivered frequently** (weeks rather than months)
  - Face-to-face conversation is the best form of communication
  - Close, **daily cooperation** between business people and developers
  - Continuous attention to **technical excellence** and good design
  - Regular adaptation to changing circumstances
  - Even late changes in requirements are welcomed

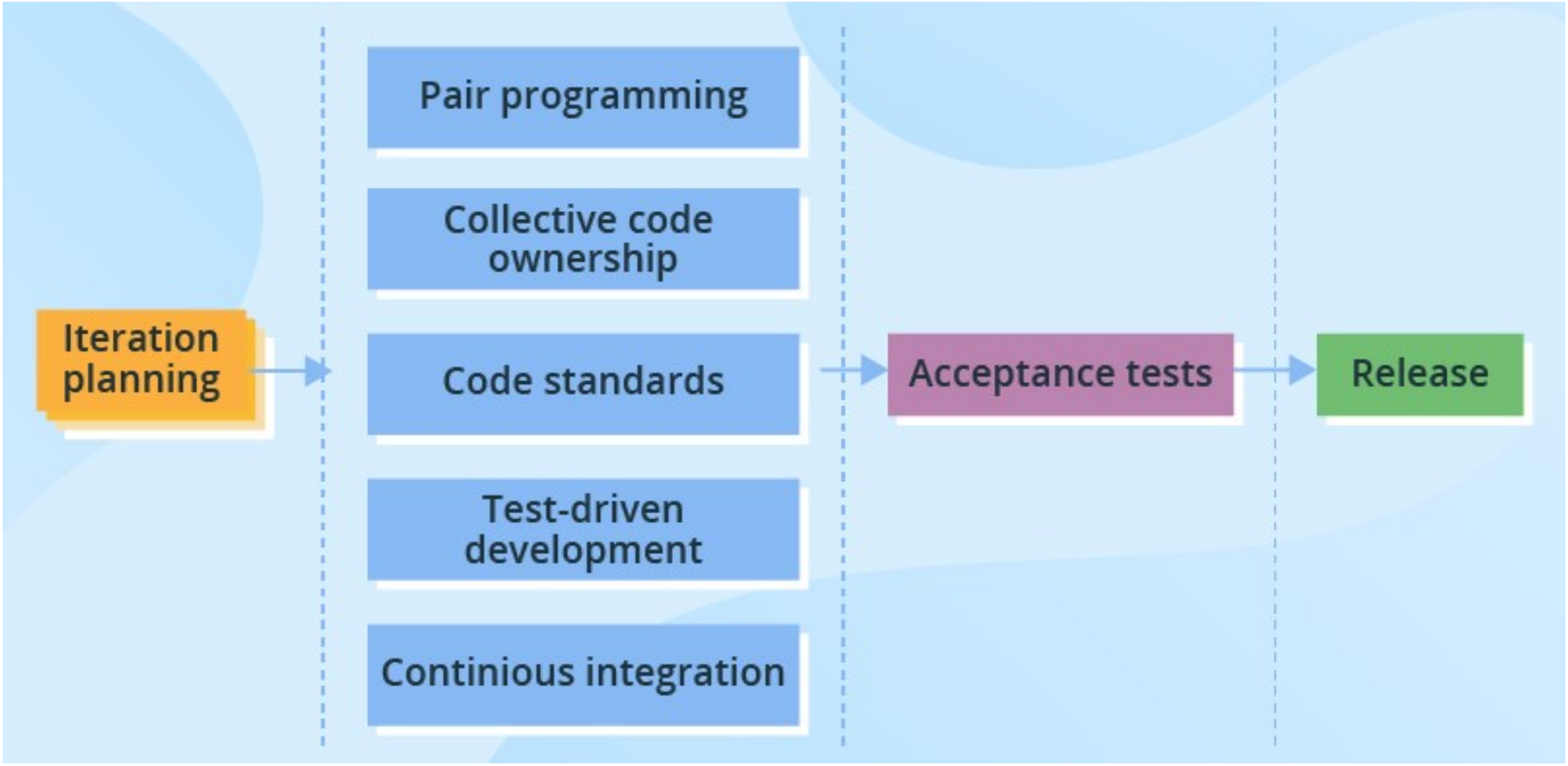
Agile processes

- **Disadvantages**
  - Difficult to **assess the effort** required at the beginning of the dev. life cycle.
  - There is **lack of emphasis** on necessary **designing** and **documentation**.
  - The project can easily get taken off track
- When to use Agile processes:
  - When new changes are needed to be implemented
  - End users' needs are ever changing in a dynamic business and IT world

