UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SOFTWARE ENGINEERING – LECTURE 02

SOFTWARE PROCESSES AND SOFTWARE QUALITY

Proff. Sergio Di Martino, Luigi Libero Lucio Starace

sergio.dimartino@unina.it, luigiliberolucio.starace@unina.it

https://www.docenti.unina.it/sergio.dimartino, https://www.docenti.unina.it/luigiliberolucio.starace

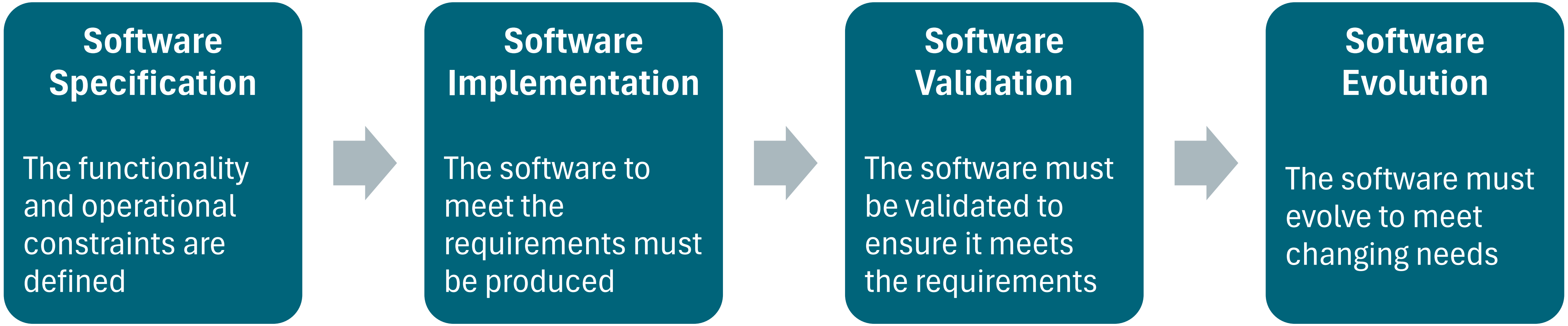
PREVIOUSLY, ON SOFTWARE ENGINEERING

- In the first lecture, we've seen what Software Engineering is and why it's important.
  - *«Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software»*
    - IEEE Standard Glossary of Software Engineering Terminology
  - *«State of the art of developing **quality** software on time and within budget.»*
    - Bruegge B. and Dutoit A. *H. Object-Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall, 1994
- Today, we'll briefly go over the waterfall software process
- Then, we'll focus on the concept of software quality

SOFTWARE PROCESSES

SOFTWARE PROCESSES

- A **software process** is a set of related activities that leads to the production of a software system
- There is **no universal process** that works everytime, and many different processes exist and are used
- All of them include, in some form, the following **fundamental activities**



Software Specification

The functionality and operational constraints are defined

Software Implementation

The software to meet the requirements must be produced

Software Validation

The software must be validated to ensure it meets the requirements

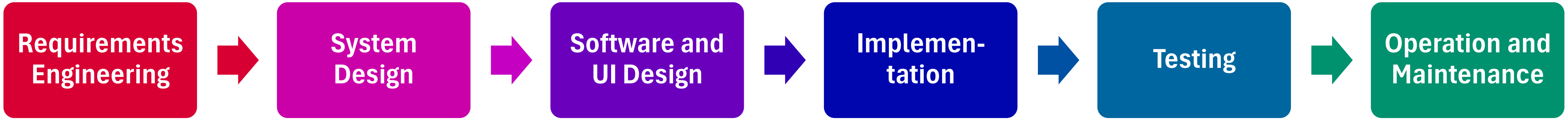
Software Evolution

The software must evolve to meet changing needs

SOFTWARE PROCESS MODELS

- A **Software Process Model** or **Software Development Life Cycle** (SDLC) is a **simplified** representation of a **Software Process**
- A Software Process Model may focus on a particular perspective
- In the Software Engineering course, we’ll discuss some very general process models
  - We’ll start with the so-called **Waterfall model**
  - Later on, we’ll discuss other approaches (e.g.: ***incremental*** or ***agile*** models)

THE WATERFALL SOFTWARE PROCESS MODEL



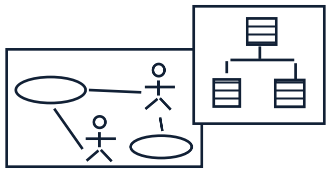
Requirements Engineering → System Design → Software and UI Design → Implementation → Testing → Operation and Maintenance

Requirements collected via:

- Interviews with Stakeholders
- Personas
- Stories and Scenarios

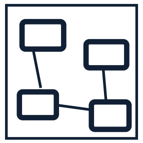
Specified using:

