UNIVERSITY OF MISKOLC
FACULTY OF MECHANICAL ENGINEERING
AND INFORMATICS

Software Technology Lab

GEIAL316-B2a

Software models

Tamás Tompa, PhD
assistant professor
Department of Information Technology

Miskolc, 2026

The software crisis

- **Software Crisis (NATO Software Engineering Conference, 1968)**
- The most software projects failed
- Reasons:
  - they are delivered over budget,
  - they take longer than planned (over time),
  - they do not meet requirements,
  - they are of poor quality / inefficient / hard to maintain,
  - they cause financial / environmental / health damage,
  - or they are never delivered at all

→ The software crisis highlighted the necessity of software testing

→ Testing addresses quality problems and helps prevent damage

Testing in the software life cycle

- **Software Development Life Cycle** (SDLC)
  - After delivery the software, new demands appear
    - the software is further developed
    - renews cyclically
    - this is the life cycle
  - Software life cycle phases are defined by **methodologies**
  - New demands
    - the first and last step → cycle
  - The life cycle of a software
    - in theory, it is infinite
    - in practice, it becomes obsolete (language, technology, environment, etc.).

The life cycle of software

Software development methods

- **Goals**
  - **to define the steps of the software life cycle** and their order
  - to serve as a rulebook
    - to define what documents must be produced
    - what software should be developed and how, what way etc.
  - the choice of **methodology depends on project goals** and tasks

- Waterfall model
- V-model
- Prototype model
- Incremental development
- Rapid Application Development
- Agile development
- Extrem programming

Waterfall model

- The first published model of the software development process, derived from other development models
- simple phases
- each phase produces a document
- a phase can only start if the previous one has finished
- unflexible, requires major decisions early

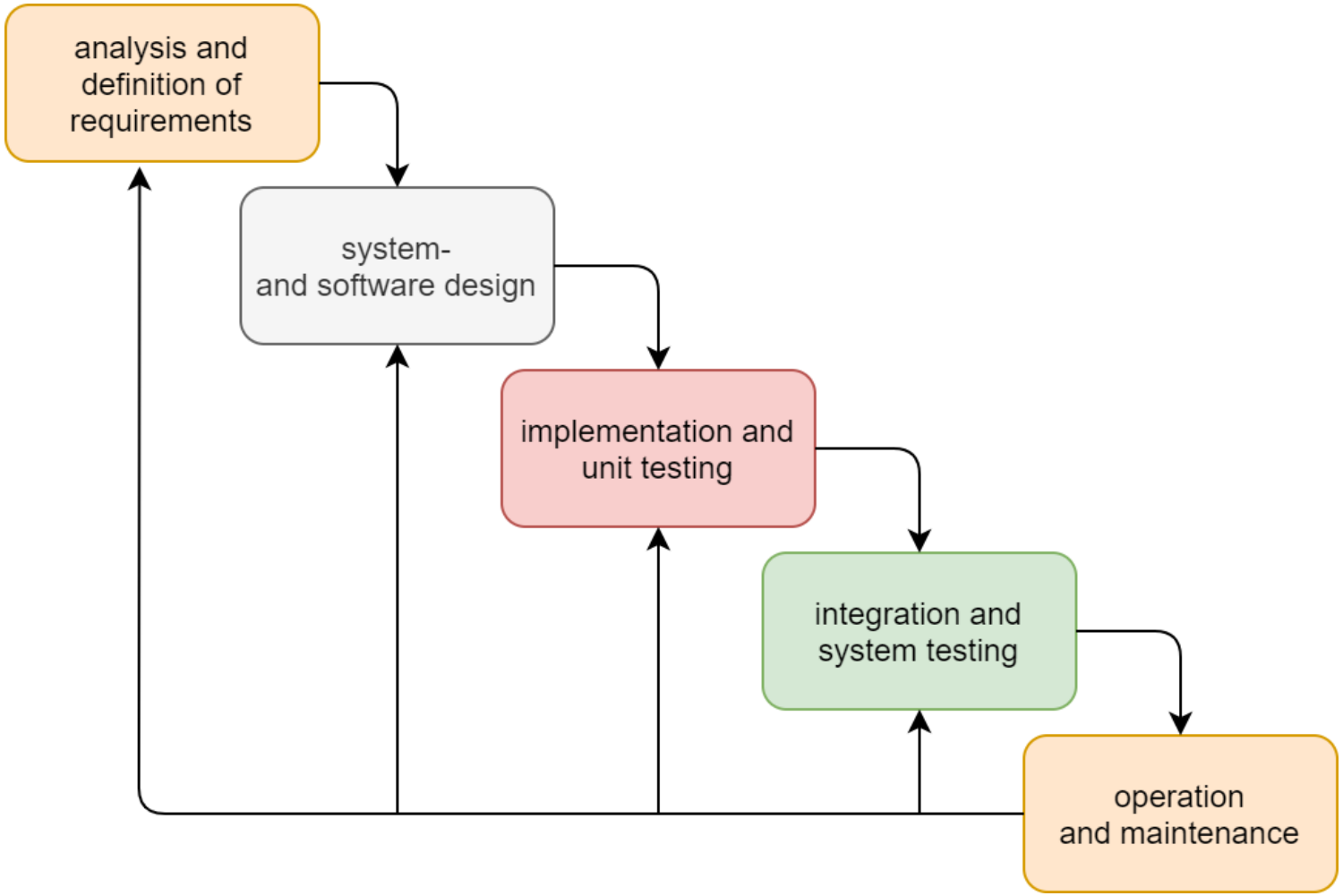
- **1. phase: analysis and definition of requirements**
  - The system is developed based on consultation with the users:
    - services of the system,
    - constraints,
    - goals

- **2. phase: system- and software design**
  - defining the system architecture

Waterfall model

- **3. phase: implementation and unit testing**
  - the software design will be implemented in this phase
- **4. phase: integration and system testing**
  - integration of individual, separate program units
  - testing as a complete system
- **5. phase: operation and maintenance**
  - fixing discovered defects, errors
  - further development of system components
  - new demands may arise, thus may be necessary to further develop the system's services

