

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SOFTWARE ENGINEERING – LECTURE 01

INTRODUCTION TO SOFTWARE ENGINEERING

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>

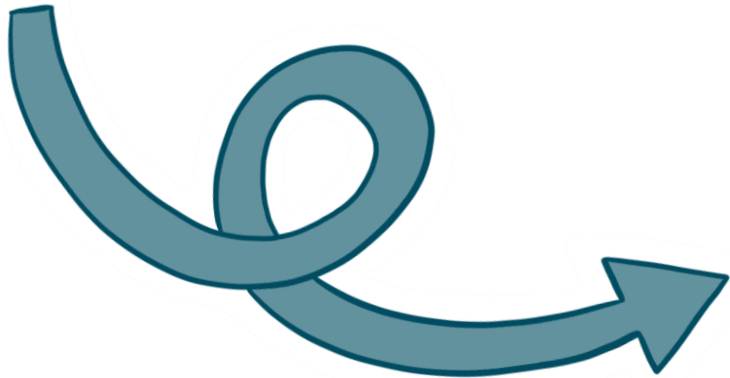
BEFORE WE START, DID YOU KNOW?



- The Computer Science degree program website is **constantly updated** with news, events, guides and relevant information
- Our Computer Science degree program activated a dedicated **Telegram channel** to notify students of relevant news and events in real-time
- We are also on the main **social media** platforms, where we post some interesting content

BEFORE WE START, DID YOU KNOW...

- You'll find all the **relevant links** and **social media handles** at taplink.cc/informatica_unina



WHY ENGINEERING?



- 1-3 artisans, 2-5 days, € 5k
- An unstructured approach is feasible

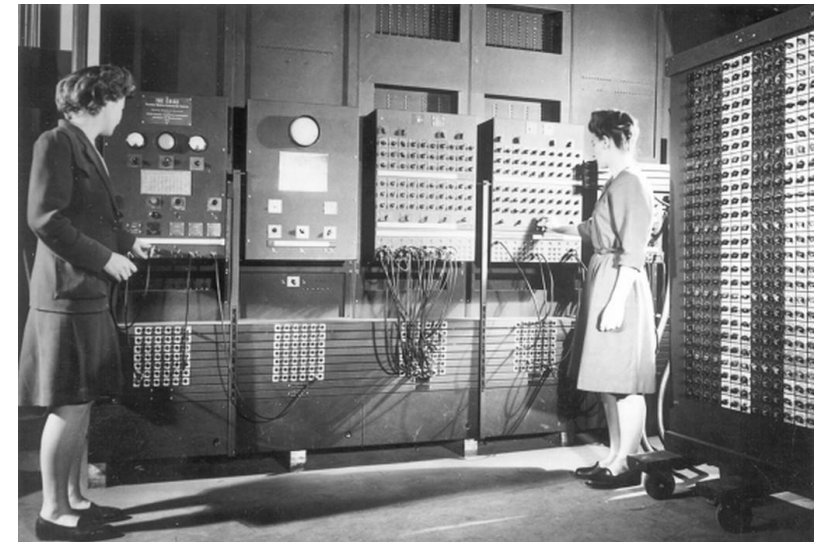


- 1000 people, 330 companies
- A systematic, structured approach is needed to deal with this level of complexity

SOFTWARE ENGINEERING: A BRIEF HISTORY

ENIAC (Electronic Numerical Integrator and Computer) was introduced in 1946

- The first programmable computers were astonishing machines
- The machine was way more spectacular (and expensive) than some sheets of coding
- Programming was generally regarded as **less important** than hardware engineering
- In the first years, programs were mostly developed **individually**



[Jean Bartik \(left\) and Frances Spence operating the ENIAC](#)

NAÏVE APPROACH TO SOFTWARE DEVELOPMENT

Somewhat similar to the approach you have been exposed to so far



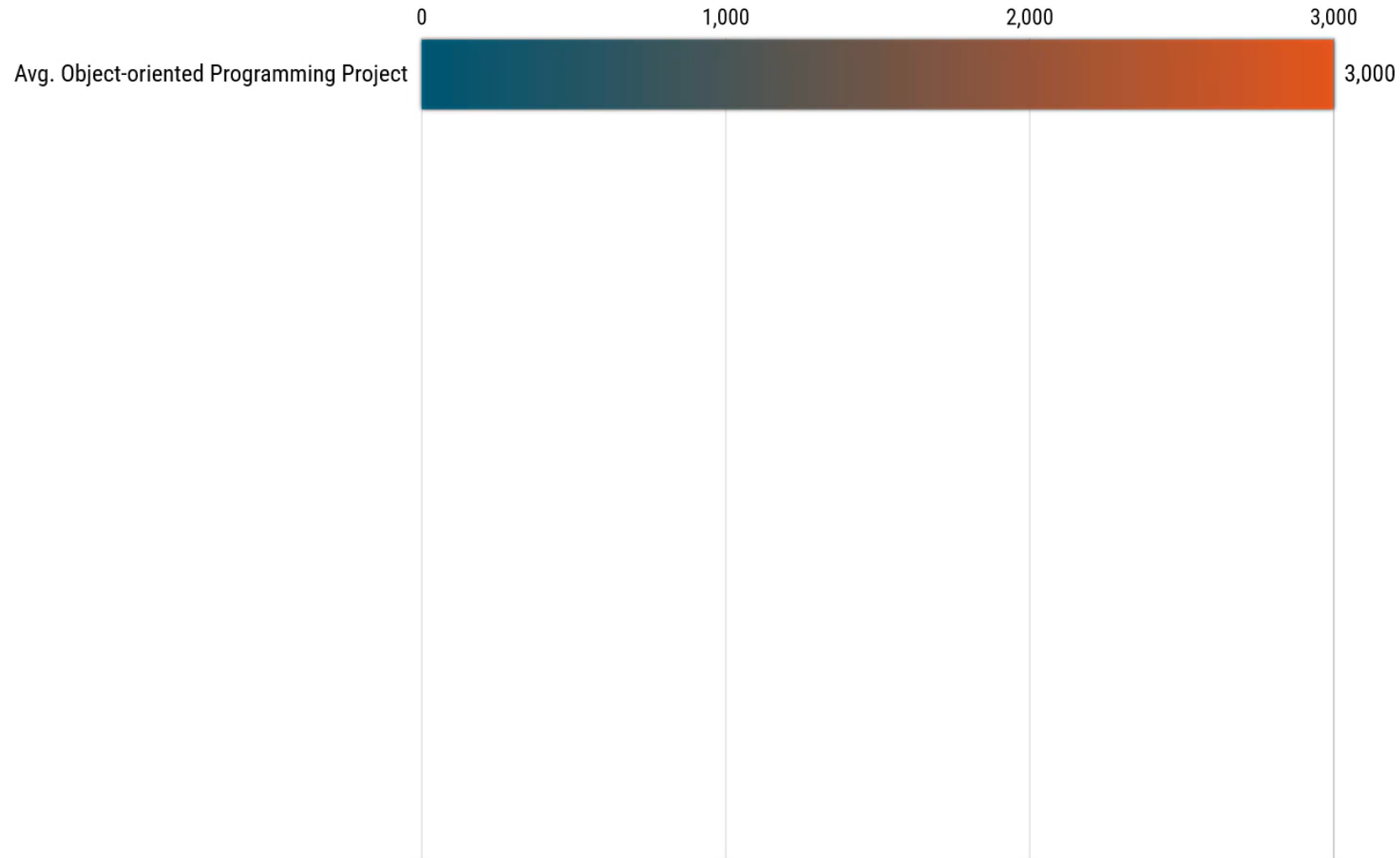
Some challenges quickly arise as the software becomes more complex:

