Software Engineering for HPC - Notes

260236

March 2024

Preface

Every theory section in these notes has been taken from two sources:

- None

About:

GitHub repository

Contents

1 Introduction

1.1 The importance of software engineering

Software engineering is so important because it is everywhere. Our society is now totally dependent on software-intensive systems. Think about it. The society could not function without software, for example:

- Transportation systems;
- Energy systems;
- Manufacturing systems.

For these reasons, **software failures cannot be tolerated**.

In the following list, we can see some famous software issues:

- **911 Outage on April 2014**. On 10th April 2014, Washington State had no 911 service for six hours. A software issue causes this event. The software dispatching the calls had a counter used to assign a unique identifier to each call. The counter went over the threshold defined by developers. All calls from that moment on were rejected.

  More info is here.

- **Ariane 5, 1996**. On 4th June 1996, forty seconds after take off, Ariane 5 broke up and exploded. The total cost for developing the launcher has been 8000 million dollars. The launcher contained a cluster of satellites for 500 million dollars. Again, the explosion was caused by software failure.

  More info is here: accident tech report and video.

1.2 Software engineering: definition

There are some fields of computer science dealing with software systems:

- Large and complex;
- Built by teams;
- It exists in many versions;
- Last many years;
- Undergo changes.

In each field, a software engineer needs to have some skills. In contrast to a *programmer* that has the following abilities:

- They develop a complete program;
- They work on known specifications;
- They work individually.

A **software engineer** has the following **skills**:

- **Identifies** *requirements* and develops *specifications*;
- **Designs** a component to be combined with other components, developed, maintained, and used by others; component can become part of several systems;
- **Works in a team**.

We can <u>summarize</u> the skills of a software engineer as follows:

- Technical
- Project management
- Cognitive
- Enterprise organization
- Interaction with different cultures
- Domain knowledge

The **main goal** of a software engineer is to **develop software products**. Not only is the product significant, but the **process is also fundamental**. The quality of the process affects the quality of the product.

1.3 The software product and the process

The product developed by a software engineer differs from traditional product types. It isn't easy to describe and evaluate because it is intangible. Some **aspects affecting the product quality**:

- **Development technology**;
- **Process quality**;
- **People quality**;
- **Cost, time and schedule**.

1.3.1 ISO/IEC 25010

An ISO (International Organization for Standardization) called **ISO/IEC 25010** comprises the **nine quality characteristics**:

SOFTWARE PRODUCT QUALITY								
FUNCTIONAL SUITABILITY	PERFORMANCE EFFICIENCY	COMPATIBILITY	INTERACTION CAPABILITY	RELIABILITY	SECURITY	MAINTAINABILITY	FLEXIBILITY	SAFETY
FUNCTIONAL COMPLETENESS FUNCTIONAL CORRECTNESS FUNCTIONAL APPROPRIATENESS iso25000.com	TIME BEHAVIOUR RESOURCE UTILIZATION CAPACITY	CO-EXISTENCE INTEROPERABILITY	APPROPRIATENESS RECOGNIZABILITY LEARNABILITY OPERABILITY USER ERROR PROTECTION USER ENGAGEMENT INCLUSIVITY USER ASSISTANCE SELF-DESCRIPTIVENESS	FAULTLESSNESS AVAILABILITY FAULT TOLERANCE RECOVERABILITY	CONFIDENTIALITY INTEGRITY NON-REPUDIATION ACCOUNTABILITY AUTHENTICITY RESISTANCE	MODULARITY REUSABILITY ANALYSABILITY MODIFIABILITY TESTABILITY	ADAPTABILITY SCALABILITY INSTALLABILITY REPLACEABILITY	OPERATIONAL CONSTRAINT RISK IDENTIFICATION FAIL SAFE HAZARD WARNING SAFE INTEGRATION