Extreme Programming (XP)

- Agile project management
- Intended to improve
  - software quality and
  - responsiveness to changing requirements
- Based on multiple short development cycles to reduce the cost of changes in requirements
- Some key elements
  - programming in pairs or doing extensive code review,
  - unit testing of all code,
  - not programming features until they are really needed,
  - code simplicity and clarity
  - a flat management structure

Extreme Programming (XP)



www.scnsoft.com/software-development/software-development-models

Extreme Programming (XP)

- **Best for:**
  - Small, co-located teams (5–12 people) that deal with rapidly changing software requirements.
  - Complex modules or components that need frequent iterations and high-quality output, for example, core modules of a larger system under active development, where practices like pair programming and TDD ensure maintainability.
  - Rapid prototyping for R&D projects and innovative technology software.
- **Not ideal for:**
  - Large distributed teams with complex collaboration workflows.
  - Projects with less experienced developers or limited access to end users or product experts for rapid feedback and iteration.

Extreme Programming (XP)

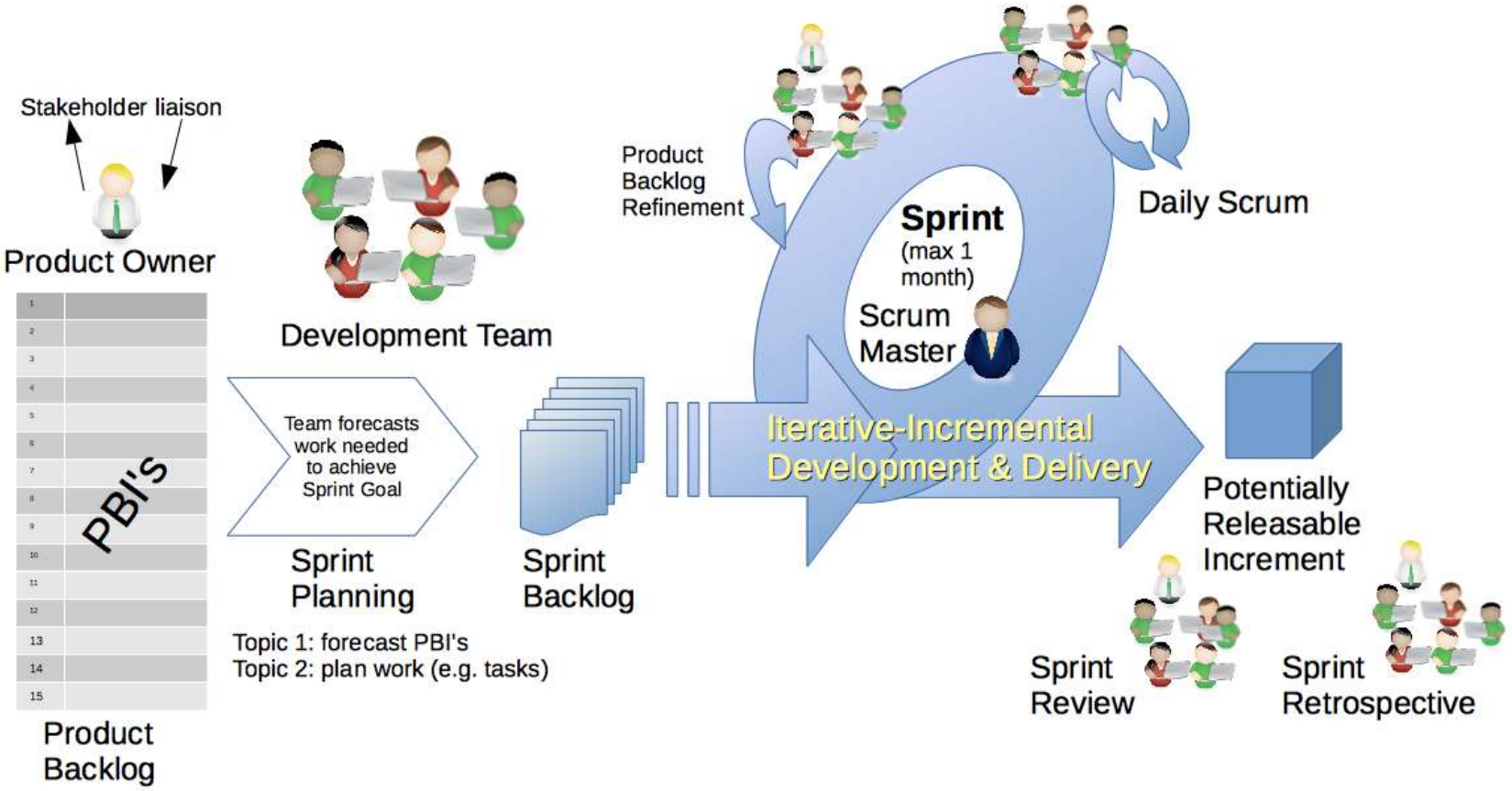
- **Benefices**
  - Ensures strong focus on code quality.
  - Welcomes mid-iteration changes.
  - Improves knowledge sharing through pair programming
- **Challenges**
  - Demands significant resources due to specific roles (XP coach, tracker, etc.) and multiple developers per task (e.g., for pair programming).
  - Requires constant client (or proxy) presence.
  - Relies on face-to-face communication => distributed teams hard to manage.
  - Creates pressure on team members due to rapid cycles and tight deadlines.
  - Requires mature infrastructure and DevOps practices to support frequent releases.
  - Knowledge management overtime (no documentation)