- Where do requirements come from? Are they correct? Complete?
- How do we organize the program?
- How to split tasks and collaborate with other programmers?
- Are the requirements actually satisfied? Is the program correct?
- How to cope with changes in requirements?

SIZE DOES MATTER

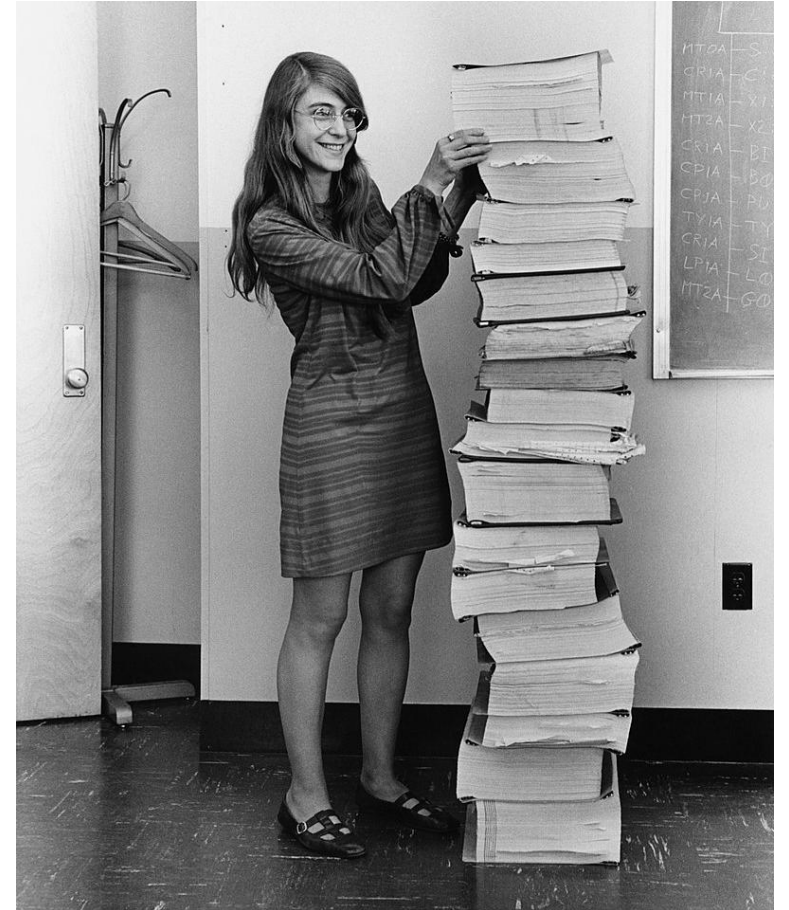
- What's the **biggest** program you ever developed?
- How do we even measure the «size» of software?
- Average Object-Oriented Programming Project in the past year:
 - ~ **3000** Non-commenting lines of code (including automatically-generated ones)
 - ~ **35** Classes
- That was quite a lot of work, right?

SOFTWARE SIZE COMPARISON: LINES OF CODE



SOFTWARE ENGINEERING: A BRIEF HISTORY

- In the 1960s, systems started to become more complex (e.g.: @NASA)
- Individual approaches did not scale up to large and complex software systems
- The **Software Crisis** started
 - Software systems were **unreliable**, **overrun costs** and **release schedules**
- The need for a new discipline studying methodologies for Software Production became evident



Margaret Hamilton, with the code she and her team wrote for the Apollo Guidance Computer, 1969

PROGRAMS vs SOFTWARE PRODUCTS

- **Program:** generally developed by a single person, which is also the main user
 - No need for a formal software process or documentation
 - Generally not marketable
- **Software Product:** developed by teams, used by other people
 - Development **costs** significantly higher, structured software process required
 - Includes way more than just source code or an executable file
 - Documentation, User manuals, Install and configuration manuals
 - **Automated Test Suites**
 - It's an industrial product, developed and tested according to **standards**
 - Not a one-shot affair, it needs to **evolve** over time