- Use Cases
- Natural Language
- Domain Models
- Mock-ups



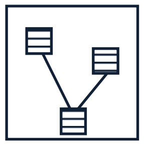
Define System Architecture

- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns



Define Subsystems

- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing

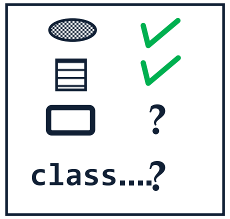


Each Subsystem is implemented

- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing



System is put into practical use

Maintance will be required at some point

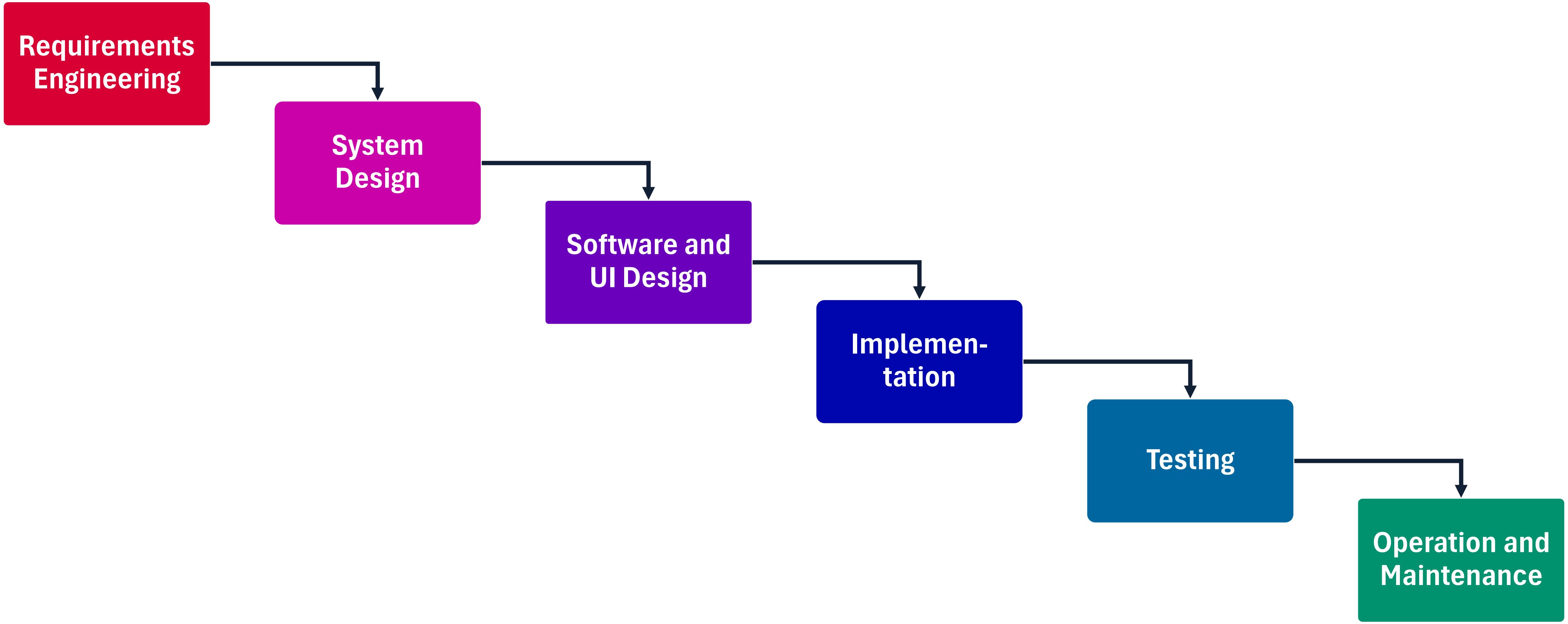
- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

THE WATERFALL SOFTWARE PROCESS MODEL

- Derived from engineering process models used in large military systems engineering [1]
- The software process consists of a number of sequential stages, in a **plan-driven** process
- The result of each phase is a document that is approved (signed-off)
- The following phase cannot start until the result of the previous phase is complete

[1] ROYCE, Winston W. Managing the development of large software systems: concepts and techniques. In: *Proceedings of the 9th international conference on Software Engineering*. 1987. p. 328-338. https://www.praxisframework.org/files/royce1970.pdf

