Model Based Systems Engineering - MBSE
(2025/2026)

- *Teacher*:

  Prof. Andrea D'Ambrogio

- *Objectives*:
  - provide methods and techniques to consider the production of software-intensive systems as the result of an engineering process (*software systems engineering*)
  - illustrate principles, standards and technologies of *model-driven engineering*
- *Exams*:
  - 2 dates at the end of the I semester
  - 2 dates at the end of the II semester
  - 2 dates in September
- *Teaching Material*:
  - lecture notes (posted on the **MS Teams** platform)

Systems Engineering [1]

- Systems engineering is an interdisciplinary approach and means to enable the realization of *successful systems*
- It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing, and disposal
- SE considers both the business and the technical needs of all customers with the goal of *providing a quality product that meets the user needs*

[1] INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 5th ed.

Model Based Systems Engineering

- The *formalized application of modeling* to support system requirements, design, analysis, verification, and validation activities throughout development and later life cycle stages
- MBSE enhances the conventional document-based approach to obtain:
  - improved communications among stakeholders
  - increased ability to manage system complexity
  - improved product quality
  - reduced cycle time
  - reduced risk
  - enhanced knowledge capture and reuse

What is a system?

- A purposeful collection of inter-related components working together to achieve some common objective
- A system may include software, mechanical, electrical and electronic hardware and be operated by people
- System components are dependent on other system components
- The properties and behaviour of system components are inextricably inter-mingled

System categories

- Technical computer-based systems
  – Systems that include hardware and software but where the operators and operational processes are not normally considered to be part of the system
- Socio-technical systems
  – Systems that include technical systems but also operational processes and people who use and interact with the technical system. Socio-technical systems are governed by organisational policies and rules
- Socio-technical software-intensive systems
  – Socio-technical systems in which software represents the largest segment in terms of development cost and time, development risk or functionality

Software Systems Engineering

- Discipline for software production founded on well-known engineering principles (design and validation)
- Essential to consider software as an industrial product
- When missing we observe:
  - software products not providing the expected quality
  - reduced competitiveness:
    - late delivery
    - budget overrun

A young discipline…

- Electrical and electronic engineers, interested in building computers, regarded programming as something to be done by others – either scientists who wanted the numerical results or mathematicians interested in numerical methods
- Engineers viewed programming as a trivial task, akin to using a calculator
- Many refer to programming as a “skill” and deny that engineering principles must be applied when building software

The Unconsummated Marriage[1]

- Unconsummated marriage between…
  - computer science (programming theory) and
  - engineering principles (design and validation)
- Software engineering should wed a subset of computer science with the concepts and discipline taught to other engineers
  - Engineers must accept that they don't know enough computer science
  - Computer scientists must recognize that being an engineer is different from being a scientist, and that software engineers require an education very different from their own

(1) D.L. Parnas, Software Engineering: An Unconsummated Marriage, Comm. of the ACM, Sept. 1997

The Unconsummated Marriage

- Successful marriage example: *chemical engineering*
  - a marriage of chemistry with classical engineering areas (such as thermodynamics, mechanics, and fluid dynamics)
  - nowadays chemical engineering is not regarded as a branch of chemistry
- SwEng, term coined about 50 years ago
  - NATO conference at Garmisch, Germany (1968)
  - to testify the need of considering software production as the result of an engineering effort

Results of the NATO Conference

- Programming is neither science nor mathematics
- Programmers are not adding to our body of knowledge, they build products
- Using science and mathematics to build **products** for others is what engineers do
- Software is a major source of problems for those who own and use it. The problems are exactly those to be expected when products are built by people who are *educated for other professions* and believe that building things is not their “real job”

Typical Aspects of SW Products (1)