KINDS OF SOFTWARE PRODUCTS

- **Off-the-shelf software:** general-purpose software that can be purchased (if commercial) or acquired by anyone interested
- **Customly-developed software:** software developed specifically for the needs of a customer
- The key difference between is **who** defines the requirements:
 - Off-the-shelf: requirements are defined by the marketing team
 - Customly-developed software: requirements are defined by the customer

PROFESSIONAL SOFTWARE DEVELOPMENT

- Professional development is neither (only) a **craft** nor (only) a **science**
- It's an **industrial process**
 - Software is developed in companies, organized in teams with many members
 - Under time and cost constraints, with strict quality requirements
- As with any industrial process, **methodologies** have been developed to guide the design, development and verification of software systems
 - These methodologies formalize and organize knowledge coming from prior experience
- As Computer Science professionals, you **need** to know them
 - Being the best programmer won't cut it!

SOFTWARE ENGINEERING

Software Engineering is the **engineering discipline** concerned with **all aspects of professional software production**

- **Engineering discipline:** Engineers make things work. They apply theories, methods, and tools where these are appropriate, to solve problems within the given **organizational** and **financial constraints**.
- **All aspects of software production:** not limited to technical programming aspects! It also includes aspects related to project management and the development of tools, methods, and theories to support software development.

ENGINEERING SOFTWARE IS NOT EASY

Producing high quality software within budget and organizational constraints is not easy!

- Software is **abstract** and **intangible**
 - Not constrained by properties of materials or laws of physics
 - Can quickly become **extremely complex** and difficult to manage
- Software is **everywhere** and each software is **diverse**
 - National infrastructures and utilities are controlled by computer-based systems
 - Most electrical products include a computer and controlling software
 - Industrial manufacturing and the financial system are largely computerized
 - Entertainment (movies, tv-shows, video games, ...) is software-intensive

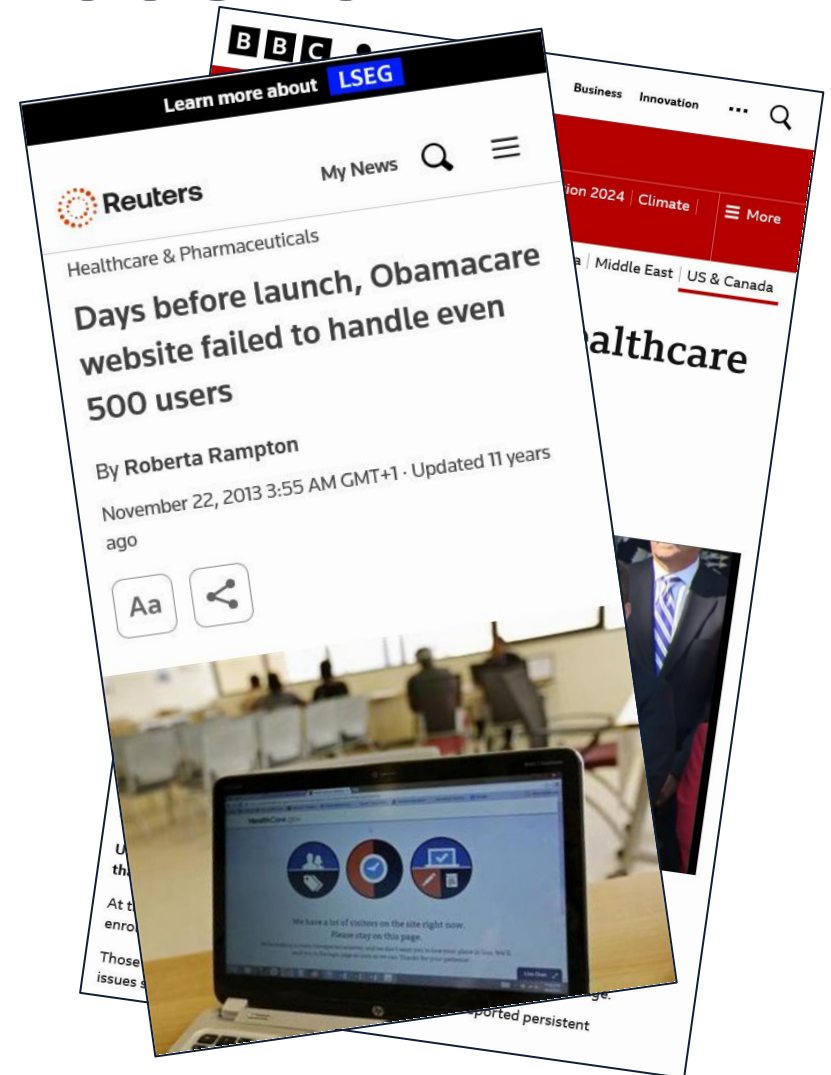
ENGINEERING SOFTWARE IS NOT EASY

- There exist no universal methodologies suitable for all kinds of software, all kinds of companies and all kinds of situations
- No **silver bullet** technology can magically fix all issues!
- Software projects still fail (or face critical issues)!
 - They go **overbudget**
 - They **overrun** the **planned release** schedule
 - The delivered product does not meet the expected **quality** requirements..
 - .. or it does not work at all, delivering little to no added value to the customer!