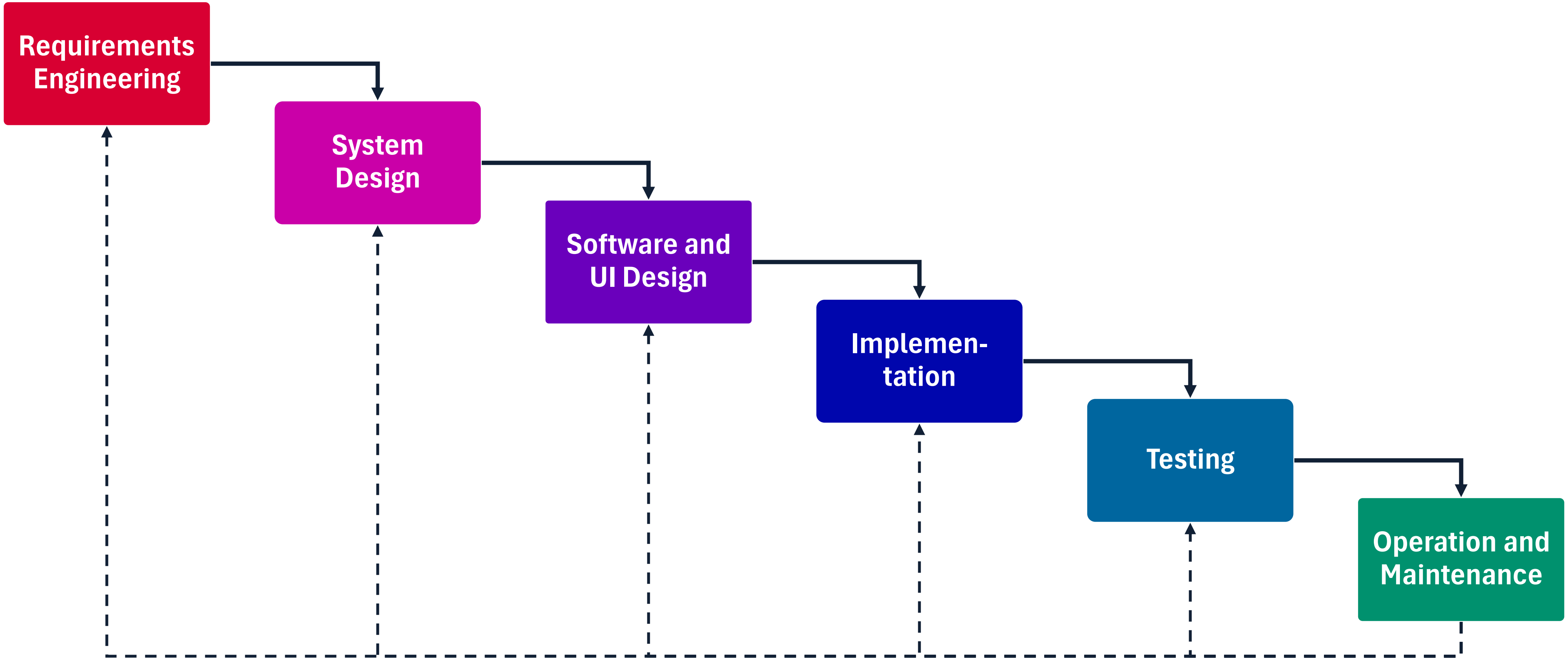
THE WATERFALL SOFTWARE PROCESS MODEL



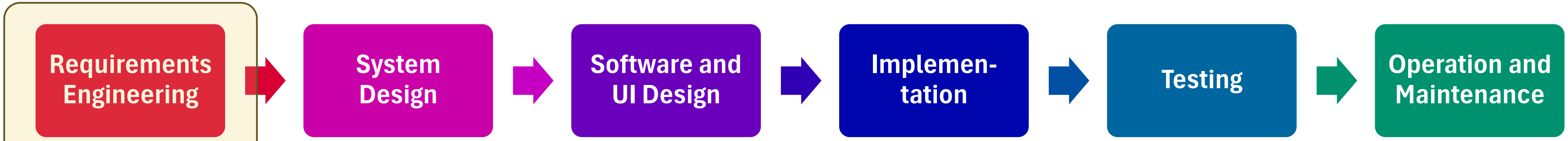
THE WATERFALL PROCESS MODEL

- This rigid, feed-forward approach makes sense for hardware engineering, where high manufacturing costs are involved.
- For software development, these stages may overlap and feed information to each other
  - During system design, problems with requirements may be identified
  - During implementation, problems with software design may be found
  - Requirements may change

THE WATERFALL SOFTWARE PROCESS MODEL



THE WATERFALL SOFTWARE PROCESS MODEL



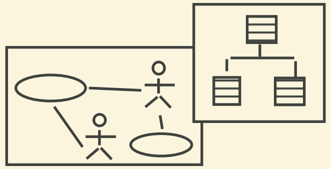
Requirements Engineering

Requirements collected via:

- Interviews with Stakeholders
- Personas
- Stories and Scenarios

Specified using:

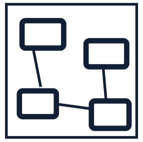
- Use Cases
- Natural Language
- Domain Models
- Mock-ups



System Design

Define System Architecture

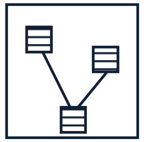
- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns



Software and UI Design

Define Subsystems

- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing



Implementation

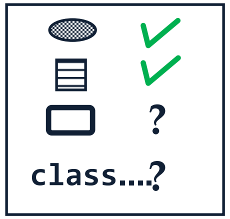
Each Subsystem is implemented

- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

Testing

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing



Operation and Maintenance

System is put into practical use

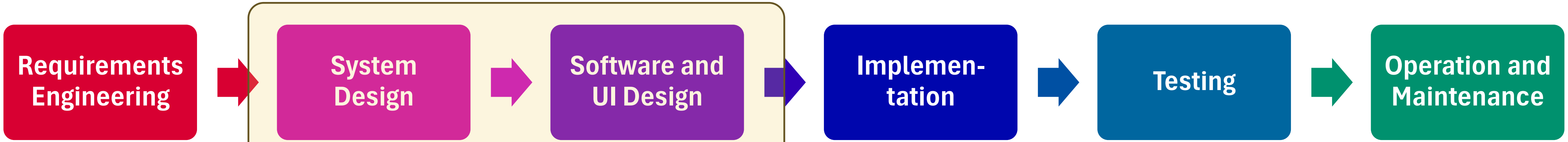
Maintance will be required at some point

- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

REQUIREMENTS ENGINEERING

- Goal: understand **what** the Software-to-be should do
  - Not **how** to implement it!
- Careful analysis of the user’s need and of the problem domain
- Customers, end users, and Software Engineers are involved
- The main output is a **Software Requirements Specification Document**

THE WATERFALL SOFTWARE PROCESS MODEL

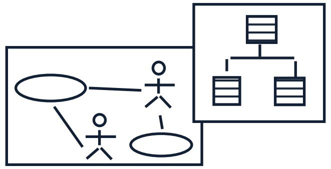


Requirements collected via:

- Interviews with Stakeholders
- Personas
- Stories and Scenarios

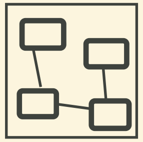
Specified using:

- Use Cases
- Natural Language
- Domain Models
- Mock-ups



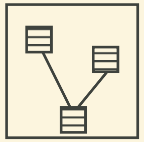
Define System Architecture

- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns



Define Subsystems

- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing



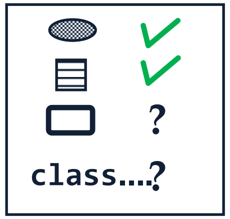
Each Subsystem is implemented

- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

```
class...
class...
class...
```

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing



System is put into practical use

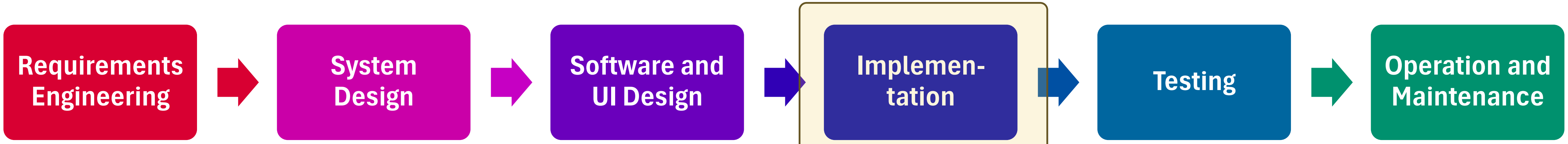
Maintance will be required at some point

- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

SYSTEM AND SOFTWARE/UI DESIGN

- Goal: design an adequate structure (**architecture**) for the software
- Two different levels:
  - **System design**: overall architecture of the system
    - Decomposition in modules and components
    - Allocation of functionalities to modules and modules to hardware components
    - Relationships and collaborations between modules are defined
  - **Software and UI design**: details on how to implement each module
    - Includes lower-level software architecture design (e.g.: what classes will we need?)
    - Includes UI prototyping
- Output is a set of design specifications
  - Often formalized using design languages such as **UML**

