# Software Engineering Cheatsheet

Edoardo Riggio

June 26, 2021

Operating Systems - SP. 2021
Computer Science
Università della Svizzera Italiana, Lugano

# Contents

# 1 Software Development Life Cycle

## 1.1 Waterfall Model

This is a family of models. They identify phases and activities, forcing a linear progression from a phase to the next. No return is allowed (i.e. you cannot go from a phase to one before it).

## 1.2 Royce's Waterfall

### 1.2.1 Feasibility Study

This is a cost/benefits analysis.

It determines whether the project should be started, explore possible alternatives and needed resources.

Output → **Feasibility Study Document** - This document contains a preliminary problem description, scenarios with alternatives, and costs for these alternatives.

### 1.2.2 Requirement Analysis and Specification

This is a domain analysis.

It is needed in order to identify requirements and derive specifications for the software. It requires an interaction with the user, and an understanding of the properties of the domain.

Output → **RASD** (Requirements Analysis and Specification Document)

### 1.2.3 Design

Defines the software architecture.

It determines all the aspects of the software, such as: components/modules, relations among components and interactions among components. It also enables concurrent development and separates responsibilities.

Output → **Design Document**

### 1.2.4 Coding and Unit Testing

It is used to produce and test code.

Each module is implemented using the chosen programming language. Each module is then tested in isolation by the module's developer.

Output → **Software** - Composed of classes and modules.

#### 1.2.5 Integration and System Test

All of the modules are integrated.

Modules are integrated into (sub)systems and all of the latter are tested. A complete system test is needed in order to verify the overall properties of the software. Sometimes **alpha tests** and **beta tests** are used.

Output → **Software** - Integrated modules.

#### 1.2.6 Deployment

The goal is to distribute the application and manage the different installations and configurations at the clients' sites.

#### 1.2.7 Maintenance

These are all the changes that are done after the delivery. It often is more than 50% of the total cost.

Maintenance can be of three types:

- **Corrective Maintenance**

  It is made up of diagnosing and fixing errors, which are possibly found by users.

- **Adaptive Maintenance**

  Modify the system to cope with changes in the software environment.

- **Perfective Maintenance**

  Implement new or changed user requirements.

### 1.3 Spiral Model

In a spiral model, **prototypes** are produced at each new spiral. Prototypes are useful because they are an approximation of the model of the application. There exist two types of prototypes:

- **Throw-Away**

  There is little to no reuse of the components of the prototype.

- **Evolutionary**

  The components of the prototype are reused even in the final product.

## 1.4 Boehm's Spiral Model

### 1.4.1 Determine Goals, Alternatives and Constraints

This enables to identify specific objectives, alternatives and constraints for that specific phase.

### 1.4.2 Evaluate Alternatives and Risks

Access each alternative in order to reduce the potential risks.

### 1.4.3 Develop and Test

Artifacts are produced for the next phase. This also includes code.

### 1.4.4 Plan

Review the project and plan the next phase of the spiral.

## 1.5 Synchronize and Stabilize

What people are doing as individuals and as members of parallel teams is continually **synchronized**, and the product increments are periodically **stabilized** as the project proceeds.

### 1.5.1 Planning Phase

This phase is itself divided into three more phases:

- **Vision Statement**

  The product and program management create a vision statement defining the goals for the new product. Product features based on costumer input are identified and prioritized.

- **Specification Document**

  This is written by the program management and the developer teams. It defines the functional features, architectural issues, and interdependent components. It does not cover all the details of each feature, and does not lock the feature set of the project.

- **Schedule and Team Formation**

  The program management defines the schedule and arranges the teams. The teams are small and have a 1:1 ratio between developers and testers. The project is divided into sequential sub-projects containing priority-ordered features. The schedule is divided into milestone cycles.

### 1.5.2 Development Phase

All teams have to go through a complete cycle of development, which includes: development, integration, testing and fixing problems.

Teams and individuals work in parallel, and revise the feature set as they learn.

Whenever developers commit code, regression testing is automatically performed. Debugging and synchronization between teams happen on a daily basis.

### 1.5.3 Stabilization Phase

In this phase program managers coordinate and monitor the customer's feedback. Developers perform final debugging and code stabilization.

The **golden master disks and documentation** are prepared for final release.

## 1.6 Extreme Programming

### 1.6.1 Release Planning

This is made up of four different phases:

- **User Stories**

  User stories are written by customers as things that the system is expected to accomplish. They are short, three sentences format of written text without technicalities.

- **Architectural Spike**

  Find the simplest system metaphor which allows to explain the system to new people without resorting to huge documents. The design must be kept as simple as possible. Naming of classes and methods must be kept simple and consistent.