Figure 1: ISO/IEC 25010

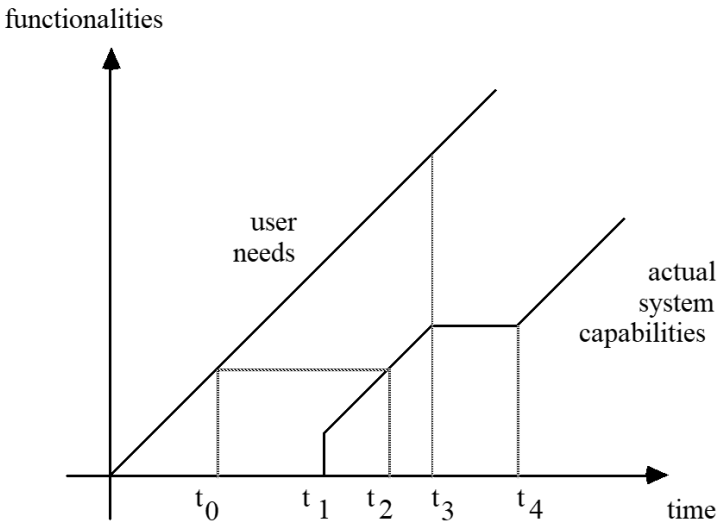
1.3.2 Productivity

A process quality to consider is **productivity** (the **process of producing a product**). The definition can be: "ability to produce a good amount of product". To **measure it**, we can use **delivered items by unit of effort**, where:

- **Delivered items**: lines of code (and variations) function points;
- **Unit of effort**: person month (note: persons and months cannot be interchanged).

1.3.3 Timeliness

Another process quality to consider is timeliness. The definition is: "**the ability to respond to change requests in a timely fashion**".



As you can see by the graph, the "*user needs*" is a linear function (and sometimes can be exponential!). A software engineer should be able to respond to the client's requests as soon as possible. As the graph shows, a request made on time t_0 is completed on time t_2; but another request can be made at that time, and so on. The actual system capabilities can't grow up always because sometimes there are "brainstorming times" to increase product quality (ISO/IEC 25010).

1.4 Software Lifecycle

Initially, no reference model is inside a software lifecycle: code and fix (or refactoring). However, a traditional waterfall model is chosen to react to the many problems that a software engineer faces.

1.4.1 Waterfall model

The **waterfall model** is a **breakdown of development activities into linear sequential phases**, meaning they are passed down onto each other, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. [1] Its organization is the following:

- High phases:
  - **Feasibility Study**: this is a **cost-benefit analysis**.

    The ***main goal*** is determining whether the project should be started (e.g. buy or make), possible alternatives, and needed resources.

    The ***outcome*** is a **feasibility study document**. This paper provides:

    * A preliminary problem description;
    * Some scenarios describing possible solutions;
    * Costs and schedules for the different alternatives.
  - **Requirements Analysis and Specification**: this is an **analysis of the domain in which the application takes place**.

    The ***main goal*** is to **identify requirements** and **derive specifications for the software**. Note these specifics require a (continuous) interaction with the user and an understanding of the properties of the domain.

    The ***outcome*** is a particular document called **Requirements Analysis and Specification Document** (RASD).
  - **Design**: this is the **definition of the software architecture**. There, the definition of components (modules) and the relations/interactions among these components.

    The ***main goal*** is to **support the concurrent development of separate responsibilities**.

    The ***outcome*** is a summary of this info in a **design document**.
- Low phases:
  - **Coding and Unit Test**: each module is **implemented using the chosen programming language**. Furthermore, each module is **tested in isolation** by the module developer. Also, the programs should include their documentation.
  - **Integration and System Test**: the modules are **integrated into (sub)systems**. The integrated (sub)systems are **tested**. Follows an incremental implementation scheme. A complete system test is needed to verify the overall properties. Note that sometimes we have alpha test and beta test.

- **Deployment**: is the process used to conceive, specify, design, program, document, test, and bug fix to create and maintain applications, frameworks, or other software components.
- **Maintenance**: the maintenance is divided into **two types**:
  * **Corrective** deals with the **repair of faults or defects found**.
  * **Evolution** is also divided into **three types**:
    · ***Adaptive*** maintenance: consists of **adapting software to changes in the environment** (the hardware or the operating system, business rules, government policies).
    · ***Perfective*** maintenance: mainly deals with accommodating **new or changed user requirements**.
    · ***Preventive*** maintenance: concerns activities aimed at **increasing the system's maintainability**.

⚠ Problems derived from correction and evolution

Note: the **distinction between correction and evolution can be unclear** because specifications often must be completed and clarified. This causes problems because specs are usually part of a developer and customer contract.