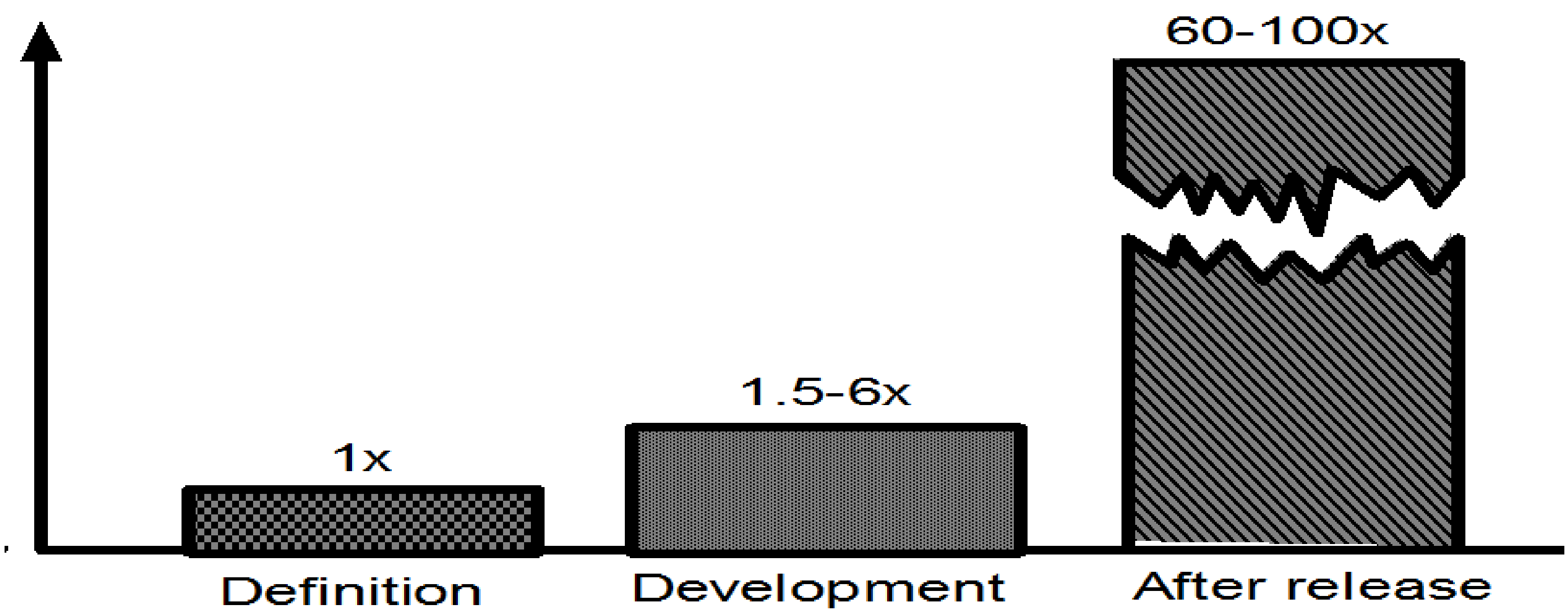
- ***ACCIDENTAL*** difficulties (can be solved by technology advancements)
  - attitude
  - maintenance
  - specification and design
  - teaming

SW lifecycle = 3 Stages, 6 Phases

- SW production = development + maintenance
- Development (stage 1) = 6 phases
    1. Requirements definition
    2. Requirements specification (or analysis)
    3. Planning
    4. Design (architectural and detailed)
    5. Coding
    6. Integration
- Maintenance (stage 2)
    - covers 60% of lifecycle costs
- Phasing-out/Retirement (stage 3)

The impact of change

- The impact of change depends on the phase during which the change is accommodated
- Changes during later phases have a severe impact on cost and may be over an order of magnitude more expensive than the same change requested earlier



Where is Testing?

- Not explicitly mentioned at stage 1
- Not a separate phase
- Activity to be carried out along the entire lifecycle
- Two types:
  - **Verification (*at the end of each phase*)**
  - **Validation (*at the end of development, typically*)**
- Verification = are we building the *product right*?
- Validation = are we building the *right product*?

Defect Removal Efficiency (DRE)

- DRE refers to the percentage of *defects found before delivery* of the software to its actual clients or users
- If the development team finds 900 defects before delivery and the users find 100 defects in a standard time period after release (normally 90 days), then the DRE value is 90 percent
- The *U.S. average* in 2016 is only about 92 percent (values change according to the software lifecycle model)

Typical Aspects of SW products (2)

- ***ESSENTIAL*** difficulties (not solved by science and technology advancements)
  - complexity
  - conformity
  - changeability
  - invisibility

Typical Aspects of SW Products (3)

- ***COST***
  - cost vs. product size
  - cost vs. replicas
  - cost vs. market size