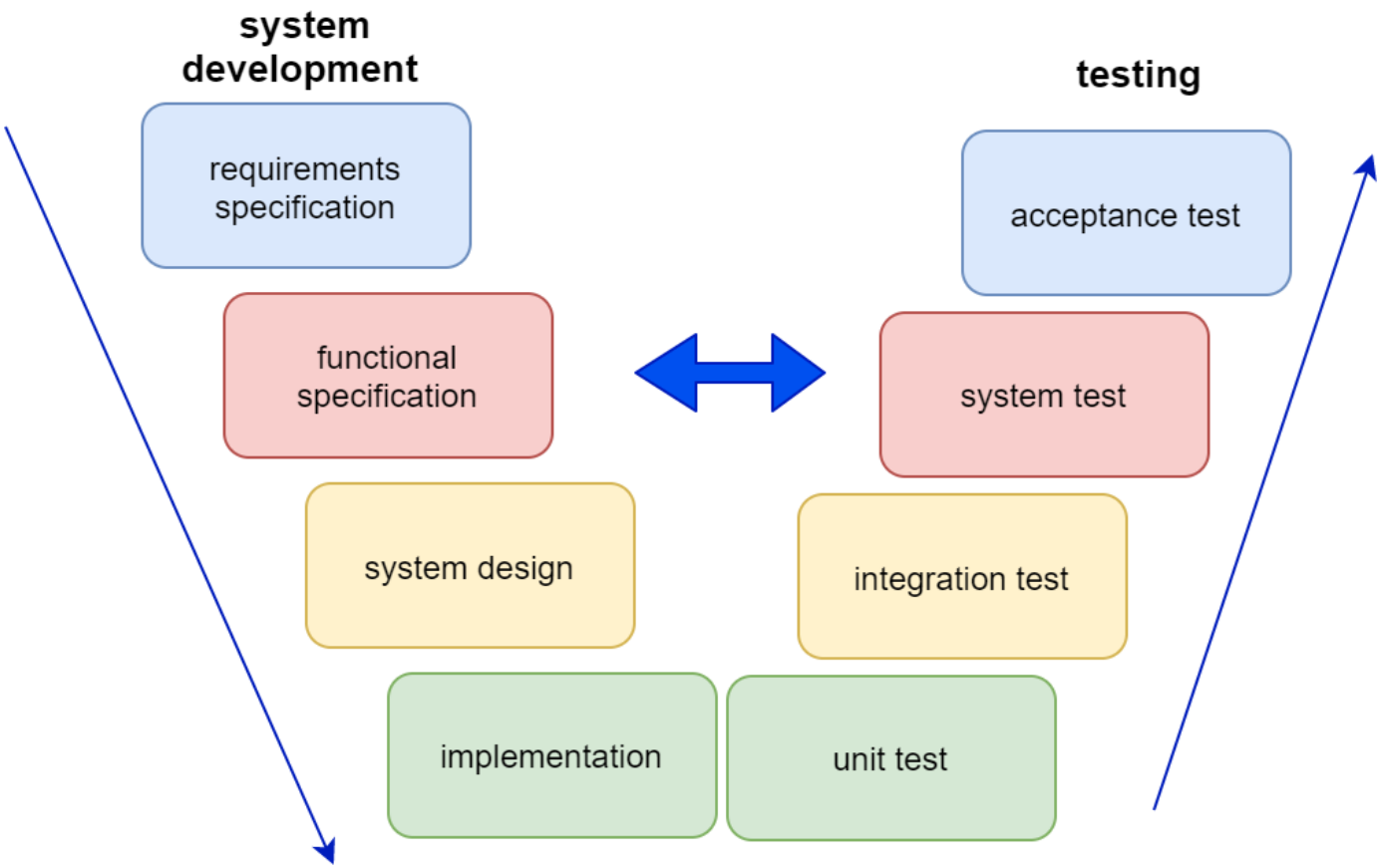
Waterfall model



V-model

- Two branches forming a V shape
  - **development branch → waterfall model**
  - **+ testing branch**
  - testing branch complements development with testing
    - development then testing
    - if testing discovers errors, return to development
  - corresponding development and testing levels belong together
    - testing phase uses documents created in the development phase, or testing the developed product
- Widely used but very rigid, unflexible
- Suitable if requirements, demands do not change during the development
- Otherwise → iterative or agile methods are preferred

V-model


system development
testing
requirements specification
functional specification
system design
implementation
unit test
integration test
system test
acceptance test

V-model

- **Disadvantages**
  - derived from the Waterfall model
  - unflexible and poorly adaptable to change
  - presents software development as a linear process (not a linear process)
  - it also treats testing as an inflexible process, instead of leaving it up to the testers to find the best method to investigate a given problem
  - many variants cause uncertainty

