**CHAPTER 1**

A **software** is a set of instructions that provide an expected function and behaviour when execute. It is developed, not manufactured in the classical sense, it doesn’t break down: it deteriorates as a result of changes.

The **software crisis** has been between the 1970 and 1980: costs are higher than planned, the delivery dates have a lot of delays. The software has a bad performance and has an impossible maintenance -> this led to a high cost of updates and took unreliable products.

The classification of the **quality factors** of software considers three important aspects of a software product:

1. Its operational features

2. Its capability to support updates

3. Its adaptability to new environments

We have different definitions of **Software Engineering**:

1. SE is the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them.

2. SE is a discipline that integrates methods tools and procedures for the development of Software”

3. SE is the application of scientific principles for the transformation of a problem into a SW solution and its maintenance during all its life

4. SE is an engineering approach covering all aspects of software production

The **management of a software project** is the first level of a software development process, and it covers all the development process.

There are some differences and similarities between Software Engineering and Engineering

Similarities:

1) Activities to be done are not specific of software projects.

2) Many common management techniques.

3) Many similar problems (time, resources , changing specifications).

Differences:

1) The product (software) is not tangible and flexible.

2) The software process is not standard, several alternatives exist.

3) Many times, software projects are unique.

The **Software projects management** includes these activities:

1) Writing proposal

2) Project planning

3) Cost estimation

4) Selection and evaluation of human resources

5) Project control

6) Writing and presenting reports

The **project planning** is divided in:

1. Introduction

2. Project organization

3. Risks analysis

4. Software and Hardware requirements

5. Division of labour

6. Planning

7. Supervision and reporting

**CHAPTER 2**

The **Software process** is a framework for the development of a software, include both the production process and the management process.

The **development process** is the collections of activities towards the development or evolution

of software, it also known as *Lifecycle*. Generic activities that always carried out are:

1. Specification 🡪 Analysis
2. Development 🡪 Design
3. Validation 🡪 Implementation and Testing
4. Evolution 🡪 Maintenance

We have a lot of types of *Lifecycles* (development process):

* Code and fix: the name say all
* Diagram

  Description automatically generatedClassic or Waterfall: all the validation and maintenance is performed on the source code.
* Classic model with prototyping

Diagram

Description automatically generatedThe prototype is the first version of a product in which only some features are integrated or all of them are featured but unfinished, we have 2 types of prototypes: *horizontal*, all views of the system are shown(simulated), and *vertical* which has only some functionality of the system that are fully developed. It helps customers to clearly establish the requirements and developers to improve their products. **PRO**: it reduces the risk of patching on the final product, and it helps both customers and developers to understand the requirement. **CONTRO**: The customer sees a version of the final product; it requires an additional investment. Bad decisions taken during a rapid development of the prototype are usually transferred to the final product.

* Automatic code generation

The goal is to *automatize the software development process*. The basic features are:

Use of formal specification languages

The specification is a prototype of the product

The requirements are discussed by running the specification

The application is derived semi-automatically

**PRO**: it helps reducing human errors, it reduces development costs. **CONTRO**: It is difficult to use formal languages.

Text

Description automatically generated

* **Evolutionary Development**

Adaptable to changing requirements and more elaborated versions are built at each iteration.

* + Incremental model

Sequence of applications of the classical model, each iteration produces a delta of the product. It ends when the final product is delivered.

**PRO**: Useful when not enough human resources for a complete deliverable and each deliverable may be evaluated by the customer: highly interactive.

**CONTRO**: Difficult to know the required increase for each iteration.

* + Spiral model

Use an approach iterative, interactive and evolutive. It introduces risks analysis in the development process.

**PRO:** Each time more complete versions of the productare obtained. Each version is evaluated by the customer ➔Highly interactive.

**CONTRO:** It is difficult to assess risks. Hard to guarantee path towards the final product.

The **methodology** defines an explicit process of software development, this process must be:

* Reproducible
* Defined
* Measurable with respect to performance
* Subject to Optimizations

There is NO UNIVERSAL software methodology, we distinguish:

Structured methodologies

Object oriented methodologies: RUP

The Agile Methodologies **XP**

The main priority is to satisfy the customer with early and continuous releases of usable software, agile processes apply updates for the customer to remain competitive. They release the developed software frequently and with the shortest possible interval of time between releases. Face to face dialogue is the most efficient and effective method to communicate information within a development team, developed software is the first metric of progress. Agile processes promote a bearable development. Funding entities, developers and users that can keep a peaceful ambient.

The continuous attention to technical quality and good design increases agility,

*Simplicity is key*: the best architectures, requirements and designs arise from the organization of the team

At regular intervals, the team reflects about how to be more effective and how to

Table

Description automatically generatedsynchronize and adjust their work.