- According to the 2022 State of Agile report, only 7% of Agile teams use Extreme Programming.

SCRUM

- Agile project management
- Scrum team: product owner, developers, scrum master
- Sprint
  - fixed period, between one week and one month
  - starts with a sprint planning to define sprint goal
  - ends with
    - A *sprint review* = to elicit stakeholders feedback
    - A *sprint retrospective* = to identify lessons and improvements for the next sprints
- Daily scrum meetings
  - are intended to be less than 15 minutes in length,
  - to announce progress and issues that may be hindering the goal

SCRUM


Stakeholder liaison
Product Owner
PBI's
Product Backlog
Development Team
Team forecasts work needed to achieve Sprint Goal
Sprint Planning
Topic 1: forecast PBI's
Topic 2: plan work (e.g. tasks)
Sprint Backlog
Product Backlog Refinement
Sprint
(max 1 month)
Scrum Master
Daily Scrum
Iterative-Incremental Development & Delivery
Potentially Releasable Increment
Sprint Review
Sprint Retrospective

SCRUM

- Benefits
  - Provides a clear process
  - Maintains team focus on the most important work first through defined sprint goals.
  - Facilitates easy incorporation of changes.
- Challenges
  - Requires to make the roles and ceremonies effective.
  - Becomes time-consuming without strict focus and discipline.
  - Demands self-discipline to maintain code quality, as Scrum doesn’t prescribe specific technical practices (like test-driven development).

- 68% to 80% of Agile teams use Scrum

SCRUM

- **Best for:**
  - Feature-driven projects with a clear vision but evolving requirements.
  - Solutions that benefit from time-boxed sprints with regular demos and retrospectives, such as customer-facing apps, SaaS features, and new modules for enterprise software that need early validation by business users.
- **Not ideal for:**
  - Fast-paced projects where priorities may change daily.
  - Projects where work doesn't produce frequent changes visible to users (e.g., back-end refactoring, database migration, internal API or middleware development).

KANBAN

- Agile process
- Absence of pronounced iteration
- Work is visualized on a **Kanban board**,
- Tasks flow continuously based on team capacity
- Communication with the client is ongoing

www.scnsoft.com/software-development/software-development-models

KANBAN

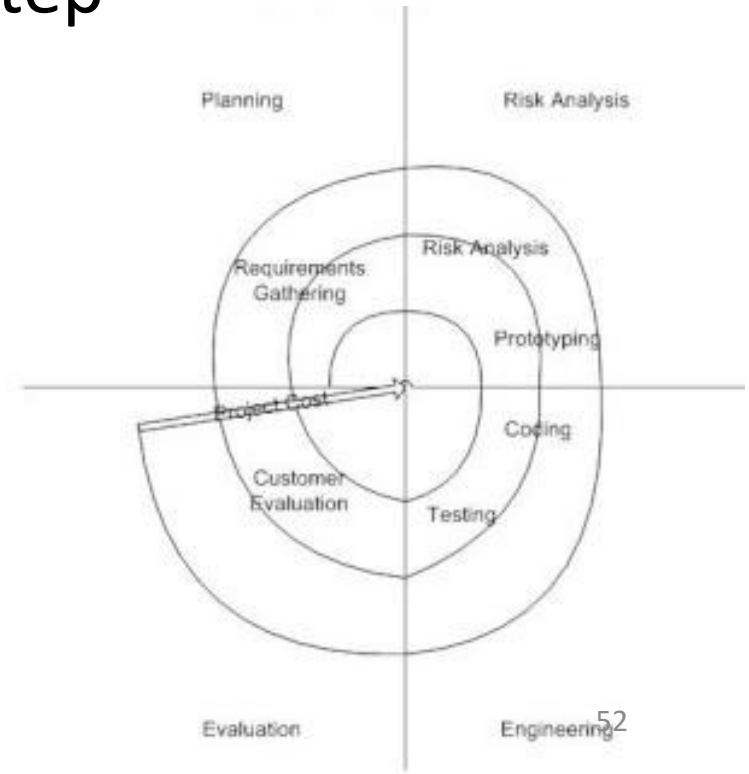
- **Advantages/benefits**
  - Enables quick delivery with minimal planning overhead.
  - Visualizes the entire workflow, improving progress visibility and task prioritization.
  - Allows tasks to be reshuffled on the fly.
- **Challenges**
  - Requires strong leadership to prevent miscoordination
  - Demands careful setting and enforcement of work-in-progress (WIP) limits to prevent delays and quality issues.
  - Risks moving in the wrong direction, delivering outcomes that don’t fully align with business needs.