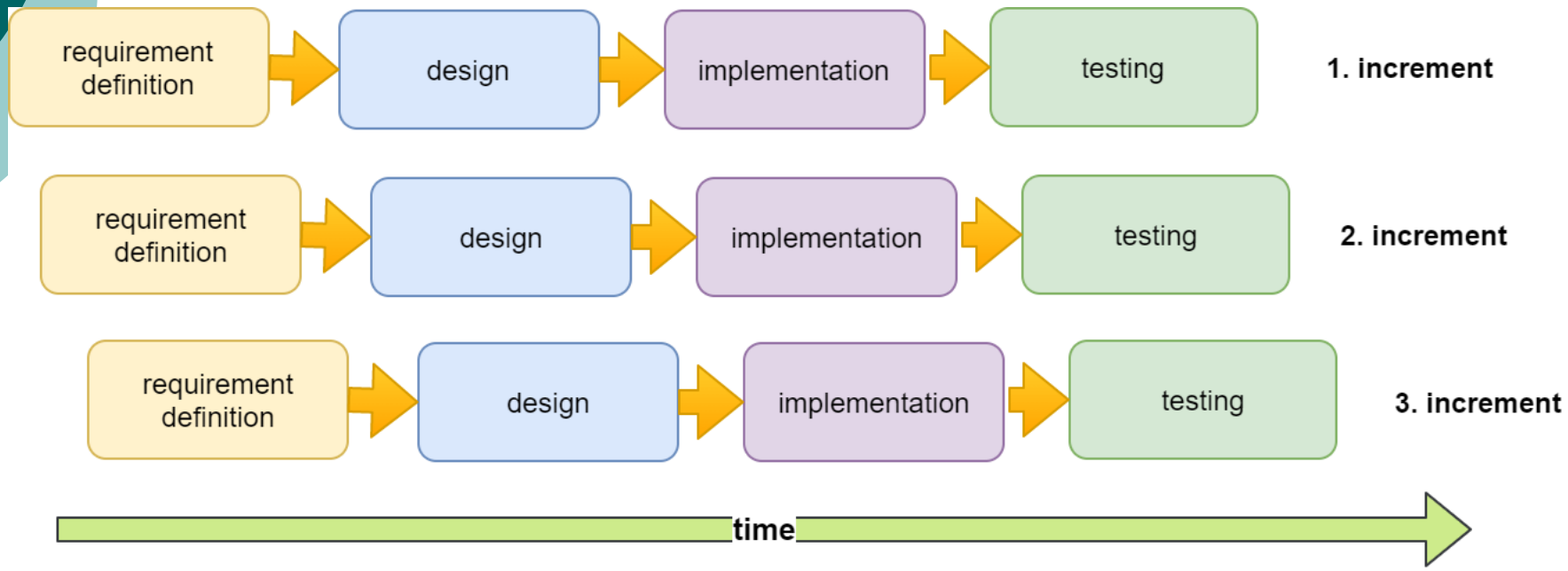
V-model - Test

- **Testing**
  - testing is executed after implementation at a given level
  - but it does not mean that the testers' work begins when the development is completed
  - once the functional and non-functional requirements are established, the design of acceptance test cases and compliance expectations can begin
  - in the case of program units, creating unit tests based on the specification of their interfaces

Iterative-incremental methodologies

- **the development is a series of small iterations**
- **design and implementation in every iteration**
- the life cycle steps are not sequential, as in structured methodologies, but overlap in time
- **continuous refinement/development**
  - each iteration complements the already developed prototype
- they focus on the process (for the iterations)
- by adding new additions, a growing subsystem is created that needs to be tested
- most of today's methodologies they belong to this family
  - prototype model
  - rapid application development (RAD)
  - Rational Unified Process (RUP)
  - agile development models

Iterative-incremental methodologies



Iterative-incremental methodologies - Test

- **Similar to V-Model**
  - **over time, test design and testing tasks are increasingly deeper level**
- inspection, feedback and proposal appearance
- involving testers in the development process as soon as possible
- building on their inspection work at all levels of development
- good solutions can execute in the early phases of the project
  - based on experience
- but: continuous changes → incorretly/badly structured system

Prototype model

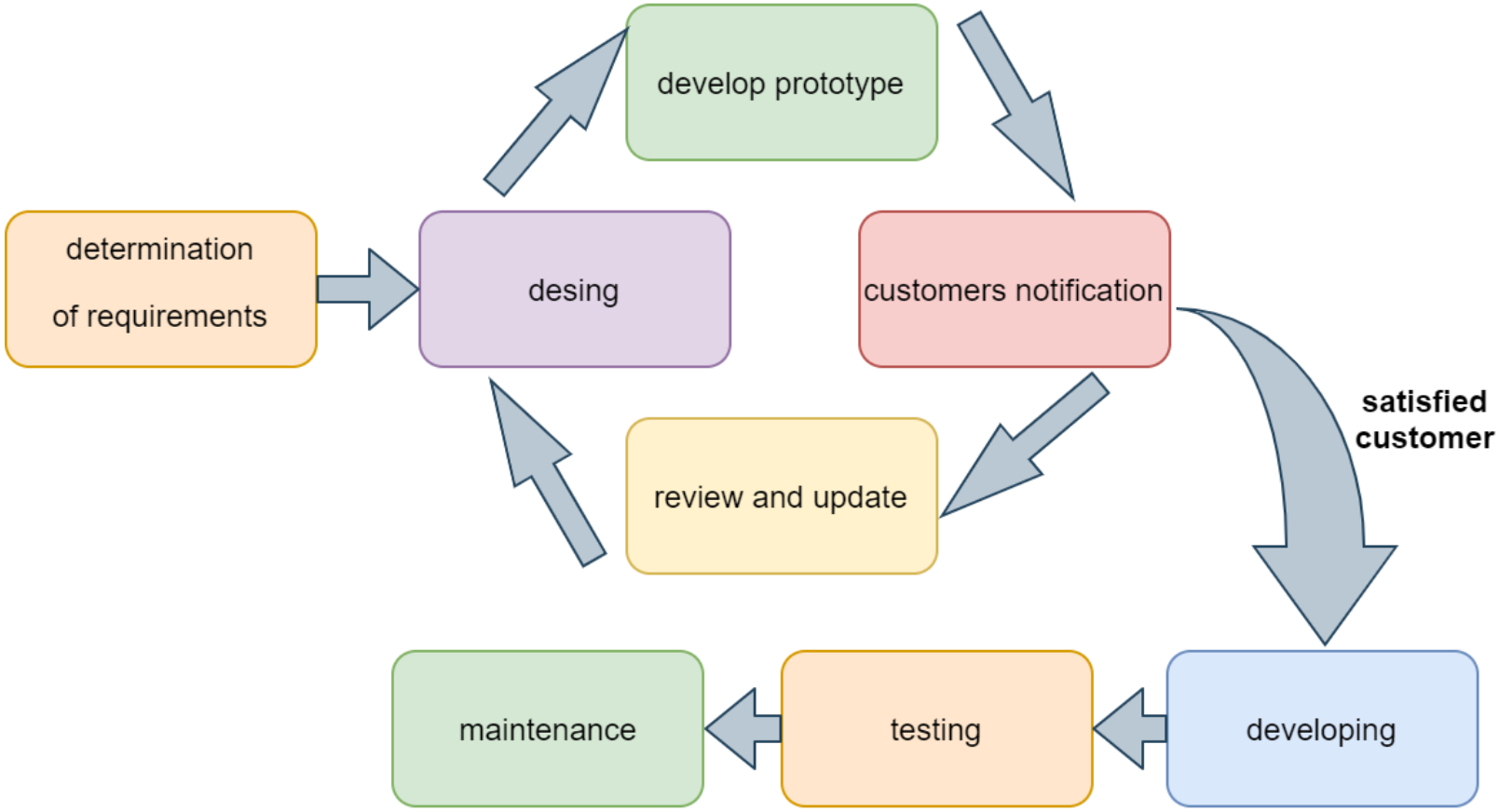
- **In case of the Waterfall model:**
  - **users see the system only the end of project**
  - often only at that time clear that they misunderstood each other
  - problem solving (solution for that) → **prototype model**