SOFTWARE PROJECTS WITH ISSUES

Healthcare Exchange website
(<https://www.healthcare.gov/>)

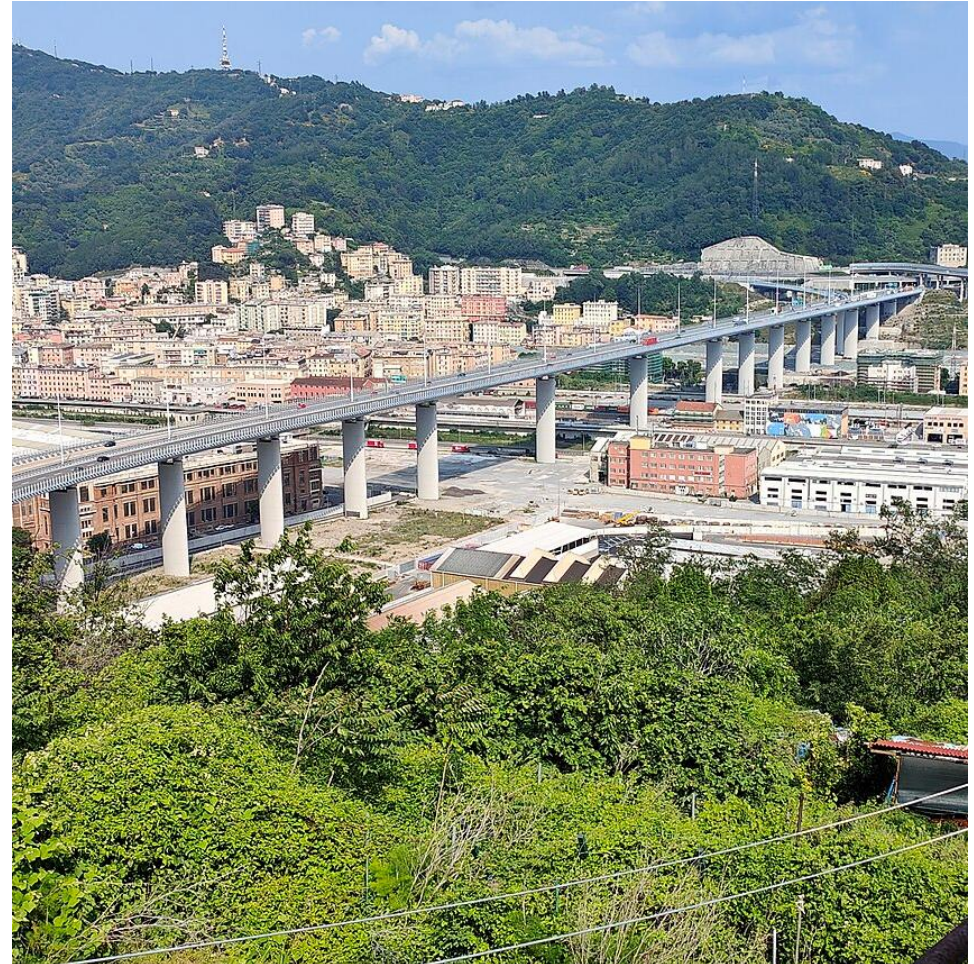
- Commissioned by the US Federal Government
- Only 1% of users managed the enroll with the site during the first week of operation
- Costs: **\$ 1,5 B** (15 times more than the expected cost **\$ 93,7 M**)



JUST FOR REFERENCE: BUILDING AN OVERPASS

The San Giorgio overpass in Genova, Italy.

- Replaces the sadly famous «Morandi» overpass, partially collapsed in 2018
- 1.1 km long, 31 m wide, 45 m high
- 4 lanes + 2 emergency lanes
- Costs: **€ 202 M.** Built in **~1 year.**



SOFTWARE PROJECTS WITH ISSUES

Queensland Health new payroll system

- Managed the payroll of 80.000 workers employed in 16 departments.
- Project is contracted with IBM Australia and starts in 2007
- Planned release in 2008
- Actual release in 2010. System barely worked. To fix it, it needed 'til 2018
- Costs: ~ **\$ 850 M** (200 times more than the expected costs of **\$ 4,2 M**)



SOFTWARE PROJECTS WITH ISSUES

Cyberpunk 2077 was one of the most hyped games ever

- Open world RPG by CD Projekt Red
- Announced in May 2012
- Initial release planned for april 2020.
- Delayed to september, then november, then december 2020.
- On release, game was riddled with bugs, unplayable on some consoles
- Game costed **\$ 400 M**



SOFTWARE PROJECTS WITH ISSUES



https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects

SOFTWARE PROJECTS WITH ISSUES

yclopedia


You can help by expanding it.

https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects



SOFTWARE PROJECTS WITH MORE SERIOUS ISSUES


- Not only budget and release schedule overruns

 [SUBSCRIBE](#)

TONY LONG OCT 26, 2009 12:00 AM

Oct. 26, 1992: Software Glitch Cripples Ambulance Service

Computerized systems can help, or hurt. In the case of the London Ambulance Service, it definitely hurt.

 **SAVE**

1992: A software error causes London's brand new computer-aided ambulance-dispatch system to fail. The ensuing snafu is blamed for anywhere between 30 and 45 deaths.

 CATEGORIES TV RADIO COMMUNICATE WHERE I LIVE INDEX



You are in: Science/Nature
Thursday, 12 December, 2002, 01:54 GMT

Europe's super rocket explodes



The flight lasted barely a few minutes

By Dr David Whitehouse
BBC News Online science editor

Europe's new heavy-lift rocket has failed on its maiden flight.

The Ariane 5-ECA blasted off from the Kourou spaceport in French Guiana at 1921 local time (2221 GMT) and exploded over the Atlantic three minutes later.

Sources close to the launch site have told BBC News Online that first indications suggest a failure of the rocket's main Vulcain-2 engine, as its performance was seen to reduce dramatically before the vehicle veered off course.

The rocket's payload of two satellites, valued at more than 600m euros, was also destroyed.