Table

Description automatically generated

**Development Cycle**: the customer selects the next version to be built, choosing the functional features that he considers more valuable (known as *Stories*) from a set of possible stories, being informed about costs and the required time of their implementation. Coders convert stories into tasks to be done and then convert tasks into a set of test cases to demonstrate that the tasks have been completed. Working with a teammate, the coder runs the test cases and updates the design (evolution) trying to keep it simple.

**Object oriented methodologies: RUP (Rational Unified Process)**

Uses **UML** as modelling language, has these features:

* *Use cases driven process*: from specification to maintenance.
* *Iterative and incremental process*: iterations depending on the importance of use cases and the study of risks.
* *Architecture centred process*: reusable and serving as a guide towards the solution

It’s an **iterative** and **incremental** methodology, activities are performed in a mini fall with a limited scope.

From a **dynamic view** (time-oriented organization) we can divided the software in: cycles, *Phases*, Iterations and Milestones.

The phases are:

*Inception*: the scope and goals of the project are defined; the functionality and capabilities of the product are defined.

*Elaboration*: The problem domain and the desired functionality are studied in depth. The basic architecture is defined, the project plan is defined according to the available resources.

*Construction*: On each iteration analysis, design and implementation tasks are performed; the architecture is refined, and an important part of the work is dedicated to coding and testing. The system and its use are documented. This phase provides a built product and a documentation

*Transition*: The product is delivered to the user for its use. Marketing, packaging, installation, configuration, training, support and maintenance; guides are completed and refined.

From a **static view** we have Workflows:

Table, timeline

Description automatically generated

Timeline

Description automatically generated

**CHAPTER 3**

When systems grow in size it is required an organization in terms of subsystems so that they are manageable, different strategies to manage complexity, usually related with design at different levels of abstraction, have been used.

Structured methods

The **modular architecture** (and the *modular diagram)*is based on the notion of *module*, a system is partitioned in modules that invoque or provide service to other modules, possibly with data passing in both directions.

A *module* is a part of a program that implements part of the functionality, a module may be decomposed in terms of other modules of a lower level, we divide the *architecture* in 2 parts:

Preliminary design: Structuring the system in terms of modules, building structure diagram. Module = black box

Detailed design: Description of process that are implemented by modules.

Module = white box.

The **objected oriented architectures** are made by *Classes* as decomposition units (structure + behavior in a module). The structures of classes are propagated to the code and new classes are incorporated when lowering the abstraction level. Approaches based on modules and objects are low level ones, they do not divide the application in terms of functional blocks, but they are mere groupings of code.

The **software architecture** has to do with the design and implementation of high-level structures. It is the outcome after assembling a number of different architectural elements in order to adequately satisfy both functional and non-functional requirements such as trustability, scalability, portability and availability. In the *description phase* of the Software Architecture the system must be organized in terms of **subsystems**. Many times, the architecture is based on other similar previously developed systems by means of architectonic patterns.

Some interesting patterns in information systems are:

* Distributed Systems: a software system in which information processing is distributed among different computing nodes.
* Personal system: non distributed systems that are designed to be run in a personal computer or workstation.
* Embedded system: information systems (hardware + software), usually real-time ones integrated in a more general engineering system that perform functions of control, processing and/or monitoring.

**Distributed systems**

*Multi-processing architectures*: The system consists of multiple processes that may or may not be run in different processors.

*Client/Server architectures*: The system is seen as a set of services that are provided to client applications by server applications. Client and server applications are handled separately. The server (S) is a service provider, The client (C) is a consumer of services. C and S interact by means of a message passing mechanism: service request and answer.

*Distributed objects architectures*: The system is seen as a set of interacting objects whose location is not relevant. There is no distinction between a provider of a service and a consumer

**Multi layered architecture**

A **layered system** is a sorted set of subsystems each one defined in terms of the ones located below them and providing the implementation base of the systems above. The objects in each layer may be independent although there use to be some dependencies between objects of different layers. There is a *relationship client/server* between the lower layers (providing services) and the upper layers (using those services).

Layered architectures may be open or closed depending on the dependencies between layers.

open: a layer may use characteristics of any layer.

closed: a layer may only use characteristics of its adjacent lower layer.

It is recommended to use closed architectures, because there are fewer dependencies between layers and because it is easier to apply changes because the interface of a layer only affects to its immediate upper layer.

The **2 layers architecture: thin layers** is useful for legacy systems in which the separation between processes and data management is not feasible and for data intensive applications with little processing. Layer refers to a logical segmentation of the solution whereas tier refers to a physical segmentation or location. (Business Logic + Data and Presentation).

Let’s see some variations:Diagram

Description automatically generatedA picture containing timeline

Description automatically generated

**3 layers architecture** is characterized by:

Presentation layer: presentation of computation results to the user and user input detection.

Business logic layer: provide the functionality of the application.

Data layer: provide persistence to data by means of databases or files.

This architecture has many advantages: first of all, isolate business logic in a separate component, the Distribution of layers are settled in different machines or processes. It’s possible parallel development and assigning resources to each layer. Has a good *software reuse*.