SW Product Cost Issues

- *Cost* proportional to the *square of size* ($C=aS^2$)
  - building two products of size S/2 has a total cost lower than building a single product of S
- Building a replica has a *null* cost
- Putting in the market a product of *double* size
  - requires a *price four* times greater if the market size is kept unchanged
  - requires a *market size four* times greater if the price is kept unchanged

Definitions (1)

- *SW product* (or *SW, briefly*) =
  = Code + Documentation
- *Artefact* = *intermediate* SW product
  - requirements definition document
  - requirements analysis document
  - design document
- *Code* = *final* SW product
- *SW system* = integrated set of SW products

Definitions (2)

- *Customer* = who commissions SW production
- *Developer* = who builds the SW product
- *User* = who uses the SW product
- *SW types*
  - *Internal* SW
    - customer and developer belong to the same organization
  - *Contract* SW
    - customer and developer belong to different organizations
  - SW *for the market*
    - the customer is the market

SW Reliability Issues

- Informally
  - SW product credibility
- Formally
  - probability that the product works "correctly" in a given timeframe (*mission time*)

Defect, Failure, Error

- **Defect (Bug)**
  - anomaly present in a SW product
- **Failure**
  - unexpected behavior of a SW product due to the presence of one or more defects
- **Error**
  - wrong action of the developer who introduces a defect into the SW product (because of ignorance, lack of attention, etc.)

SW Reliability

- Intuitively:
  - a SW product with many defects is not reliable
- It is obviuos that:
  - SW reliability improves as long as defects are fixed

SW Reliability Characteristics (1)

- The relationship between:
  - observed reliability and
  - number of hidden (dormant) defects

  is not easy
- Removing defects from the product parts less used (executed)
  - has a negligible impact on the observed reliability

The rule 10-90

- Experiments carried out on SW programs of large size show that:
  - 90% of the execution time is spent by executing only 10% of the program instructions
- Such 10% is referred to as:
  - the *core* of the program

SW Reliability Characteristics (2)

- The reliability improvement due to the removal of a single defect:
  - depends on where that defect is located (in other words, if that defect is part or not of the program core)

SW Reliability Characteristics (3)

- Then, the observed reliability depends on:
  - how the software product is used
  - in technical terms, the *operational profile*

SW Reliability Characteristics (4)

- Due to the fact that different users may use the SW product according to different operational profiles:
  - the defects that are revealed to a user
    - may not be revealed to a different user
- In conclusion, SW reliability:
  - depends on the user

HW Reliability vs. SW Reliability (1)

- **SW failures are due to:**
  – the presence of defects
  – software does not wear out
- **HW failures are typically due to:**
  - components wear out
  - components not behaving as specified
  - components damages

HW Reliability vs. SW Reliability (2)

- HW defects examples
  - a damaged resistor
  - a short circuit in a capacitor
  - a logic gate that halts (on 1 or 0)
- To fix an HW defect:
  - the failed component is replaced

HW Reliability vs. SW Reliability (3)

- SW defects are hidden (dormant)
  - the SW product keeps on failing
    - if the necessary fixes are not carried out
- Due to the different effects
  - the metrics valid for HW reliability cannot be extended to SW reliability