WATCH/LISTEN ON THIS STORY
The BBC's Owen Franks
"Everything seemed to be working normally"
Jean-Yves Le Gall, Arianespace Chief Executive
"It is too early to give a clear explanation"
Dr Chris Welsh, Lecturer in Space Technology
"It's going to be another bad hit for the space insurance industry"

The Business of Space
Challenging times
Nasa's dash for cash
Communication blues
Is bigger better?
Sea launch success
Rocket surplus

BBC SPORT
BBC WEATHER
BBC NEWS
SERVICES
Daily E-mail
News Ticker
Mobile/PDAS
Text Only
Feedback
Help
EDITIONS
Change to World




“ We have already known failures, we will know more ”
Jean-Yves Le Gall, Arianespace


BBC SCIENCE
See a space launch
See also:
12 Dec 02 | Science/Nature
Flagship space mission in doubt
12 Dec 02 | Science/Nature
Blow to launcher market
12 Dec 02 | Science/Nature
Ariane 5: A short history
21 Nov 02 | Science/Nature
Blast-off for Boeing hopes
21 Aug 02 | Science/Nature
New US rocket blasts off



FRONTLINE **MENU**

As Boeing Agrees to Plead Guilty to Fraud, a Look Back at What Led Up to the 737 Max Crashes That Killed 346 People

Share:   



A screengrab from the FRONTLINE/New York Times documentary 'Boeing's Fatal Flaw.'

JULY 8, 2024

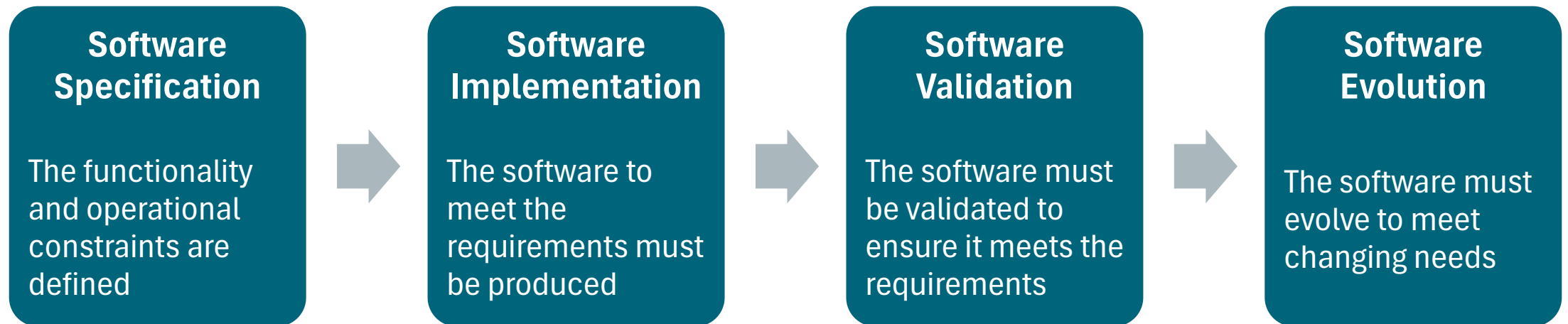
by [Patrice Taddonio](#)