- **Project Velocity**

  This is a measure of how much work is getting don on the project. It is also used as an indicator of how many user stories can be done before a deadline.

- **Release Plan**

  The development team estimates the ideal developing time needed for each user story. The customer must indicate which of the user stories must have higher priority.

### 1.6.2 Iteration

This is made up of several phases:

- **Iteration Planning**

  During iteration planning meetings, customers select the user stories from the release plan. User stories and failed tests are broken down into programming tasks. These tasks are done by developers.

- **Stand-up Meetings**

  These meetings are scheduled at the beginning of every day.

- **CRC Cards**

  Class, Responsibilities and Collaboration should be used to design systems. **Responsibilities** represent anything an object knows or does, while **collaborators** represent the classes to collaborate with in order to fulfill the responsibilities. Individual cards are used to represent objects.

- **Unit Test First**

  Unit tests are created before any coding activity. This is known as **Test Driven Development** (TDD).

- **Pair Programming**

  All code is created by two people working together at a single computer. One of the couple is the **navigator**, which reviews the code, and one is the **driver**, which writes the code.

- **Continuous Integration**

  Developers should be integrating code every few hours. All unit tests must pass in order to detect any compatibility problem early.

### 1.6.3 Acceptance Tests

Here we have two phases:

- **Creation**

  Acceptance tests are created from user stories. The customer specifies some scenarios in which to test for a user story. These are used as regression tests prior to production release.

- **Customer Approval**

  Customers are responsible for verifying the correctness of the acceptance tests. Customers decide which failed tests are of highest priority. A user story is not considered completed until it passes all acceptance tests.

## 1.7 Scrum

Scrum is a framework that has **three roles** – product owner, scrum master and scrum team, **three ceremonies** – sprint planning, daily scrum meetings and sprint review, and **three artifacts** – product backlog, string backlog and burndown chart.

### 1.7.1 Roles

There are three different roles in Scrum:

- **Product Owner**

  This could be a customer representative or a customer proxy. He devises the vision, business plan and releases. He is also responsible for defining the features of the product prioritized on the return of investment. He also builds the product backlog and leads the scrum planning.

- **Scrum Master**

  Does whatever it's necessary to help the team be successful, by supporting the practices of Scrum. He manages conflicts within the team, identifies and prioritizes impediments, and implements remediation plans. The Scrum master is not a project manager, he does not assigning tasks to people.

- **Scrum Team**

  A team is typically composed by 5 to 10 people. These teams are cross functional and include all the expertise needed. Teams are also self-organizing. They set their sprint goals, and do everything within the boundaries of a project to achieve them.

### 1.7.2 Ceremonies

There are three ceremonies in Scrum:

- **Sprint Planning**

  This takes place at the beginning of each sprint, and sets the goals of the sprint. The project owner and the team review the product backlog, and discuss goals. The items selected are then broken down into tasks and added to the sprint backlog.

- **Daily Scrum Meetings**

  These are short, 15 minutes meetings, where tams report what they have done and what they are going to do. After the meeting, the Scrum master updates the burndown chart with the data of the completed tasks.

- **Sprint Review**

  The team demos the work they have done to everyone in the three roles. Presentations are not allowed.

## 1.8 Agile

The 12 rules of the Agile manifesto are the following:

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

## 1.9 Hacker Way

In the hacker way, command is decentralized. This means that subordinate leaders must use their initiative in order to accomplish tasks supporting their senior's intent. The goals of a layer are set by the immediate upper layer, and this is a continuous feedback loop. Three levels of command are:

- **Senior Leadership**

  It sets the strategic long-term direction, and ensures that the feedback engine runs smoothly. It is also responsible for the overall people management, meaning acquiring new capabilities, expelling obsolete capabilities and react to external changes.

- **Management**

  It provides operational context (goals) by linking strategy with tactics. They also devise the constraints that are aligned with the strategy of the company under which the developers operate. There is trust towards developers, and it provides them with all the software and hardware needed.

- **Developers**

  They are responsible for the tactical level of operations. The teams are composed of about 10 developers.

The decision is data-driven, and every layer must have a data scientist to help analyze data.

The philosophy of this method is to deploy, and if it fails, fix it. Some companies release versions of the product only to a subset of the users in order to test it. If the version is broken, they simply rollback to the previous working version.

# 2 Requirement Engineering

Requirement engineering analyses the context in which the problem to be solved lies, and finds out what needs to be developed, purchased or installed in order to fix the problem.