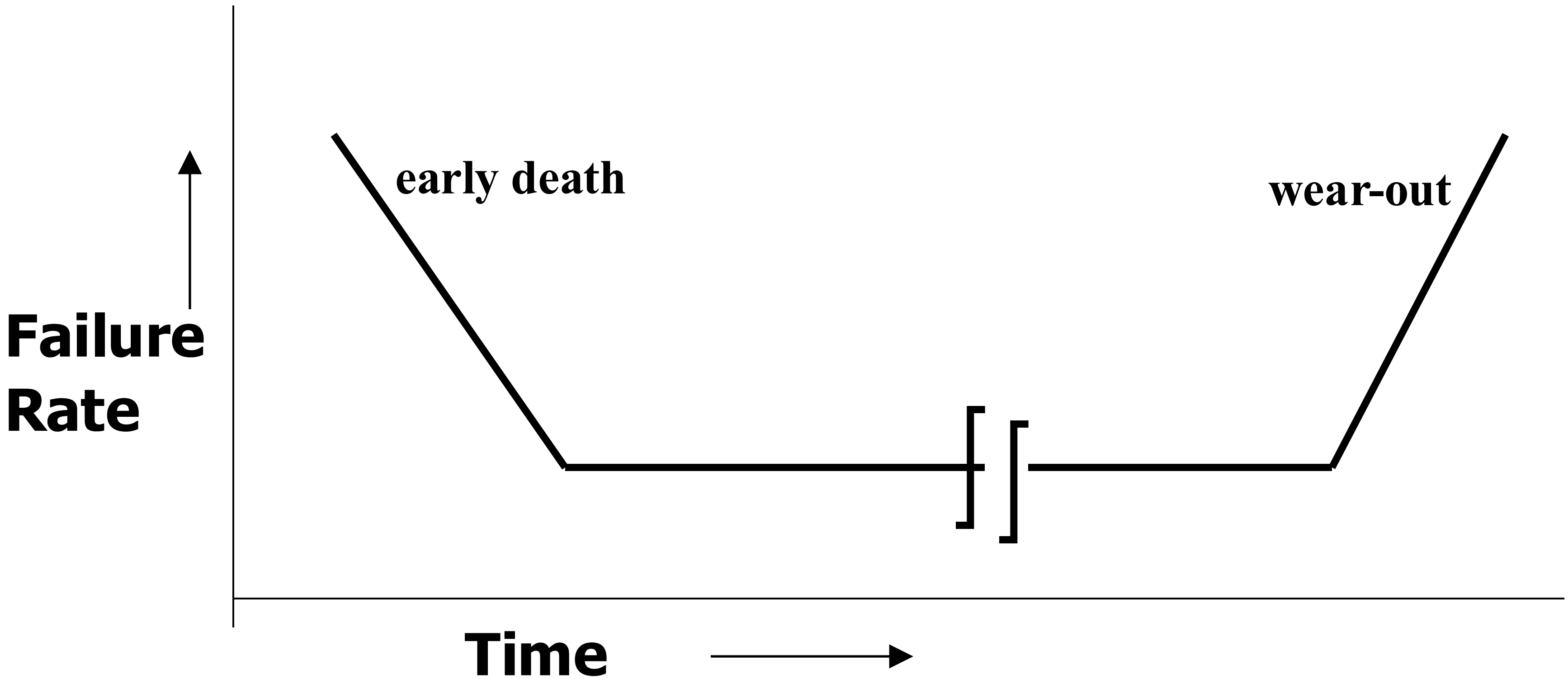
HW Reliability vs. SW Reliability (4)

- After fixing the HW product
  - its reliability returns to be as it was before
- After fixing the SW product
  - its reliability may result improved or worsened