- **Prototype model**
  - the customer's business processes and requirements cannot be fully understood
  - these change over time
  - refining requirements using prototypes
  - after using the prototype, stating why it does not meet the requirements
  - then refinement
  - there will be a lot of dialogue between the system and the user

Prototype model

- **Step 1** : determination of **requirements**
- **Step 2** : develop **an initial prototype**: Only the user interfaces, the real functions (on GUI) not
- **Step 3** : **demo** : a type of user acceptance test
- **Step 4. refine** the requirements: refine the requirements specification based on feedbacks. If the specification is still not precise enough, then further develop the prototype and return back to step 3.

Prototype model



Agile software development

- **how to produce good software (90s)**
  - thoroughly design project
  - knowing about quality requirements
  - use of analysis and design methods
  - controlled, precise processes
  - → a lot of extra work
  - → more time spent on planning than on real development and testing

- **→ for these reasons**
  - application of new, fast (algyl) methods
  - **the development team focuses on software development and not on design and documentation**
  - keywords: rapid iterations, customer focus, flexibility

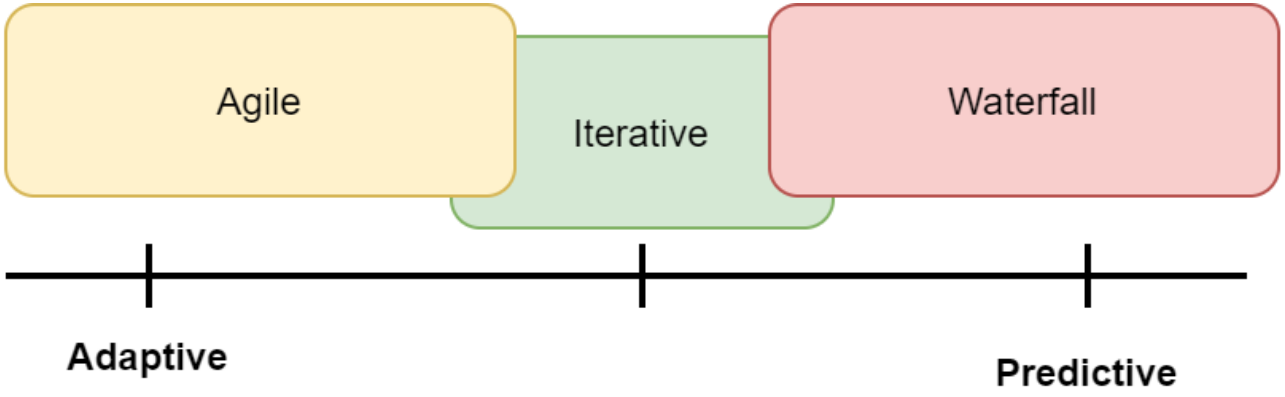
Agile software development

- **iterative software development method**
- **2001 (Agile Manifesto publication): contain basic principles**
- word meaning: agility, speed
  - an ability that is able to respond to changes
- participants' adaptation to the project
- According to this perspective:
  - **individuals** and interactivity are more valuable than **processes** and tools
  - **working software** is more valuable than **documentation**
  - **cooperation** is more valuable with the customer than **negotiations**
  - **the adaptation to changes** is more valuable **than follow a plan**

Agile software development

- **Principles**
  - the most important is the customer to be satisfy
  - late changes of the requirements are not a problem
  - delivery of working software/ delivery prototypes regularly as soon as possible
  - daily cooperation between the customer and the developers
  - the most effective communication type is face-to-face meeting
  - progress is based on working software
  - continuous attention to technical excellence
  - simplicity for maximum efficiency
  - self-organizing teams make the best plans/designs

Methodologies - comparison



- Overlapping between methodologies
- Agile ↔ Waterfall

- Adaptive
  - long-term planning, no prediction
  - focuses on immediate problems
  - "What are we going to do this week?"
- Predictive
  - planned steps
  - every step focusing on the whole

Agile software development

- E.g.
  - Scrum
  - Extreme Programming (XP)
  - Kanban

  - Common characteristics
    - less documentation
    - increasing flexibility, decreasing risk
    - easier communication, improved cooperation
    - involving the customer in the development process
    - weeks instead of months
    - deadlines are important
      - they are not flexible

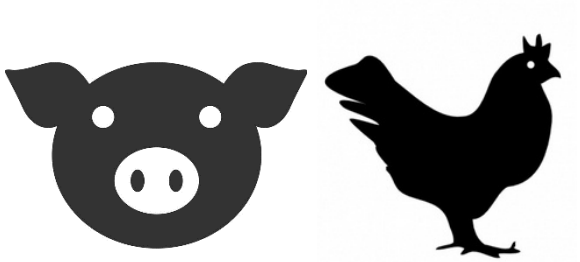
Scrum

- The term borrowed from rugby
  - meaning: to struggle, to clash
  - 1990
- remaining the sport attitude
  - E.g. relay race
    - baton is the software
    - the phases are the runners
    - if one runner performs poorly, the whole team loses
- Basic idea
  - the phases are strongly overlapped
  - smaller groups, experts in various topics
  - they work together in all phases

Scrum - characteristics

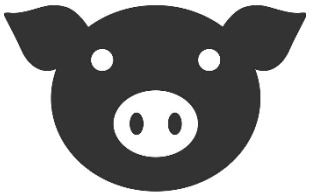
- Adaptive properties
- There is no “script”
- The development team starts working at the same time
  - the team is responsible for the final result
  - Scrum Team
- Communication
  - different fields of expertise
- Transparent, clear
  - who and why is responsible for, what deadline

Scrum - roles



- The pig and the chicken are walking on the street
- the chicken says:
  - “Let’s open a restaurant!”
- pig :
  - "Good idea, what should it be called?"
- The chicken thinks about this and then replies:
  - "Let's call it Ham and eggs!"
- Then the pig:
  - “I don’t like it, because I would definitely give it my all, and you would just like to participate.”