THE WATERFALL SOFTWARE PROCESS MODEL

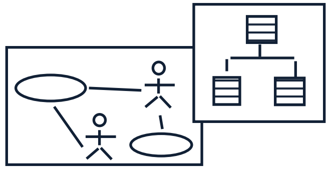


Requirements collected via:

- Interviews with Stakeholders
- Personas
- Stories and Scenarios

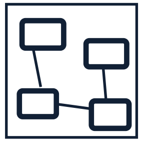
Specified using:

- Use Cases
- Natural Language
- Domain Models
- Mock-ups



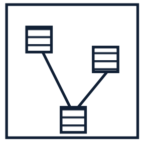
Define System Architecture

- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns



Define Subsystems

- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing



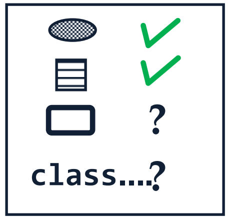
Each Subsystem is implemented

- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

```
class...
class...
class...
```

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing



System is put into practical use

Maintance will be required at some point

- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

IMPLEMENTATION

- Goal: «translate» design specification in a chosen programming language/technology
- Not just any translation, but an **high quality** one
- The resulting code should be
  - Readable
  - Maintainable
  - Reusable
  - Extendable
  - Testable
  - … or, in a single word, «**clean**»

THE WATERFALL SOFTWARE PROCESS MODEL

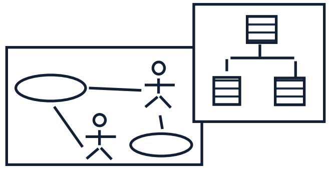
Requirements Engineering

Requirements collected via:

- Interviews with Stakeholders
- Personas
- Stories and Scenarios

Specified using:

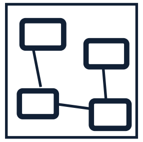
- Use Cases
- Natural Language
- Domain Models
- Mock-ups



System Design

Define System Architecture

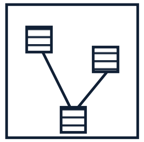
- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns



Software and UI Design

Define Subsystems

- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing



Implemen-tation

Each Subsystem is implemented

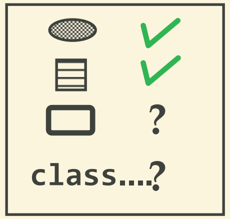
- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

```
class...
class...
class...
```

Testing

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing



Operation and Maintenance

System is put into practical use

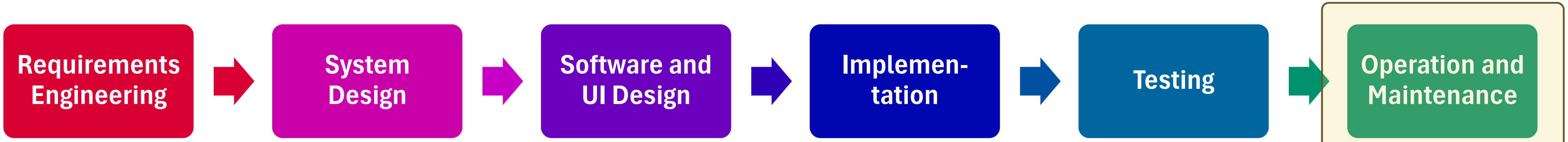
Maintance will be required at some point

- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

VERIFICATION AND VALIDATION (V&V)

- We have an implementation. But does that implementation actually satisfy the needs of users?
- **Verification**: does the system conform to its specification?
  - Have we built the thing right?
  - This is typically done with program testing
    - Running the software in a controlled environment, and checking that its behaviour is correct w.r.t. specifications
- **Validation**: does the system meet the expectation of the customers?
  - Have we built the right thing?
  - This is typically done by defining and running acceptance tests

THE WATERFALL SOFTWARE PROCESS MODEL



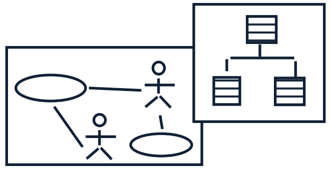
Requirements Engineering

Requirements collected via:

- Interviews with Stakeholders
- Personas
- Stories and Scenarios

Specified using:

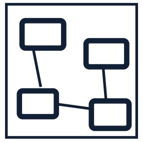
- Use Cases
- Natural Language
- Domain Models
- Mock-ups



System Design

Define System Architecture

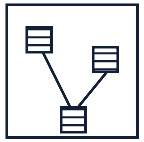
- Requirements are allocated to software sub-systems
- Sub-systems are allocated to hardware resources
- Architectural Patterns



Software and UI Design

Define Subsystems

- Objects required to realize each subsystem are defined.
- Software Design Patterns
- Usability Engineering
- High-fidelity Wireframing



Implementation

Each Subsystem is implemented

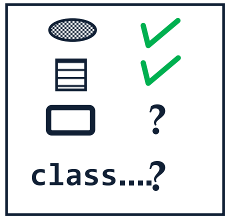
- Source code and other artifacts
- Clean Code
- Frameworks and ORMs
- Focus on Software Quality

```
class...
class...
class...
```

Testing

Ensure the Software satisfies customers

- Code inspections
- Functional Testing (unit, integration, system testing)
- Usability Testing



Operation and Maintenance

System is put into practical use

Maintance will be required at some point

- To fix errors that were not discovered in previous phases
- To adapt the software to changes in requirements on in its environment

OPERATION AND MAINTENANCE

- **Operation:** the system is distributed and installed for the customers, and its put into actual use
- **Maintenance**: the software will change at some point.
  - Customer's needs may change
  - Context of use may change (e.g.: a new law may be put in place, requiring different procedures…)
  - Bugs that slipped through the V&V phase might emerge
- … but these are topics for the **Software Project Management and Evolution** M.Sc. course

SOFTWARE QUALITY

PRODUCT QUALITY

- Product quality is a complex concept
- What does quality mean outside of software?

Car

- Breaks down rarely?
- Consumes little fuel?
- Low maintenance costs?

Watch

- Very precise?
- Water and dust resistant?
- Resistance to scratches?