It is divided into two overlapping sets. The first is the **problem world's phenomena** set, and the second is the **machine solutions' phenomena** set. The intersection of these two sets is called **shared phenomena**.

There is also the distinction to be made between **system-as-is**, which is the system as it exists before the machine is built into it, and the **system-to-be**, which is the system as it should be when the machine will be built and operated in it.

- **System-As-Is**

  Contains the problems, opportunities and domain knowledge.

- **System-To-Be**

  Contains the who, what and why.

## 2.1 Categories of Requirements

### 2.1.1 Environmental Phenomena

This set is subdivided into:

- **Domain Properties**

  These are descriptive statements about the problem world that is expected to hold regardless of how the system will behave.

- **Assumptions**

  Statements that need to be satisfied by the environment and formulated in terms of environmental phenomena.

- **System Requirements**

  Prescriptive statements to be enforced by the **software-to-be**, possibly in cooperation with other system components, and formulated in terms of environmental phenomena. They are divided into:

  - **Functional Requirements**

    They define the functional effects that the **software-to-be** is required to have in its environment. They may also refer to environmental conditions under which such operations should be applied. They are also known as **features**.

  - **Non-Functional Requirements**

    They define constraints on the way the **software-to-be** should satisfy functional requirements or on the way it should be developed. There could be several such requirements, such as:

    * **Quality Requirements**
    * **Compliance Requirements**
    * **Architectural Requirements**
    * **Development Requirements**

### 2.1.2 Shared Phenomena

These are:

- **Software Requirements**

  Prescriptive statements to be enforced solely by the **software-to-be** and formulated only in terms of shared phenomena.

## 2.2 Requirements Engineering Process

### 2.2.1 Requirements Elicitation

This is the discovery of candidates requirements and assumptions that will shape the **system-to-be** based on the weaknesses of the **system-as-is**. It requires a preliminary phase of knowledge acquisition and discovery about the application domain, the organization and the **system-as-is**. This phase can be of two types:

- **Artifact Driven**

  Collect, read and synthesize relevant documentation about the **system-as-is**. Submit a list of specific questions to selected stakeholders with a list of possible answers and a brief context. Illustrate a typical sequence of interactions among system components that meet an implicit objective with concrete examples. Show positive/negative and normal/abnormal scenarios. Show mock-ups and prototypes of the final product to help understand and clarify its features.

- **Stakeholder Driven**

  Organize interviews with specific stakeholders to target the information we want to acquire. Interviews can be **structured**, which follow a sequence of pre-defined questions, or **unstructured**, where there is only a free informal discussion with the stakeholder about the **system-as-is**.

### 2.2.2 Requirements Evaluation

This step is for the evaluation of the elicited requirements, in order to obtain a set of low-risk, conflict-free requirements that stakeholders agree on. This step too is divided into several parts:

- **Conflict Handling**

  Negotiation may resolve conflicts there may be between stakeholders.

- **Risk Analysis**

  What is known as **functional risk** concerns the inability of the product to deliver the required services. **Non-functional risk**, on the other hand, concerns the inability of the product to deliver the required quality.

- **Prioritization**

  Some requirements often need to be prioritized. Unexpected circumstances might force re-planning of the development. Requirements with lower priority should be weakened or even dropped if necessary.

### 2.2.3 Requirements Specification

This is to detail, structure and document the agreed characteristics of the **system-to-be** as they emerge from the evaluation activity. The output docu-

ment of this phase is the **requirements document**, which contains objectives, domain properties, assumptions... This document should be easy to understand. These requirements can be of three different types:

- **Informal**

  Make use of **natural language** to document all of the agreed requirements. Ambiguity and noise are inherent to natural language, and may lead to different interpretations.

- **Semi-Formal**

  Make use of diagrammatic notations to substitute or complement the natural language specifications. All items and relationships and items are declared formally. **UML** (Unified Modeling Language) is the industry standard for documenting object oriented systems.

- **Formal**

  Make use of formal notations that have a defined syntax, semantic and proof theory, in order to describe and derive requirements.

#### 2.2.4 Requirements Consolidation

In this steps we detect defects, report them, analyze their causes, and undertake the appropriate actions in order to fix them and ensure quality of the requirements. There are four steps to this process:

- **Inspection Planning**

  Determine the size and members of the inspection team, schedule and formats of the reports.

- **Individual Reviewing**

  Each inspector reads the requirements document or parts of it to look for defects.

- **Defect Evaluation at Review Meetings**

  The defects found by each inspector are collected and discussed to recommend appropriate actions.

- **Requirements Document Consolidation**

  The requirements document is revised to address all of the concerns expressed in the inspection report.