Scrum – “Pigs”



- they give their "blood" for the success of the project
- are committed to the success of the software project

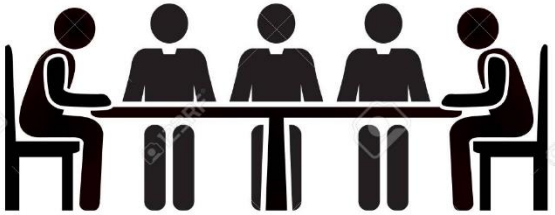
- Pigs
  - Scrum Master
    - operation, process supervision
  - Product Owner
    - customer, responsible for ensuring that the team always develops the part of the product that is most important
  - Team
    - team of 5-9 people, from different areas
    - implementation of tasks
    - teamwork is important → “passing”

Scrum – "Chickens"

- indirectly participate in the process

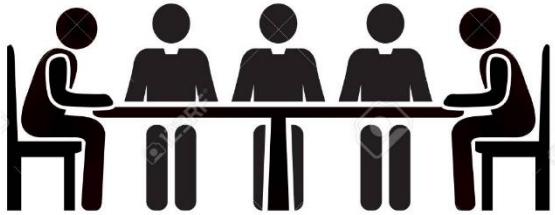
- Chickens
  - Business actors (Stakeholders)
    - customers, distributors
  - Management (Managers)
    - creating an environment for teams
    - suitable, best environment

Scrum – meetings



- Sprint Planning Meeting
  - who can take on how much work, and then the team decides which stories to take on for the next sprint based on this
- Backlog Grooming/Backlog Refinement
  - refining the Product Backlog together with the Team, For example, a task may be too large, it will become a story and then be divided into tasks will be processed
- Daily Meeting/Daily Scrum
  - during the sprint, a short meeting must be held every day, maximum 15 minutes
  - What have you been doing since yesterday's meeting?
  - What are you going to do until the next meeting?
  - What problems did you encounter while solving the given task?

Scrum – discussions



- Sprint Review Meeting (Race overview)
  - at the end of every sprint the participants come together, and they will check which stories that succeeded to prepare and it than the meets the requirements

- Sprint Retrospective
  - the one most important meeting. One of the Scrum most important function to highlight the problems that hinder developers in solving problems → adaptation to the tasks

Scrum documents

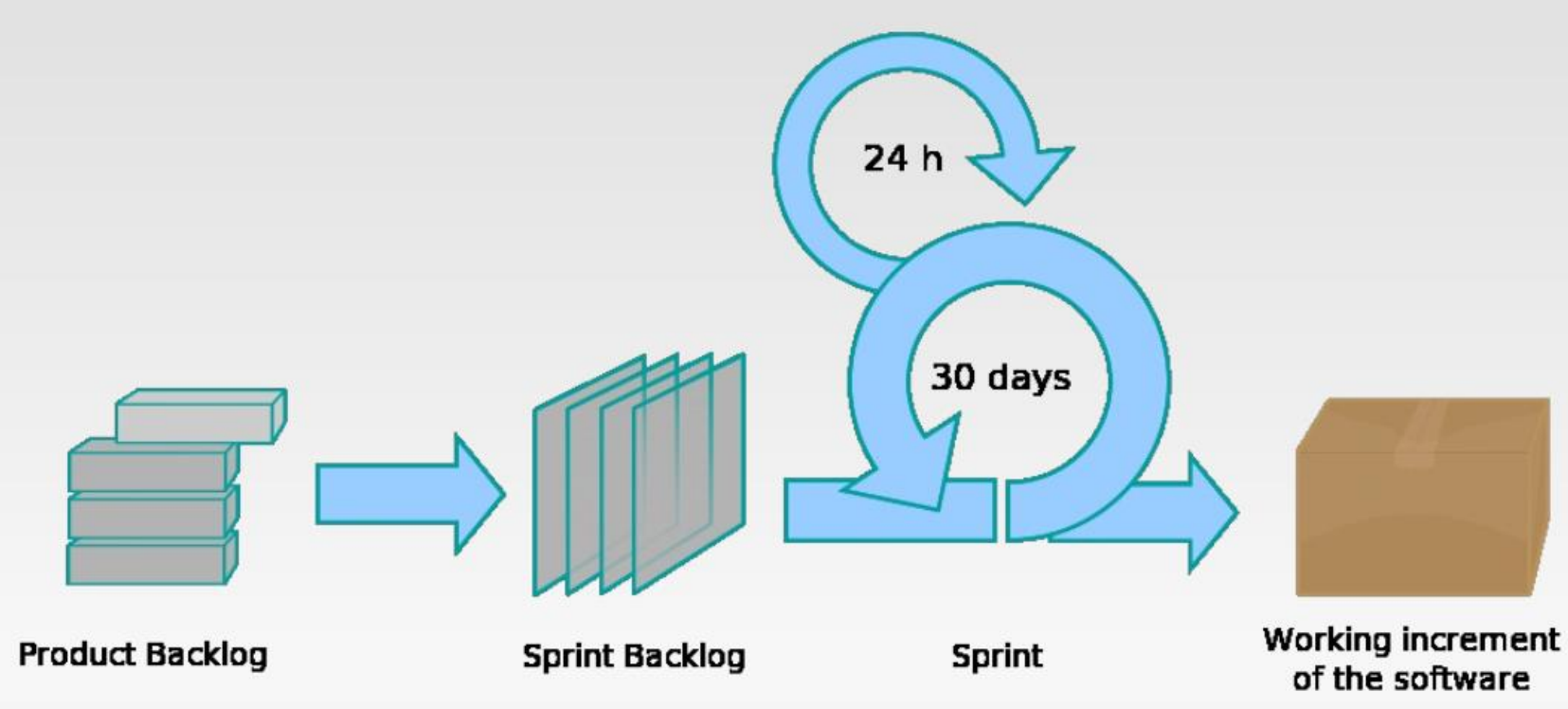
- Story
  - important description from the customers
- Product backlog
  - story processing, with priorities
- Sprint backlog (sprint to-do list)
  - detailed tasks definition to the given sprint (who, what how, what deadline)
- Burn down chart (Daily Result Statement)
  - diagram which help to show that the ideal to work pace compared to how the team is progressing the current sprint within

