Introduction To Software Engineering

Luca Onniboni, unsplash

A short history

Software Development in the early 1960s:

- Programming is art[1]
- Limited hardware → optimization is crucial
- Software development: Design and implementation of algorithms.

Software Development in the late 1960s:

- Problems to be solved by software getting more and more complex.
- Methods used for software development do not scale.
- Systematic methods for developing software urgently needed.
- Programming becomes science.

[1] Donald Knuth: The Art of Computer Programming

The First Software Engineer

> I fought to bring the software legitimacy so that it—and those building it—would be given its due respect and thus I began to use the term 'software engineering' to distinguish it from hardware and other kinds of engineering[…][1]

- Hamilton worked on the software of the Apollo Guidance Computer in the 1960s.
- First time software considered critical for mission success
- Foundation of our modern software engineering processes

[1] Margaret Hamilton: First Software Engineer, 2018.

What Is Software Engineering?

> Software Engineering is the application of systematic, quantifiable processes to the development and evolution of software products for customers, subject to cost, schedule, and regulatory constraints.[1]

Four key driving forces

1. Customers (/Users): Drive requirements ("What to build")
2. Processes: Organizing the development and team interactions ("How to build")
3. Product: A deliverable from a software project ("The result")
4. Constraints: External forces, e.g., regulations, cost, time ("To take into account")

[1] ACM and IEEE Computer Society, 2013. Computer Science Curricula 2013, New York, NY, USA (https://dx.doi.org/10.1145/2534860).

The Software Crisis?

- Standish Group's Chaos Report (2015): 29% of software projects are considered successful, 52% are challenged[1] , and 19% are considered failed.
- Software Engineering Institute (2018): Project failure 5% to 15% for small projects, 10% to 20% for medium projects, and up to 50% for large projects.
- Project Management Institute (2020): Only 64% of software projects met their original goals and business intent.

> Software development is becoming increasingly complex and requires sophisticated methodology to succeed!

[1] i.e., completed with some degree of dissatisfaction

Processes

🕮 Software Process

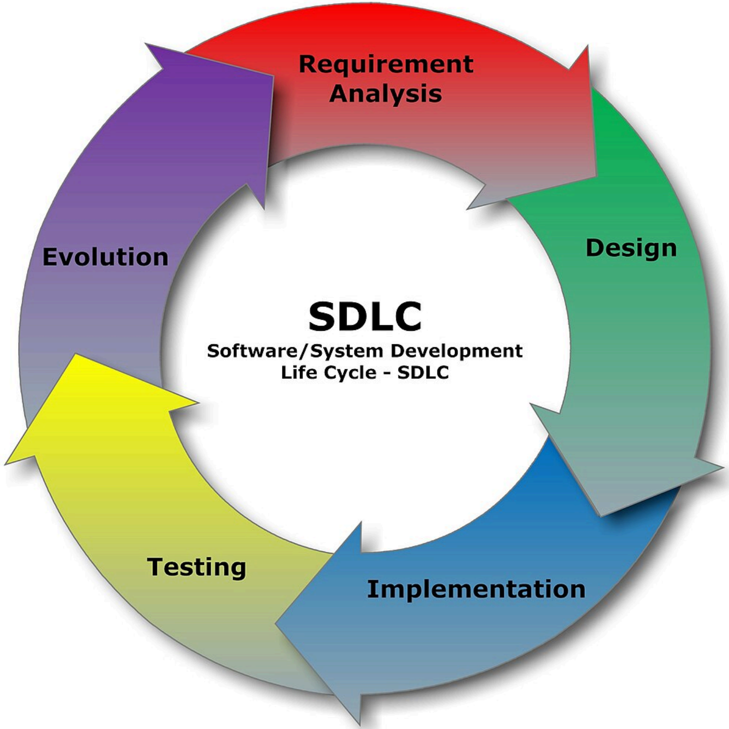
Structured set of activities required to develop a software system.

Many approaches exist, but all involve:

- Specification: Defining what the system should do.
- Design: Defining the organization of the system.
- Implementation: Technical realization of the system.
- Validation: Checking that it does what the customer wants.
- Evolution: Changing the system in response to customer needs.

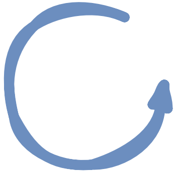
The Software Development Lifecycle (SDLC)

A process used by software development teams to plan, design, build, test, deploy, and maintain software applications.



Cliffydcw, CC BY-SA 3.0, via Wikimedia Commons

Plan-driven vs. Agile processes



- Plan-driven processes are processes where all the process activities are planned in advance and progress is measured against this plan (e.g., waterfall).
- In agile processes, planning is incremental, and it is easier to change the process to reflect changing customer requirements (e.g., SCRUM).

> ! Details on both methodologies are not part of this lecture!

🕮 Waterfall Model

Linear sequential approach to software engineering. The development process is divided into distinct phases, which are completed one after the other.

- Constitutes a plan-driven process.
- Each phase must be completed before moving on to the next phase.
- Drawback: Difficult to accommodate change after the process is underway.