Boeing has agreed to plead guilty to a

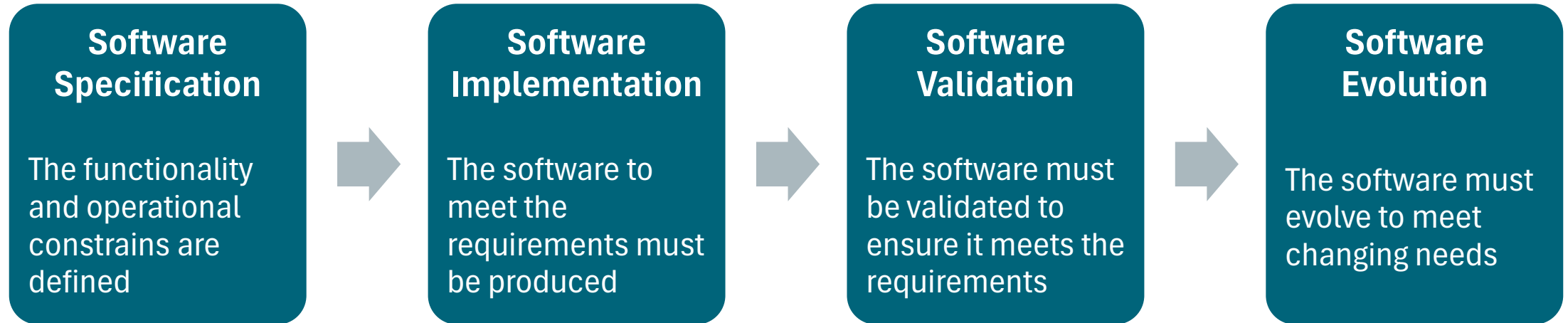
SOFTWARE PROCESSES OVERVIEW

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**

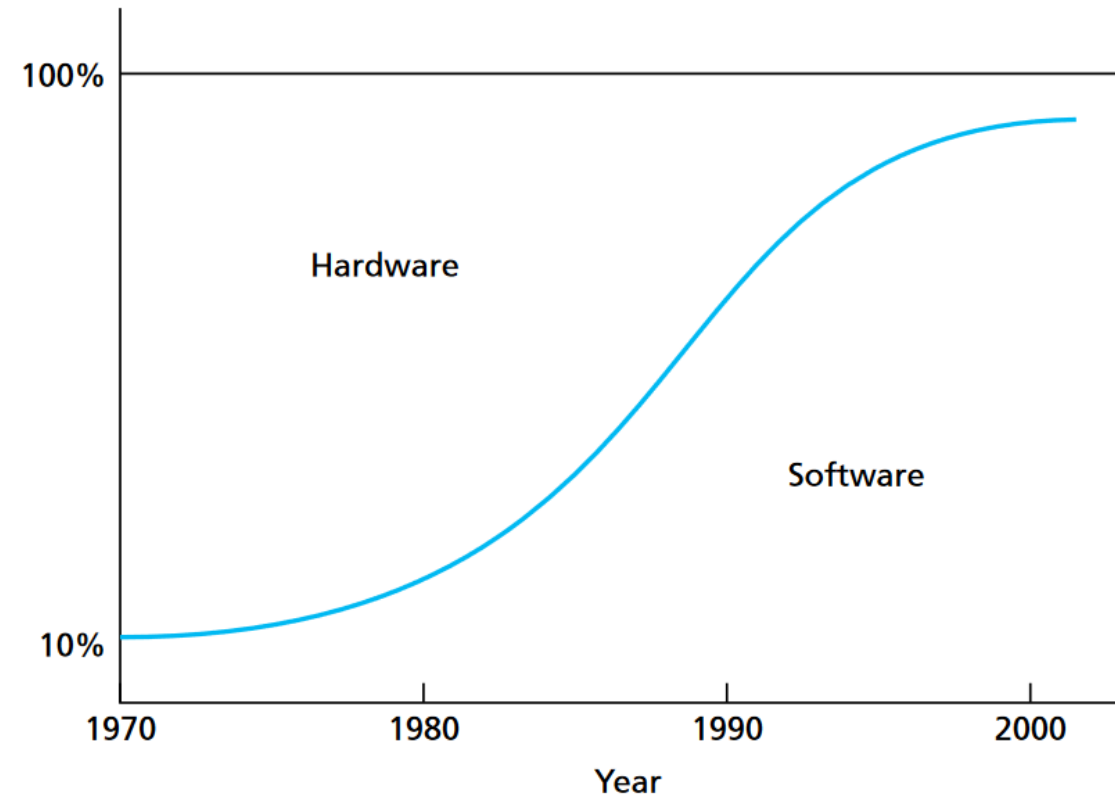


SOFTWARE PROCESSES



- These activities are complex, and possibly include many sub-activities
- Like any other creative process, **software processes** rely on **people** making **decisions** and **judgements**
- Software processes also typically include additional activities, such as **project planning** and **monitoring**

SOFTWARE DEVELOPMENT COSTS



Changes in the relative costs of hardware and software
Software Engineering for Students: a Programming Approach, Douglas Bell

SOFTWARE DEVELOPMENT COSTS



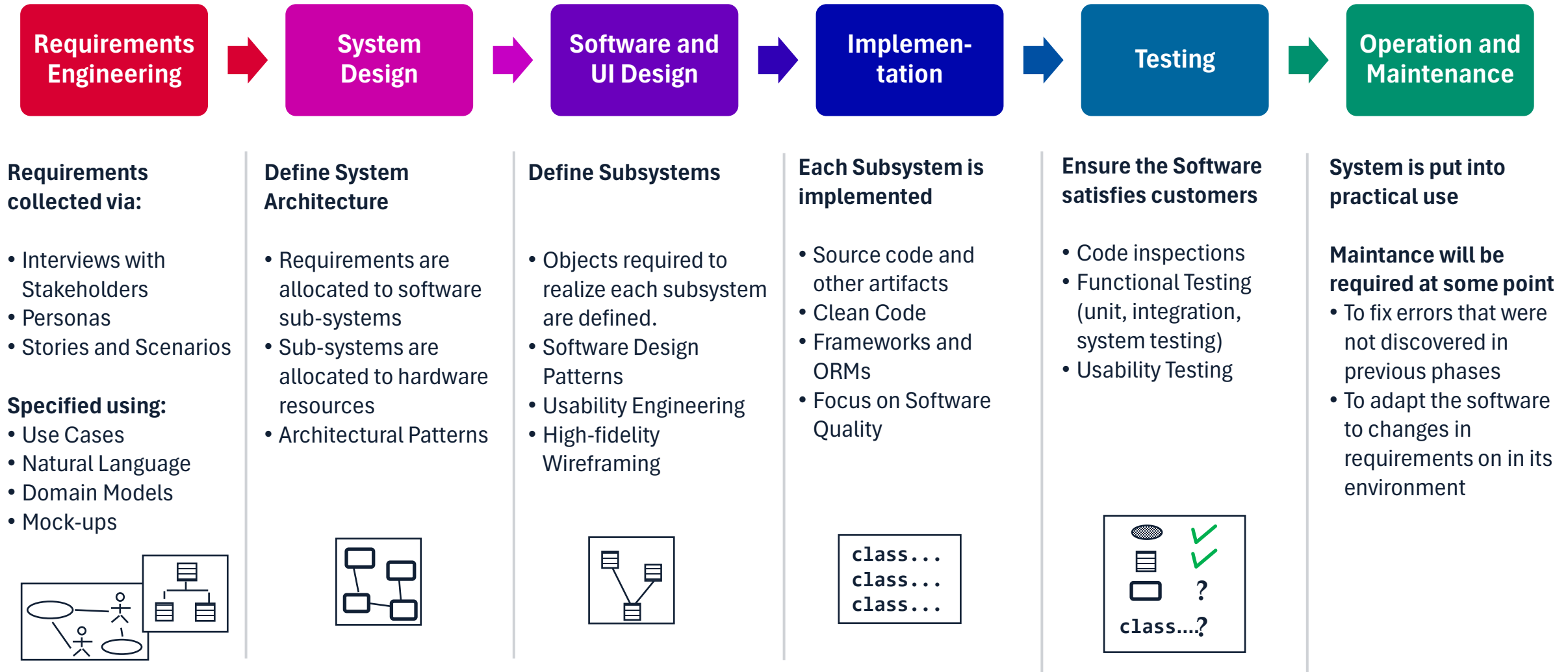
Relative costs of the stages of a software process

Software Engineering for Students: a Programming Approach, Douglas Bell

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



A SOFTWARE ENGINEER'S JOB

- To sum it all up, Software Engineers must
 - Adopt a **systematic** and **organized** approach to software production
 - Use appropriate **tools**, **techniques**, and **methodologies**
- Depending on:
 - The problem to be solved
 - Development and organizational constraints
 - Available resources
- To produce **high quality** software
 - ... within a given **budget**
 - ... before a given **deadline**
 - ... while **changes** occur!