Scrum – concepts

- Sprint (race)
  - a development period of a pre-agreed length, usually 2-4 weeks
  - iteration cycle → must be repeated until there are no user problems waiting to be solved
  - producing deliverable software at the end of each sprint

- Obstacle/Problem (Impediment)
  - inhibitor factor that makes the job hinders
  - only workplace problems considered obstacle (the team members private problems not)
    - the obstacle for example, expired a software license
  - the Scrum Master must be solve the problems (avoid the obstacles)

Scrum



Source: https://hu.wikipedia.org/wiki/Scrum

Extreme programming

- Extreme Programming, XP
- agile methodology
- adopts well-proven techniques from other methods
  - apply them not just well, but extremely well
  - everything else considers it unnecessary
- ≠ "programming to faint" method
  - 24-hour or 48-hour programming competition

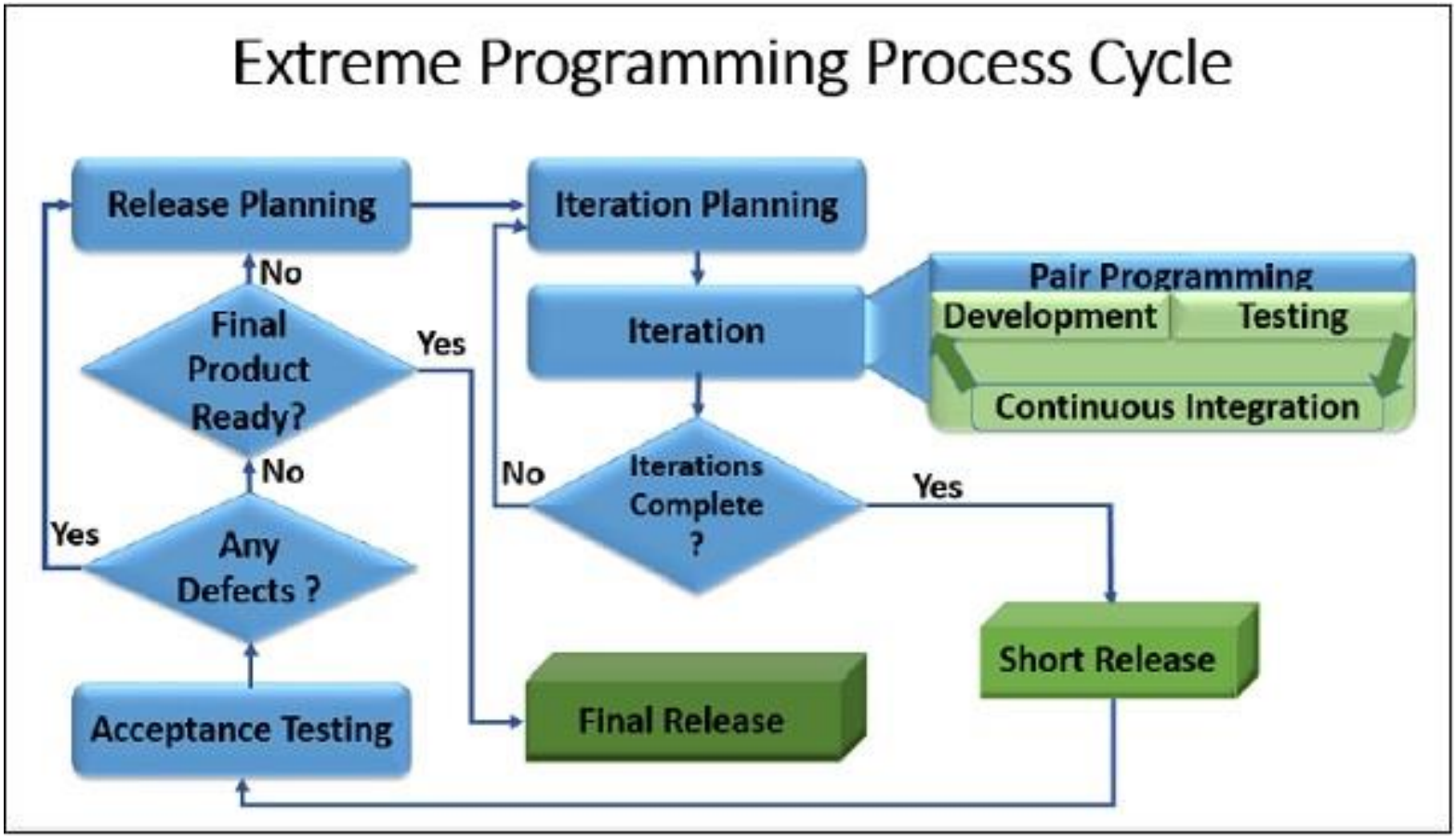
Extreme programming

- **4 activities**
  - Coding
    - most importantly, difficulties arise communication between developers → everyone understands the same
  - Testing
    - We can't be sure that a feature works properly until we test it. According to the extreme view, little testing will find few bugs, and a huge amount of testing will find them all

Extreme programming

- **4 activities**
  - Attention
    - developers must pay attention to customers and understand their demands
  - Planning
    - witout planning cannot be developed software, ad-hoc solutions lead to an opaque structure. Have to be prepared for changing demands, have to design the software that its components will be independent from other components

Extreme programming

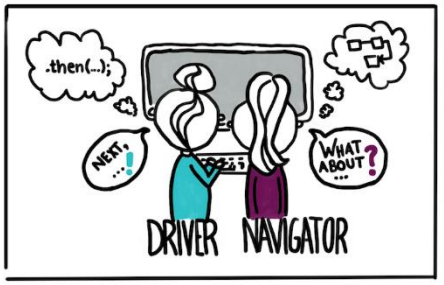


Source: https://www.tutorialspoint.com/extreme_programming/extreme_programming_process_cycle.htm

Extreme programming

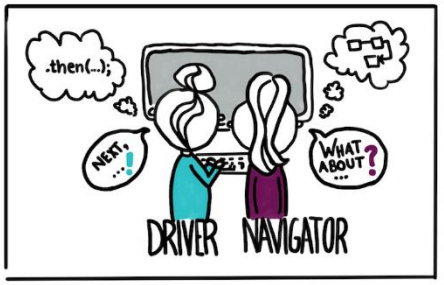
- **Typical techniques**
  - **Pair programming**
    - two people write a code, ↔ listen, if find an error than discuss
  - **Test driven development**
    - first write unit tests then implement methods
  - **Source code review**
    - lead developer reviews the code → how to do it better
  - **Continuous integration**
    - integration test of codes entered into a version control system
  - **Code refactoring**
    - already tested, working code can be improved, but its functionality cannot be changed.