Mechanical Parts

- Low tolerance?
- Adherence to specification?
- Wear resistant?

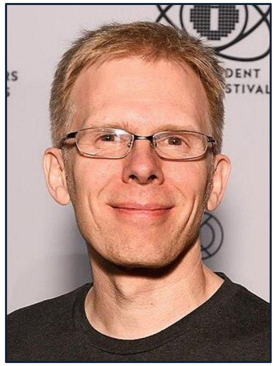
SOFTWARE QUALITY

- What is software quality for you?

```c
float Q_rsqrt(float number){
  long i;
  float x2, y;
  const float threehalfs = 1.5F;

  x2 = number * 0.5F;
  y  = number;
  i  = * ( long * ) &y;
  i  = 0x5f3759df - ( i >> 1 );
  y  = * ( float * ) &i;
  y  = y * ( threehalfs - ( x2 * y * y ) );

  return y;
}
```



This is the **inverse square root** implementation from Quake III Arena, attributed to John Carmack.

The inverse square root $\left({}^{1}/_{\sqrt{x}}\right)$ is largely used in computer graphics (e.g.: to compute the angles of incidence and reflection).

Carmack's approximation was ~4 times faster than doing `(float)(1.0/sqrt(x))`. Many consider it an example of great programming. Can you tell how this code works, though?

Is being efficient high quality?

SOFTWARE QUALITY

- What is software quality for you?

```
/*
 * message.c: functions for displaying messages on the command line
 */

#include <assert.h>
#include <errno.h>
#include <inttypes.h>
#include <stdbool.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>

#include "nvim/vim.h"
```

Some say **OpenBSD** is high quality software, as it is a very stable OS and comparably hard to break. It had only two discovered remote holes in the default install, ever.

Is being very secure against attackers high quality?

Vim is considered by many the best text editor and a high-quality software.

Experts can work extremely quick with it.

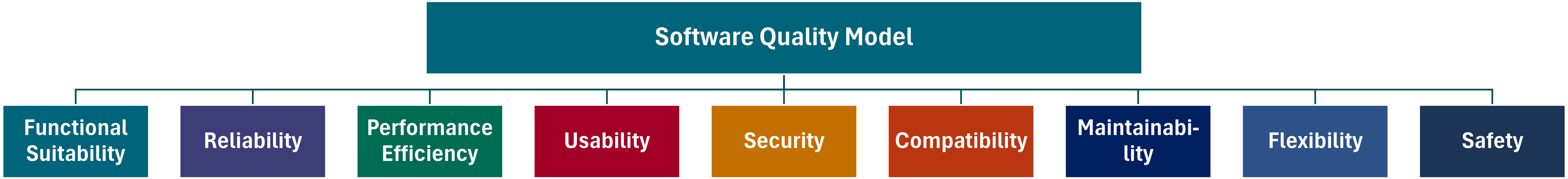
Others may say it’s too hard to learn and therefore low quality.

SOFTWARE QUALITY

- There is not only «one quality», but many different approaches and views
- The ISO/IEC 25002 standard defines **software quality models**
- In the standard, the concept of Software Quality is modelled by:
  - **Product Quality Model**, composed of **9** characteristics (further subdivided into sub-characteristics) that relate to quality properties of the products. The characteristics and subcharacteristics provide a reference model for the quality of the products to be **specified**, **measured** and **evaluated**.
  - **Quality-in-use Model**, composed of **3** characteristics (further subdivided into sub-characteristics) that can influence stakeholders when products or systems are used in a specified context of use.

THE ISO 25002 QUALITY MODEL

ISO 25002: SOFTWARE PRODUCT QUALITY



Software Quality Model

Functional Suitability

Reliability

Performance Efficiency

Usability

Security

Compatibility

Maintainability

Flexibility

Safety

SOFTWARE PRODUCT QUALITY: SUITABILITY

- **Functional suitabiliy** is the degree to which a component or system provides functions that meet stated and implied needs when used under specified conditions.
- This characteristic is composed of three sub-characteristics:
  - **Functional completeness -** Degree to which the provided functions cover all the specified tasks and user goals.
  - **Functional correctness -** Degree to which the product provides accurate results when used by intended users.
  - **Functional appropriateness -** Degree to which the functions facilitate the accomplishment of specified tasks and objectives.

SOFTWARE PRODUCT QUALITY: RELIABILITY

- **Reliability** represents the degree to which the system performs its functions under specified conditions for a specified period of time.
- This characteristic is composed of the following sub-characteristics:
  - **Faultlessness -** Degree to which a system performs specified functions without fault under normal operation.
  - **Availability -** Degree to which a system is operational and accessible when required for use.
  - **Fault tolerance -** Degree to which a system operates as intended despite the presence of hardware or software faults.
  - **Recoverability -** Degree to which, in the event of an interruption or a failure, a product or system can recover.