COURSE DETAILS



COURSE ORGANIZATION

- Software Engineering is a **10 credits** course.
- Two channels: **Canale 1 (us!)** and **Canale 2 (Tramontana/Breve)**
- **Split based on your last name**
 - If your last name starts with **A-G**, you're in the right place!
 - If your last name starts with **H-Z**, you should attend lectures in Canale 2

COURSE ORGANIZATION

- **Two modules: Module A** and **Module B**
- Each module is worth **5 credits**
 - **20 lectures** per module
 - 1-2 industry seminars
 - Classroom exercises
- Full course description is available (in english and in italian) on:
 - **Mod. A:** <https://www.docenti.unina.it/sergio.dimartino/2025/N86/U2358>
 - **Mod. B:** <https://www.docenti.unina.it/luigiliberolucio.starace/2025/N86/U2359>

TEACHERS



Sergio Di Martino (Module A)



sergio.dimartino@unina.it



<https://www.docenti.unina.it/sergio.dimartino>



Luigi Libero Lucio Starace (Module B)



luigiliberolucio.starace@unina.it



<https://www.docenti.unina.it/luigiliberolucio.starace>

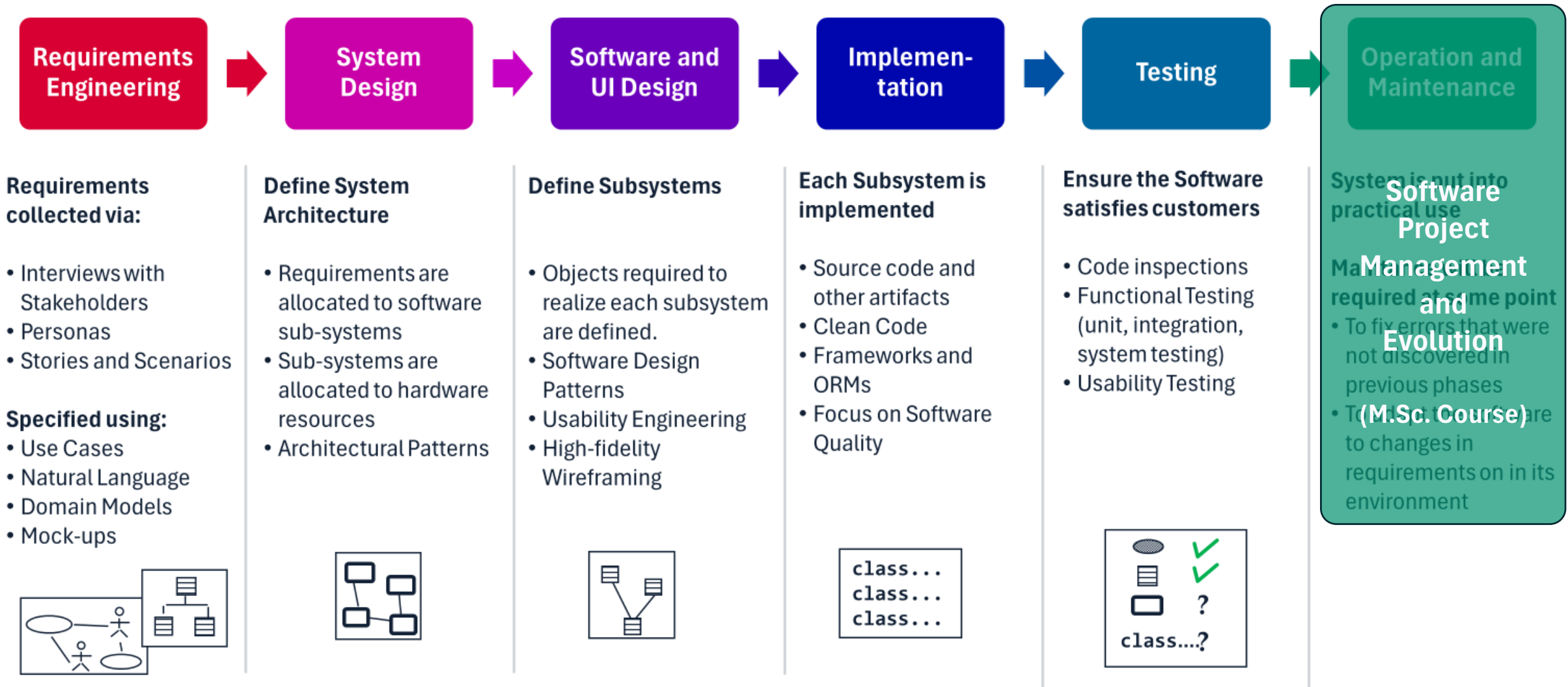


<https://luistar.github.io>

LEARNING OBJECTIVES

- Understanding Software Engineering and its processes, phases, and deliverables (programming in the large)
- Understanding modelling techniques used in software processes and their importance in software design, development, and maintenance
- Acquiring basic techniques and methodologies for designing and implementing readable and maintainable software
- Understanding the fundamental concepts of Human-Computer Interaction and being able to apply them in User Interface design
- Understanding of the challenges of collaborative Software Engineering (teamwork) and the organization and management of software projects

TOPICS AND CONTENT SPLIT



PREREQUISITES

- Prerequisites for understanding course concepts:
 - Good programming knowledge
 - Understanding of object-oriented design and programming