Extreme programming



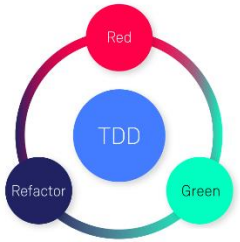
- **Typical techniques**
  - **Pair programming**
    - two people write code together
      - write ↔ pay attention, see a mistake and indicate
    - two roles:
      - Driver: writes the code
      - Navigator : monitors the code, finds errors, and suggests strategies
    - continuous communication: developers constantly discuss potential problems

Extreme programming



- **Typical techniques**
  - **Pair programming**
    - Consequences:
      - **better quality code**: real-time verification reduces the number of errors
      - **knowledge sharing**: develops the team's knowledge and experience
      - **faster problem solving**: two people can more easily find an optimal solution
      - **but more mentally tiring**: requires intense concentration, so regular breaks are recommended
      - its application is useful for solving complex problems and mentoring junior developers

Extreme programming

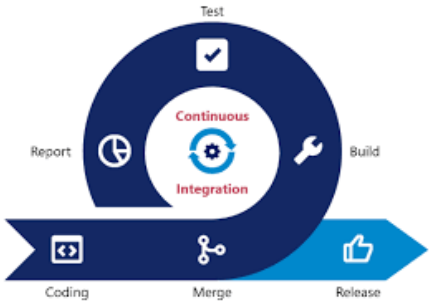


- **Typical techniques**
  - **Test driven development**
    - first write unit tests then implement methods
    - → coding starts with writing tests
    - Red-Green-Refactor cycle:
      - red: write a test that will fail first
      - green: write the minimum code to pass the test
      - refactor: optimize code while preserving functionality
    - continuous feedback: tests provide immediate feedback on the quality of the code
    - better code quality: results in more modular and maintainable code
    - fewer errors: filters out errors in early phase of development

Extreme programming

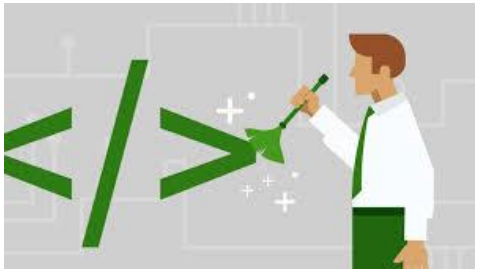
- **Typical techniques**
  - **Source code review**
    - lead developer reviews the code → how to/can it be done better
    - Team members can also review each other's code
    - Goals:
      - filtering errors and security issues
      - improving code quality and readability
      - adherence to best practices and conventions (“this is how we do it”)
      - checking for small, easily visible changes
      - providing specific and constructive feedback

Extreme programming



- **Typical techniques**
  - **Continuous integration (CI)**
    - integration test of codes entered into a version control system
    - code changes in a common code base, where run automatic tests and checks
    - automated build and testing: after every change to the code, tests are automatically run and the code is compiled (if possible)
    - CI tools: Jenkins, GitHub Actions, GitLab CI/CD, CircleCI
    - preparation for continuous delivery (CD): CI is the basis for continuous delivery (CD), automatic preparation of code changes for deployment

Extreme programming



- **Typical techniques**
  - **Refactoring**
    - already tested, working code can be improved, but its functionality cannot be changed
    - modifying existing code without changing its functionality
    - aims to improve code readability, maintainability, and efficiency
    - no new features: refactoring does not change the behavior of the code, it only improves its structure
    - better readability: results in cleaner, more understandable code
    - increase maintainability: it is easier to make future modifications and bug fixes
    - reduce code duplication: remove redundant codes

Kanban

- **Visually-based, continuous development (Lean, Agile) method**
- To managing group work

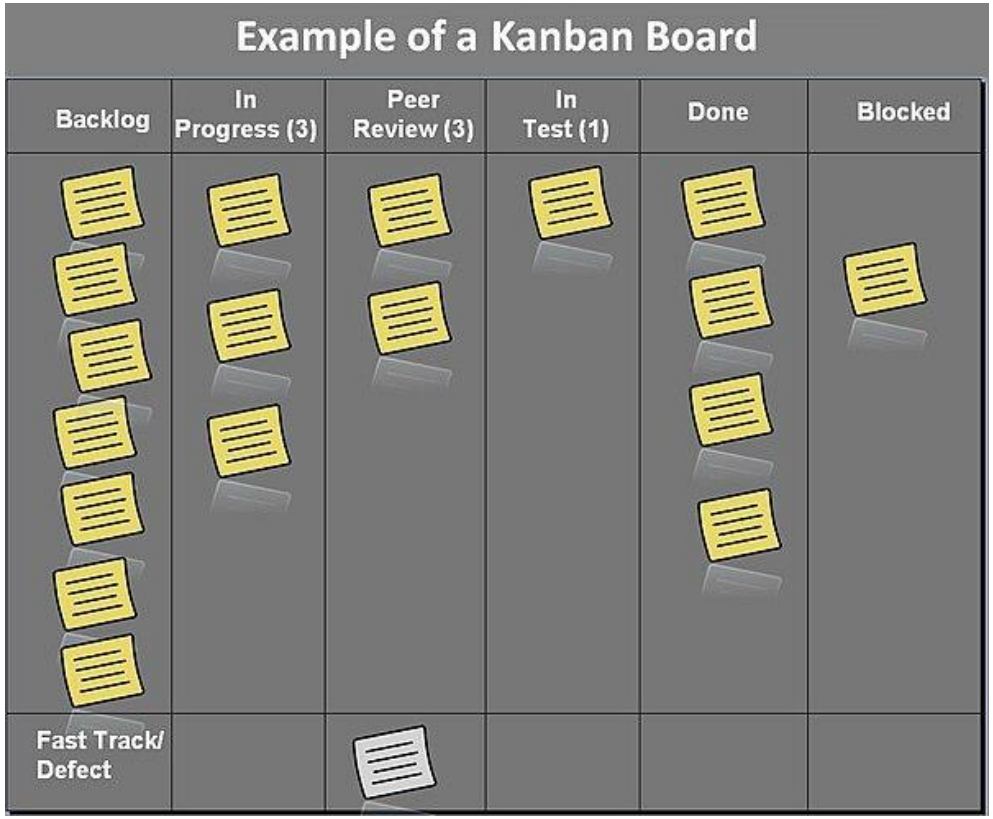
- Properties:
  - Continuous Delivery
    - there are no predefined iterations, the work is continuous
  - Visualization
    - tasks are managed on a Kanban board, which is divided into columns (e.g. "To Do", "In Progress", "Done")
  - WIP (Work In Progress) limit
    - determines how many tasks can be in a given stage at the same time, thus reducing congestion