KANBAN

- **Appropriate for:**
  - Support,
  - Maintenance and evolution projects where tasks like bug fixes and small updates arrive unpredictably.
- **Not ideal for:**
  - New development projects where features are highly interlinked and need to be delivered in a specific order.
  - Newcomer teams with less mature self-management and development discipline.

Spiral development model

- Risk-driven process model
  - **Meta development mod**el,
  - with emphasis placed on risk analysis.
- Each spiral corresponds to a “step” that can be used for an increment/iteration/activity
- Each spiral has four phases:
  - Planning,
  - Risk Analysis,
  - Engineering and
  - Evaluation



Spiral development model

- **Advantages** of Spiral model
  - High amount of risk analysis hence, avoidance of Risk is enhanced
  - Strong approval and documentation control
  - Additional Functionality can be added at a later date
- **Disadvantages** of Spiral model
  - Can be a costly model to use.
  - Risk analysis requires highly specific expertise.
  - Project's success is highly dependent on the risk analysis
  - Doesn't work well for smaller projects.

Spiral development model

- **When** to use Spiral model?
  - When costs and risk evaluation is important
  - For medium to high-risk projects
  - Users are unsure of their needs
  - Requirements are complex
  - Significant changes are expected (research and exploration)

Prototyping
(activity, not a process)

- Activity of creating a **prototype** of the application, = an **incomplete** version of the product
  - concerns only a few aspects of the final product
- **Outline**
  - Identify basic requirements
  - Develop an Initial Prototype
  - Review of the customers, including end-users, to examine the prototype and to provide feedback on additions or changes.
  - Revise and Enhance the specification / prototype using the feedback

Prototyping (activity)

- The **idea** behind
  - **Not** a standalone, complete development methodology
  - Attempts to reduce inherent project risk
  - User is involved throughout the development process
- Example of usages
  - Understanding of user requirements
  - Checking some technology relevance

Prototyping (activity)

- **Advantages** of prototyping
  - Improved and increased user involvement
  - Reduced time and costs
- **Risks** using of prototyping
  - Insufficient analysis
  - Prototype vs finished system
  - Excessive development time of the prototype

TDD (agile practice)

- Test Driven Development
  - Reverse the order between dev and unit testing
- Usual order
  - Code simple functions, then achieve unit testing
  - Done by the same person
    which makes testing not as efficient as expected
- TDD
  - Elaboration of the tests before the code
  - Help the developer to understand the requirements
  - Help the developer to identify corner cases
  - Can be applyied in any development process
- Not to be confused with V-cycle development process

Choice of a development process

- Learn the about the development processes
- Assess the needs of Stakeholders
- Use the criteria

Choice of a development process

	Flexibility to implement changes	MVP delivery speed	Client involvement	Risk management	Documentation complexity
Waterfall	Low	Low	Low	Low	High
V-model	Very low	Low	Low	Medium	Very high
Incremental	Medium	High	Medium	Medium	Medium
Iterative	High	Medium	High	Medium	Medium
Spiral	Very high	Medium	High	Very high	High
RUP	Medium	Medium	Medium	Medium	High
Scrum	High	High	High	Medium	Medium
XP	Very high	Very high	Very high	Medium	Medium
Kanban	Very high	Medium-high	High	Low-medium	Low

https://www.scnsoft.com/software-development/software-development-models

Exercise 3 for
Which development process?

- System for student management in a university (this system replace an existing system without any functional evolutions)
- An new interactive system for travelers to have schedules on their smartphones
- A system to control subway without drivers
- A very large system for Air traffic management
- A very new 3D-system for software maintenance
- Infrastructure and services for city that wants to become a “smart-city”