COMMUNICATION PROCESS

Teachers → Students

- Institutional websites:
 - <https://www.docenti.unina.it/sergio.dimartino>
 - <https://www.docenti.unina.it/luigiliberolucio.starace>
- Subscribe to the course (both for Module A and Module B), and **make sure to activate the mailing list as well**
- Team on Microsoft Teams
 - The code to join is available after you subscribe to the course on docenti.unina

CLASS ON MICROSOFT TEAMS



Join Code:

zeq2cwe

COMMUNICATION PROCESS

Students → Teachers

- Send us **email messages**
 - Put «[INGSW]» or «[SWE]» at the beginning of the subject
 - Do not forget to say who you are and to sign your email
 - Check out the course materials and the course description on the institutional website **before** sending us emails (do not waste your time and ours!)
- Come see us during office hours
 - Details available on our Institutional websites
- **NO TEAMS CHAT MESSAGES**
 - They are a pain to manage and we **will** miss them, despite our best efforts



COURSE ASSESSMENT AND GRADES

To pass Software Engineering, you need to:

- Pass a **written exam**
 - Exercises and questions on the course contents (samples will be provided).
- Prepare and discuss a **group project**
- For each part, you will receive a single grade for both Module A and B
- You will need a **passing grade** ($\geq 18/30$) in both parts
- The **final grade** will be determined as the **average** of the two grades

WRITTEN EXAM

- **The written exam consists in questions and numerical exercises**
 - Topics include all the contents covered during the course
- You may also pass the written exam by taking **two partial exams**
 - One **midterm partial** (tentatively some day in October 28 – November 7)
 - One **final partial** (last days of December 2024 – first days of January 2025)
 - The midterm partial will focus on topics covered up to the date of the exam
 - The final partial will focus on topics covered in the second half of the course
 - You will need a passing grade ($\geq 18/30$) in both partials
 - The final **written exam grade** is determined as the **average of the partials**

GROUP PROJECT

- **Group size:** 2 or 3 members
 - Larger groups are required to implement more features
 - Individual groups allowed only in documented and justified cases (e.g.: worker-students), on a case-by-case basis
- A single theme, with different variations for each group

THE GROUP PROJECT AS A ROLE-PLAYING EXERCISE

The goal is to provide a realistic - yet controlled - environment to test your **software engineering** skills

- **Client:** the teachers
 - We'll behave as typical clients. E.g.: we may provide inconsistent requirements
- **Software Company:** the student group

THE GROUP PROJECT AS A ROLE-PLAYING EXERCISE

You will be asked to apply software engineering and to develop an **high-quality software product**

- Not (only) code!
- You'll replicate the whole software process
 - Requirement Engineering
 - System Design
 - Object/UI Design
 - Implementation
 - Testing


GROUP PROJECT DISCUSSION

- After submitting the project, each group will have to discuss it with the teachers
- The project discussion is articulated in three phases
 1. **Technical Presentation** of the Project (Slides)
 - 10 minutes to convince us you developed a high quality product and made good choices
 2. **Live demonstration**: show us the application running on one of your laptops
 3. **Discussion** of design and implementation choices
- **Collective Ownership**
 - The **entire group** is **responsible** for the **entire project**

PROJECT AND DISCUSSION GRADING

- You'll be graded based on:
 - The overall quality of the produced software
 - The overall quality of the produced deliverables
 - The overall quality of the final presentation
 - Your ability to respect the provided instructions and interact with the clients (us) professionally
- You will need a passing grade on **all** the required deliverables

(NEW) INTERMEDIATE PROJECT SUBMISSIONS

- Designed for students **attending the course regularly** and planning to take the full exam in the Winter session (Jan–Mar 2026).
 - Involves a **significantly reduced set of functionalities** to implement and **lighter documentation requirements**, tailored to fit students' semester workload.
 - Includes intermediate deadlines, announced well in advance on:
 - www.docenti.unina.it
 - The course's Microsoft Teams channel.
-  Missing deadlines will result in exclusion from this mode.

CHEATING POLICY AND ACADEMIC MISCONDUCT

- Every submitted artifact is automatically processed with plagiarism and AI-generated content detection tools
- If two or more projects are deemed to be too similar, they will be invalidated. The involved groups will receive a new project.

TIMING CONSTRAINTS

- You can take the written exam and the project discussion independently
 - Either do the written exam first and the discussion later, or vice-versa
 - After you discuss the project, you have one year to pass the written exam
 - After you pass the written exam, you can discuss the project within its deadline

TEXTBOOKS AND COURSE MATERIALS

- Lecture slides and materials will be made available on Microsoft Teams
- Slides are not intended as a replacement for studying on textbooks!
- Suggested textbooks by topic are reported as follows:
 - **General Parts**
 - I. Sommerville. *Software Engineering*, Pearson
 - R. S. Pressman. *Software Engineering: A Practitioner's Approach*, VII ed. McGraw Hill
 - **Object-oriented design**
 - C. Larman, *Applicare UML e i Pattern - Analisi e Progettazione orientata agli Oggetti*, V ed. Pearson, 2020.

TEXTBOOKS AND COURSE MATERIALS

- Suggested textbooks by topic are reported as follows:
 - **Coding**
 - R. Martin, Clean code. *Guida per diventare bravi artigiani nello sviluppo agile di software*, Apogeo, 2018
 - **UML**
 - Stevens Rod Pooley, *Usare UML*, Addison Wesley, 2008.
 - **Additional topics**
 - P. Amman, J. Offutt. *Introduction to software testing*, Cambridge University Press, II ed. 2017.
 - E. Gamma, R. Helm, R. Johnson, J. Vissides. *Design patterns*, Addison Wesley
 - **Human-Computer Interaction and Usability**
 - B. Shneiderman et al., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 6th edition, Pearson
 - J. Nielsen, *Usability Engineering*, Morgan Kaufmann ed.

REFERENCES AND SUGGESTED READINGS

- **The Humble Programmer** by Edsger W. Dijkstra

<https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html>