Kanban

- Properties:
  - Pull system
    - teams select tasks when they are ready for them, rather than assigning them in advance
  - Measurable performance
    - For example, the process can be optimized based on cycle time and lead time
- Apply
  - if continuous development is needed
  - if visual transparency is needed to manage tasks
  - if the workload is variable and a more flexible process is needed than Scrum sprints

Kanban

- Kanban board
  - the team moves tasks from left to right as the process progresses



Scrum vs. Kanban

Properties	Kanban	Scrum
Iteration length	continuous	1-4 week sprints
Work management	pull-based	sprint backlog
Priority change	flexible	difficult during sprinting
Roles	slack	pre-defined

Lean

- Originates from Toyota's **Lean manufacturing philosophy**
- Its purpose is to
  - increasing the efficiency of the software development process
  - eliminating unnecessary processes, minimizing waste
  - maximizing customer value
  - Toyota, Amazon, Google
- Main principles
  - **Eliminate waste**
    - elimination of unnecessary work
    - unnecessary code and functions
    - too much documentation that no one reads
    - long waiting times (e.g. testing, approvals)
    - communication problems

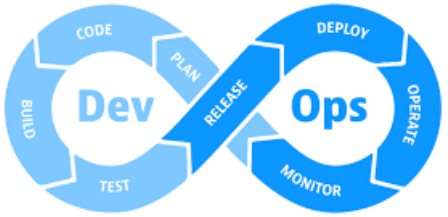
Lean

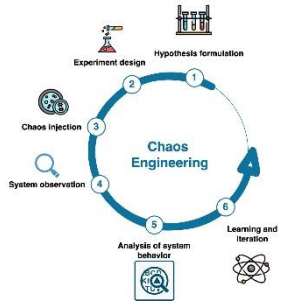
- Main principles
  - **Rapid feedback and decision-making (create knowledge)**
    - continuous feedback from customers and within the team
    - developers have independent decision-making capabilities
    - using the MVP (Minimum Viable Product) principle: first a minimal working version is built
  - **Incorporating quality into development (best quality in)**
    - automated testing and test-driven development (TDD)
    - fixing bugs during the development phase, not afterwards
    - code review for early detection of problems

Lean

- Main principles
  - **Delayed in decision making**
    - no hasty decisions, collect data and only make decisions when sufficient information is available
    - no allocated resources too early
  - **Empower your team**
    - employees are given autonomy to make their own decisions
    - self-organizing teams that respond quickly to changes
  - **Optimize the whole system**
    - focusing not only on development, but also on business goals and the entire process
    - optimizing the operation of the entire organization, not just individual sub-processes

Other models

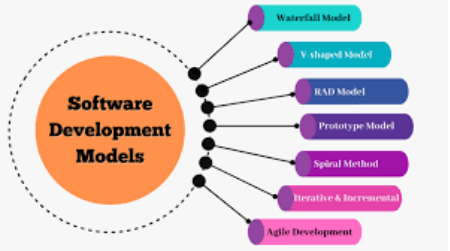




- **DevOps**
  - **Development + Operations**
  - aims to coordinate development (Dev) and operations (Ops) teams
  - developers and operators work together, using automated tools to speed up processes

- **Chaos Engineering**
  - **tests the resilience of systems with simulated failures**
  - "What happens if a server goes down? What if the database suddenly becomes inaccessible?"
  - instead of having to find out in real life, **they artificially induce failures** to prepare the system for such situations

Practical application



- **In practice, the application of these methodologies is mixed**
  - if you need to react quickly to changes → Agile
  - if continuous updates are required → CI/CD, DevOps
  - if large, scalable system → Microservice
  - if security is important → DevSecOps
  - if efficiency is the main thing → Lean development

- Some big companies
  - Google: agile, DevOps, CI/CD
  - Amazon: DevOps, CI/CD, Lean
  - Microsoft: DevOps, CI/CD, Agile, Lean,
  - Facebook: CI/CD, DevOps, Agile, Microservices
  - Tesla: Agile, DevOps, Lean, TDD

Software Code of Ethics

- During software testing, participants may have access to confidential information
  - the code of ethics is needed to do not use information unauthorizedly

- The points of the code are as follows:
  - **Public interest**
    - for qualified testers must be consistently in accordance with the public interest to act

Software Code of Ethics

- **Client and employer**
  - the qualified for software testers have to do their customers and their employers the best, but not against the public interest
- **Product**
  - the qualified software testers have to ensure the tested, ready for delivery product or system meets the highest professional standards
- **Opinion**
  - the qualified for software testers irreproachable and independent have to stay at professional when forming an opinion

Software Code of Ethics

- **Management**
  - the qualified software test managers and leaders have to support the ethical code towards to software testing management towards
- **Profession**
  - the qualified software testers have to promote the reputation and integrity of the profession in accordance with the public interest

Software Code of Ethics

- **Colleagues**
  - the qualified software testers have to act correctly and supportively towards their colleagues and facilitate collaboration with software developers
- **Personal**
  - the qualified software testers have to lifelong learn their profession and must ensure the code of ethics becomes part of their work

Thank you for your attention !