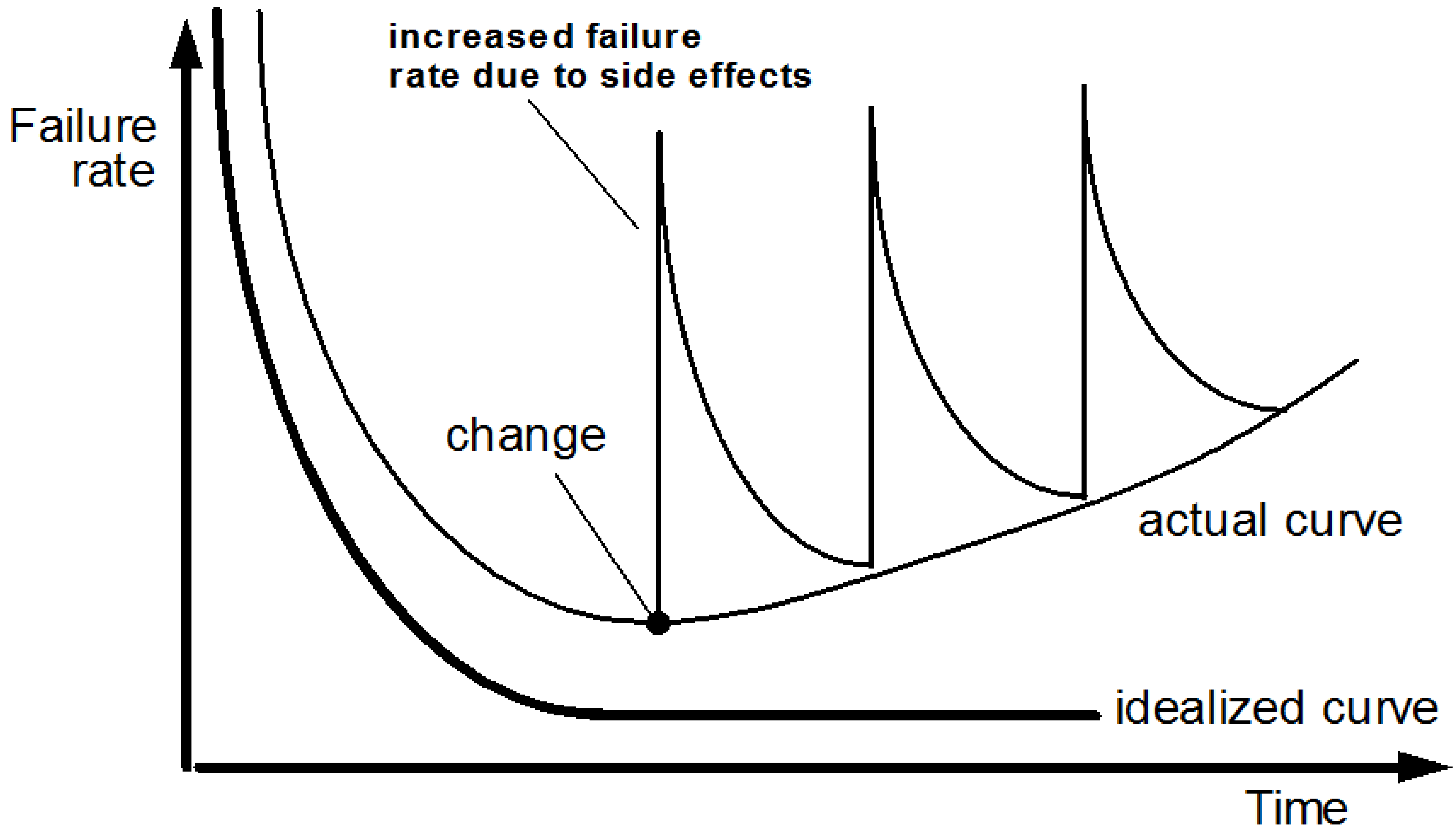
HW Reliability vs. SW Reliability (5)

- ***HW*** reliability objective
  - stability (i.e., keeping failure rate constant)
- ***SW*** reliability objective
  - reliability growth (i.e., decreasing failure rate)

HW Failure Rate (bathtub curve)



SW Failure Rate



SW Availability

- Percentage of the time that the SW product has been usable during its lifecycle
- Depends on
  – the number of failures that occur
  – the time required to fix the product

SW Reliability/Availability Significance

- Important metrics for systems in which
  - service outages lead to economic and/or social losses (critical systems)
    - transportation systems
    - air traffic control systems
    - energy production and distribution systems
    - communication systems
    - etc.

Conclusion (1)

- Over the last 50 years SW production has evolved according to the following periods
  - **craftsmanship** period, during which SW is developed by single and creative programmers
  - **pre-industrial** period, during which SW is developed by small groups of highly specialized professionals
  - **industrial** period, during which SW production and maintenance is properly planned and coordinated, and designers/developers are supported by automated tools

Conclusions (2)

- The term «software engineering» has been coined in 1968, during the NATO conference held at Garmisch (Germany), to testify the need of regarding software production as the result of an engineering effort
- The IEEE Standard 610.12-1990 (*glossary of software engineering terminology*) defines software engineering as:
  1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software
  2) The study of approaches as in 1)

Conclusions (3)

- A SW product can be considered as a set of elements that contribute to build a “configuration” of:
  - programs
  - documents
  - multimedia data
- It is built by software engineers who apply a process to eventually get products of expected quality
- An engineering approach has to be applied, as well as for other products
- SW characteristics:
  - is ”engineered”
  - does not wear out
  - is complex, must conform, is changeable and invisible

Conclusions (4)

- What can we make to meet the software quality requirements?
- What can we make to balance the ever increasing demand by keeping under control the allocated budget?
- What can we make to effectively update legacy applications?
- What can we make to avoid delayed product releases?
- What can me make to successfully apply new technologies?

Software Engineering methods, tools and techniques contribute to provide an answer to the aforementioned questions, with the objective of building software products of expected/required quality

The SW myths (…to debunk)

- If we get behind schedule, we can add more programmers and catch up
- A general statement of objectives is sufficient to begin writing programs; we can fill in the details later
- Once we write the program and get it to work, our job is done
- Until I get the program "running" I have no way of assessing its quality
- Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down