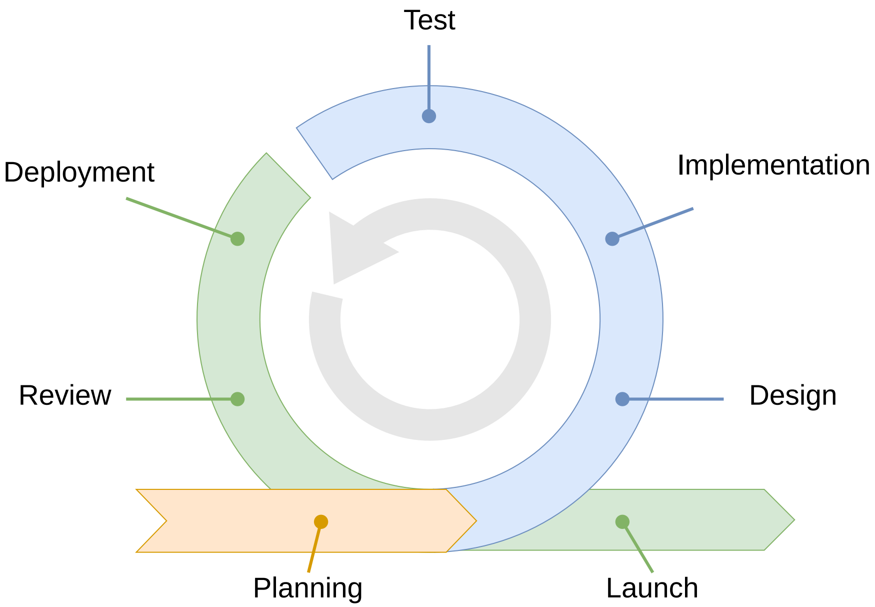
📖 Agile Manifesto

A set of guidelines and principles for software development that prioritize flexibility, collaboration, and adaptability over rigid processes and plans.

- Created in 2001 by a group of developers who were frustrated with the traditional, waterfall-style approach to software development.
- Composed of four key values and twelve principles.
- It has since been adopted by many software development teams and has given rise to a range of agile methodologies, such as Scrum or Kanban

SDLC & Agile Development

Agile software development is **iterative**:



→ What is done in each step?

Planning

- Requirements are collected and formalized
- Requirements are prioritized with customers / stakeholders
- Work is estimated and distributed

Scrum: Product and Sprint Backlog, Sprint Planning, Backlog Grooming

🛠 Example: Concepts

- Requirements Engineering
- Requirements Specification
  - User Stories
  - Use Cases
- Requirements Analysis
  - Estimation
  - Priorization

Design

- Design includes *software architecture* of larger components
- Implementation details are planned according to user stories and acceptance criteria

Scrum: Sprint Planning, Sprint

🛠 Example: Concepts

- Architecture patterns
  - Model-View-Controller
  - Onion Architecture
  - Hexagonal Architecture
- Class (and other) diagrams
  - UML / ER / BPMN
- Design patterns
  - Visitor, Singleton, Observer, Proxy, …

Implementation

- Story is implemented according to specification
- Unit tests are written as part of the implementation

Scrum: Sprint

🛠 Example: Concepts

- Version Control (e.g., Git)
- Test-driven development
- Unit testing with Mocks, Stubs & Co
- Automation of tasks with CI/CD

Test

- Software is continuously tested during implementation
- Integration tests may be added if not done in implementation
- End2End tests may be added if not done in implementation
- Acceptance criteria are tested and verified

Scrum: Sprint

🛠 Example: Concepts

- Test frameworks
- Behavior Driven Development (E2E- and Acceptance-Testing)

Deployment

- Continuous Integration (CI)
- Continuous Deployment or Delivery (CD)
- Automation

Scrum: Sprint

🛠 Example: Concepts

- Gitlab CI with Pipelines
- Containerization
  - Docker / Docker Compose
- Container Orchestration
  - Kubernetes (K8S)
- Infrastructure As Code (IAC)
  - Terraform, Ansible

Review

- Implementation is validated with stakeholders (users, customers)
- Feedback is collected and added to Product Backlog
- Processes are discussed within Team in Sprint Retrospective

Scrum: Sprint Review, Sprint Retrospective

Launch

- Software is productive and available to users

Scrum: After n sprints or continuously

Different Approaches To SE

Traditionally, Software Engineering courses focus on **Planning**, **Design** and **Test**.

- Requirements Engineering
- Architecture (UML, BPMN, SysML, ER diagrams)
- Design patterns ("Gang of Four" aka "Observer", "Visitor", "Proxy")

This course includes traditional topics, but more focus is on **Implementation**, **Test** and **Deployment**.

- Version Control
- Containerization
- Testing (unit-, integration- and end2end)
- DevOps and CI/CD

🎓 Why? → Employers highly value knowledge of modern concepts such as Containerization, Devops and CI/CD; and traditional topics have more material for self-study (e.g., UML & Co).

Case Study: A productive AI product

Project Background

The data science department of a company selling spare parts for coffee machines developed a proof-of-concept chatbot using OpenAI’s ChatGPT API in a Jupyter Notebook. It provides part suggestions based on user input in natural language.

Project Target

A product team is tasked with **bringing the chatbot to production**. It shall be useable on the companies website.

Questions / Engineering Topics

- What does "useable on the companies website" mean?
- Does the PoC solution scale with respect to API costs and performance?
- Is the code maintainable? (code modularity, architecture, tests)
- … and many more!

Case Study: Scientific Software Development

Project Background

A company sells software with efficient algorithms for operations research (e.g., timetable scheduling, rostering). The software is written in C and many parts have been in use and optimized for decades. Recently, they learned about a new, improved algorithm published by a research group.

Project Target

They plan to add the algorithm to their software package as soon as possible. Unfortunately, a severe bug was introduced in the software and they lost many customers to a competitor who added the new algorithm only few weeks later.

Questions / Engineering Topics

- Why was the bug not discovered?
- How could the competitor be so fast? Luck?
- What could the company do different in the future?

Competences

🎓 You should have acquired the following competencies:

- Explain the main goals of Software Engineering
- Understand that processes are involved in producing a software system
- Know what the stages of the Software Development Life-Cycle are