- **Early frozen specs** can be problematic because they are more likely to be wrong.
- Another problem is **software evolution** because **it is never anticipated or planned**. Since the software is easy to change, often, under emergency, changes are applied directly to code, and consequently, the state of project documents is inconsistent.

✔ Solutions - Best practices

Some good engineering practices exist to solve the evolution problem: **first, modify the design, then change implementation and apply changes consistently in all documents**. Also, the **software must be designed to accommodate changes cost-effectively**. This is one of the *main goals* of software engineering.

✔ Flexible processes

We can make the **waterfall model more flexible**. In this case, the **main goal is to adapt to changes (especially in requirements and specs)**. The idea is that the stages are **not necessarily sequential**, and **processes** become **iterative and incremental**.

2 HPC Software, Relevant Qualities and Systems Engineering Methods

2.1 High Performance Computing Software

There's no single definition of HPC, but it can be explained in a number of ways:

Definition 1

The practice of aggregating computing power in a way that delivers much high performance than one could get out of a typical desktop computer or workstation to solve large problems in science, engineering, or business.

Thousands of processors working in parallel to analyze billions of pieces of data in real time, performing calculations thousands of times faster than a normal computer.

The use of parallel processing for running advanced, large-scale application programs efficiently, reliably and very quickly on supercomputer systems.

The platform technology concerned with programming for performance, where performance takes the broad meaning of:

- Speed (reducing time to solution);
- Energy efficiency (doing more with less power);
- Upscaling (handling larger problems such as simulating a wing aand then a full plane, or a cell and then an organ);
- High throughput (the ability to handle large volumes of data in near real-time, as required in the financial services industry, telecoms or satellite imagery).

As **Parallel and Distributed Computing (PDC)** exist, it is necessary to explain the difference. The main characteristics of PDC are:

- **Concurrency**: it is a property of software. **A piece of software is also concurrent if it can have more than one active execution context**.
- **Parallelism**: it is a property of software. **The execution of different tasks/pieces of software at the same time**.
- **Distribution**. **The execution of different tasks/pieces of software on physically distinct computing nodes connected through a network, lack of a global clock**.

PDCs are **multi-core machines**, whereas **HPCs** are **quantum computers**. However, **both share parallel machines**, **HPC clusters** and **cloud infrastructures**.

There are **two categories** of HPC software:

- **Compute-intensive applications**. These are **complex calculations that require a large number of computing resources**. They also often require parallel computing.
- **Data-intensive applications**. They **focus on processing, storing and retrieving large amounts of data**. Typically built as distributed systems to ensure reliability and scalability.

2.2 Relevant Qualities

For the two categories explained, there are some important characteristics:

- For **Compute-intensive applications**:
  - **Correctness**: the **software is correct if it satisfies the specifications**, but be careful! Sometimes, modelling reality into a model (using the specifications) isn't the bigger problem. Instead, it is difficult or impossible to show actual correctness concerning reality. For **example**, imagine you are building a simulator of a planet lander before you have ever visited it.

    How can you fix this issue? We can **check that the software output fulfils the important desired properties** and **identify and apply a measure of accuracy**.
  - **Performance**: it is the **efficient use of resources**. Again, be careful! Is it a good idea? Is performance improvement always a good idea? Because it is **not necessarily if**:
    * It makes software **more difficult to read** and **maintain**
    * It **reduces the portability** of software
  - **Portability**
  - **Maintainability**:
- For **Data-intensive applications**:
  - **Reliability**: can be mathematically defined as **probability of absence of failures for a certain period**. The typical expectations are:
    * The application **performs the expected function**
    * It can **tolerate mistakes by users**
    * It **prevents unauthorized access** and **abuse**
  - **Scalability**:
  - **Maintainability**:

In the software, there can be some errors, but a software engineer should be able to recognize the type of failure, faults or defects:

- A **defect** is an **imperfection or deficiency in a work product** where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.
- A **defect encountered during software execution** is a **fault** (a fault is a subtype of defect).
- A **system failure** can be:
  - Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits.
  - An event in which a system or system component does not perform a required function within specified limits.

References

[1] F. Bomarius, M. Oivo, P. Jaring, and P. Abrahamsson. *Product-Focused Software Process Improvement: 10th International Conference, PROFES 2009, Oulu, Finland, June 15-17, 2009, Proceedings*. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2009.

Index