SOFTWARE PRODUCT QUALITY: EFFICIENCY

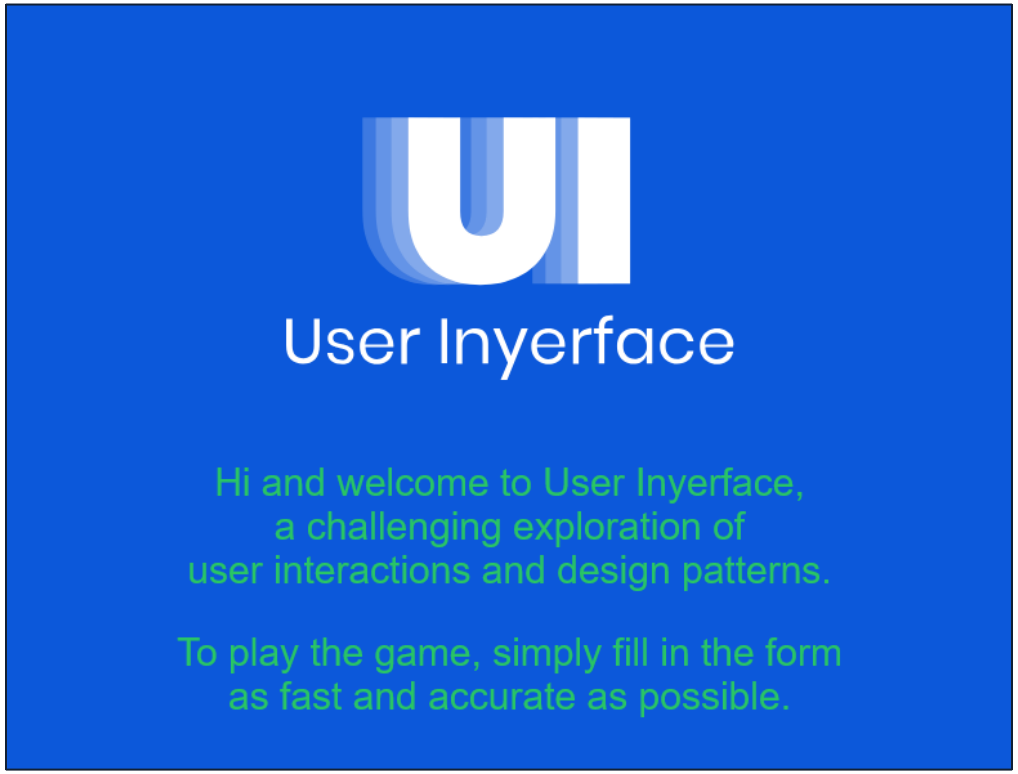
- **Efficiency** represents the degree to which a product performs its functions within specified resource constraints, and is efficient in the use of resources (CPU, memory, storage, energy, …).
- This characteristic is composed of the following sub-characteristics:
  - **Time behaviour -** Degree to which the response time and throughput rates of a product or system meet requirements.
  - **Resource utilization -** Degree to which the amounts and types of resources used by a product or system meet requirements.
  - **Capacity -** Degree to which the maximum limits of a product or system parameter meet requirements.

SOFTWARE PRODUCT QUALITY: USABILITY

- **Usability:** Degree to which a product or system can be interacted with by its users.
- The following sub-characteristics, among others, are included:
  - **Recognizability -** Degree to which users can recognize whether the system is appropriate for their needs.
  - **Learnability -** Degree to which the functions of a product or system can be learnt to be used by specified users within a specified amount of time.
  - **Operability -** Degree to which a product or system is easy to operate and control.
  - **User error protection -** Degree to which a system prevents errors.
  - **Inclusivity -** Degree to which a system can be used by people of various backgrounds (different ages, abilities, cultures, ethnicities, languages, genders, economic situations, etc.).

USABILITY

- We’ll focus on Usability for a good part of Module B
- Usability can make the difference between **success** and **failure**…
- … and even between **life** and **death!**
- Volunteer needed!
  - Let’s sign up on userinyerface.com
  - How hard can that be?



Hi and welcome to User Inyerface,
a challenging exploration of
user interactions and design patterns.

To play the game, simply fill in the form
as fast and accurate as possible.

Home page at https://userinyerface.com

SOFTWARE PRODUCT QUALITY: SECURITY

- Degree to which a system defends against attacks by malicious actors and protects information and data enforcing proper authorization mechanisms.
- Included sub-characteristics include, among others:
  - **Confidentiality -** Degree to which a system ensures that data are accessible only to those authorized to have access.
  - **Integrity -** Degree to which a system ensures that its state and data are protected from unauthorized modification or deletion.
  - **Non-repudiation -** Degree to which actions or events can be proven to have taken place so that the events or actions cannot be repudiated later.
  - **Accountability -** Degree to which the actions of an entity can be traced uniquely to that entity.
  - **Authenticity -** Degree to which the identity of a subject or resource can be proved to be the one claimed.

SOFTWARE PRODUCT QUALITY: COMPATIBILITY

- **Compatibility** represents the degree to which a system can exchange information with other products, systems or components, and/or perform its required functions while sharing the same common environment and resources as other systems.
- This characteristic is composed of the following sub-characteristics:
  - **Co-existence -** Degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product.
  - **Interoperability -** Degree to which a system, product or component can exchange information with other products and mutually use the information that has been exchanged.

SOFTWARE PRODUCT QUALITY: MAINTAINABILITY

- **Maintainability** represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.
- This characteristic is composed of the following sub-characteristics:
  - **Modularity -** Degree to which a software is composed of discrete components, so that a change to one component has minimal impact on the others.
  - **Reusability -** Degree to which a software or module can be used as an asset in more than one system.
  - **Modifiability -** Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading quality.
  - **Testability -** Degree to which test criteria can be established for a system, and tests can be performed to determine whether those criteria have been met.

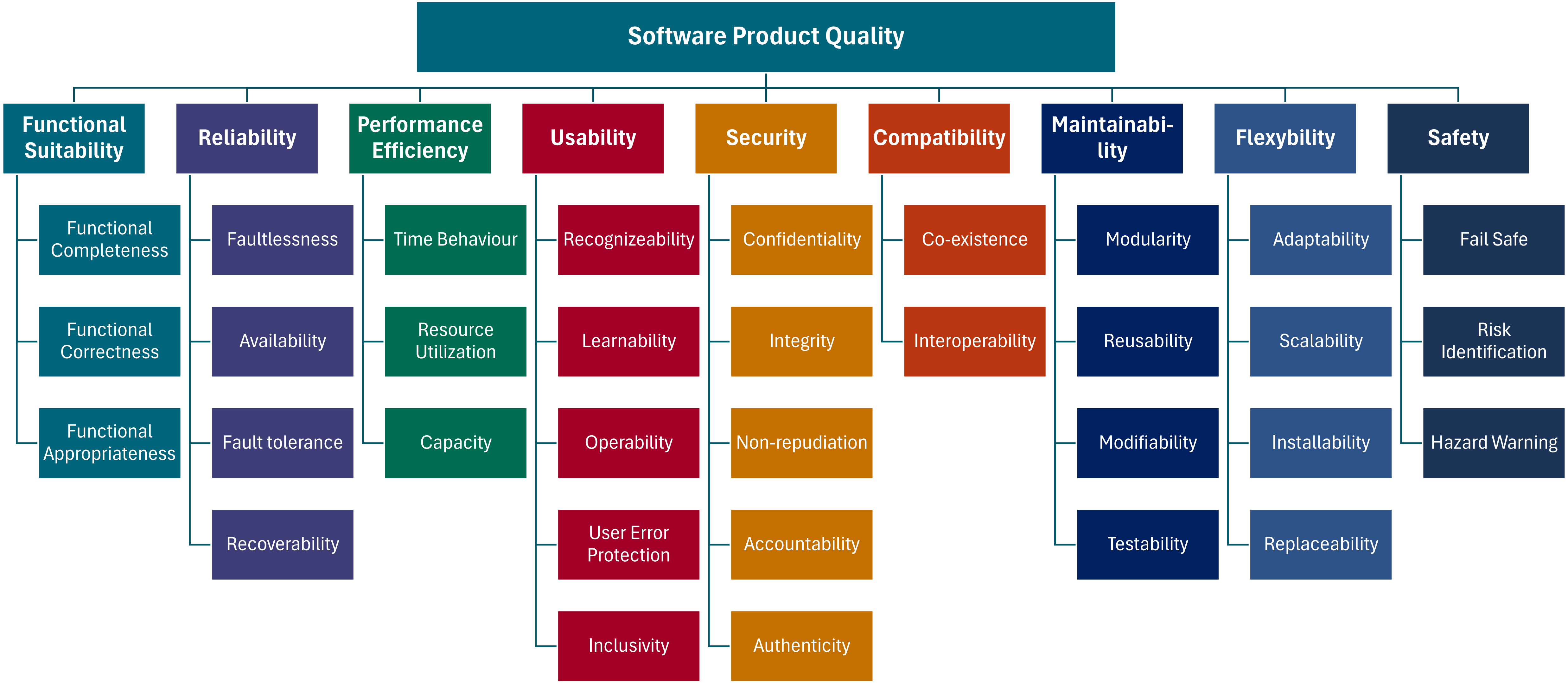
SOFTWARE PRODUCT QUALITY: FLEXIBILITY

- **Flexibility** is the degree to which a product can be adapted to changes in its requirements, contexts of use or operation environment.
- This characteristic is composed of the following sub-characteristics:
  - **Adaptability -** Degree to which a system can effectively and efficiently be adapted for or transferred to different hardware, software or other operational or usage environments.
  - **Scalability -** Degree to which a system can handle growing or shrinking workloads or to adapt its capacity to handle variability.
  - **Installability -** Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled.
  - **Replaceability -** Degree to which a product can replace another specified software product for the same purpose in the same environment.

SOFTWARE PRODUCT QUALITY: SAFETY

- **Safety** represents the degree to which a product avoids a state in which human life, health, property, or the environment is endangered.
- This characteristic includes, among others, the following sub-characteristics:
  - **Fail safe -** Degree to which a product can automatically place itself in a safe operating mode, or to revert to a safe condition in the event of a failure.
  - **Risk identification -** Degree to which a product can identify a course of events or operations that can lead to unacceptable risk.
  - **Hazard warning -** Degree to which a system provides warnings of unacceptable risks to operations or internal controls so that they can react in sufficient time

SOFTWARE PRODUCT QUALITY: OVERVIEW

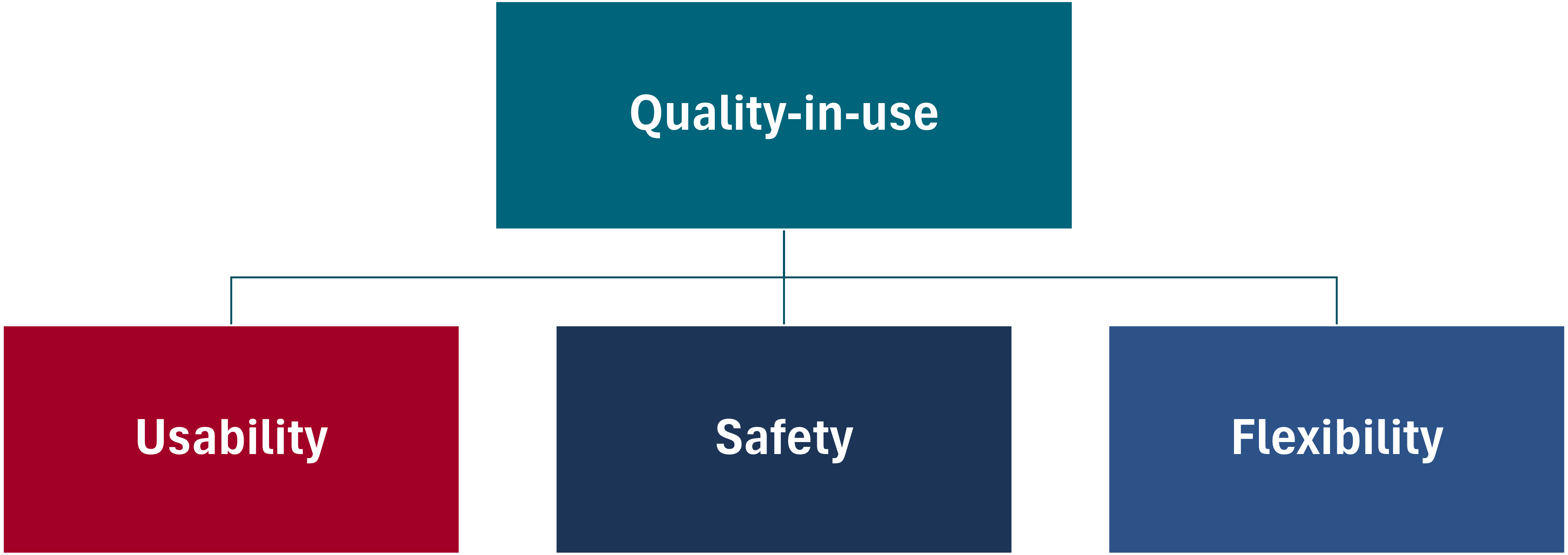


- Software Product Quality
  - Functional Suitability
    - Functional Completeness
    - Functional Correctness
    - Functional Appropriateness
  - Reliability
    - Faultlessness
    - Availability
    - Fault tolerance
    - Recoverability
  - Performance Efficiency
    - Time Behaviour
    - Resource Utilization
    - Capacity
  - Usability
    - Recognizeability
    - Learnability
    - Operability
    - User Error Protection
    - Inclusivity
  - Security
    - Confidentiality
    - Integrity
    - Non-repudiation
    - Accountability
    - Authenticity
  - Compatibility
    - Co-existence
    - Interoperability
  - Maintainability
    - Modularity
    - Reusability
    - Modifiability
    - Testability
  - Flexybility
    - Adaptability
    - Scalability
    - Installability
    - Replaceability
  - Safety
    - Fail Safe
    - Risk Identification
    - Hazard Warning

QUALITY-IN-USE MODEL

- **Quality-in-use Model**, composed of **3** characteristics (further subdivided into sub-characteristics) that can influence stakeholders when products or systems are used in a specified context of use.
- It measures the degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use

QUALITY-IN-USE MODEL



QUALITY-IN-USE: USABILITY

- **Usability** measures the extent to which users can achieve their objectives efficiently and satisfactorily using the system.
- This characteristics is composed of the following sub-characteristics:
    - **Effectiveness** - How well users can complete their intended tasks using the system.
    - **Efficiency** - The resources (e.g., time, effort) required to achieve tasks.
    - **Satisfaction** - The user's comfort and positive experience while using the system.

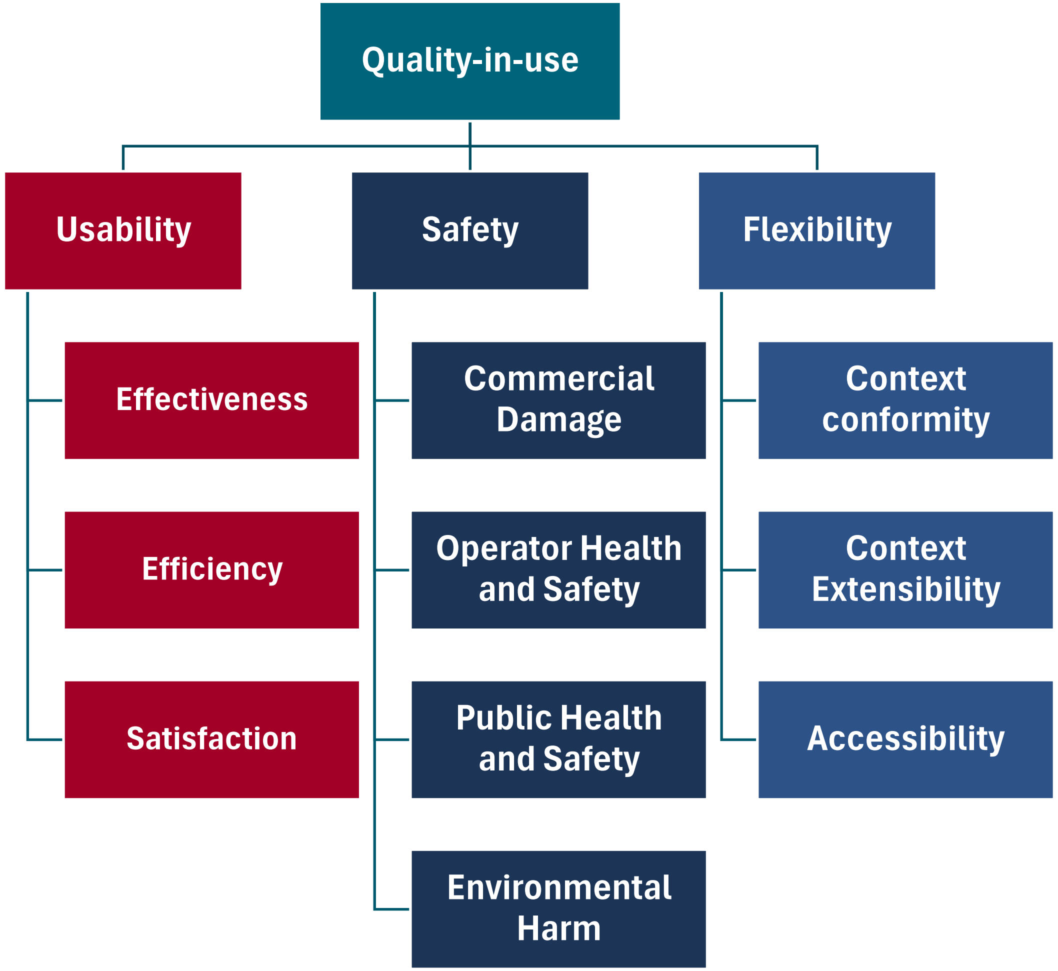
QUALITY-IN-USE: SAFETY

- **Safety** assesses the system’s ability to prevent damage or harm to people, the environment, and commercial interests.
- This characteristics is composed of the following sub-characteristics:
  - **Commercial Damage**: Evaluates how well the system prevents financial losses or harm to the business.
  - **Operator Health and Safety**: How well the system protects users from health risks or safety hazards while operating it.
  - **Public Health and Safety**: Prevents risks or harm to the general public through the system’s usage or operation.
  - **Environmental Harm**: The system should avoid or minimize negative impacts on the environment.

QUALITY-IN-USE: FLEXIBILITY

- **Flexibility** refers to the system's ability to adjust and operate effectively in different contexts or environments.
- This characteristics is composed of the following sub-characteristics:
  - **Context Conformity**: The system's ability to adapt to the specific requirements and constraints of different contexts.
  - **Context Extensibility**: The potential for the system to expand or adjust to new or changing environments without significant modifications.
  - **Accessibility**: Captures how effectively the system can be used by all people, including those with disabilities, across diverse environments.

QUALITY-IN-USE MODEL: OVERVIEW



Quality-in-use
Usability
Safety
Flexibility
Effectiveness
Efficiency
Satisfaction
Commercial Damage
Operator Health and Safety
Public Health and Safety
Environmental Harm
Context conformity
Context Extensibility
Accessibility