# Chapter 1

Introduction

# Declaration

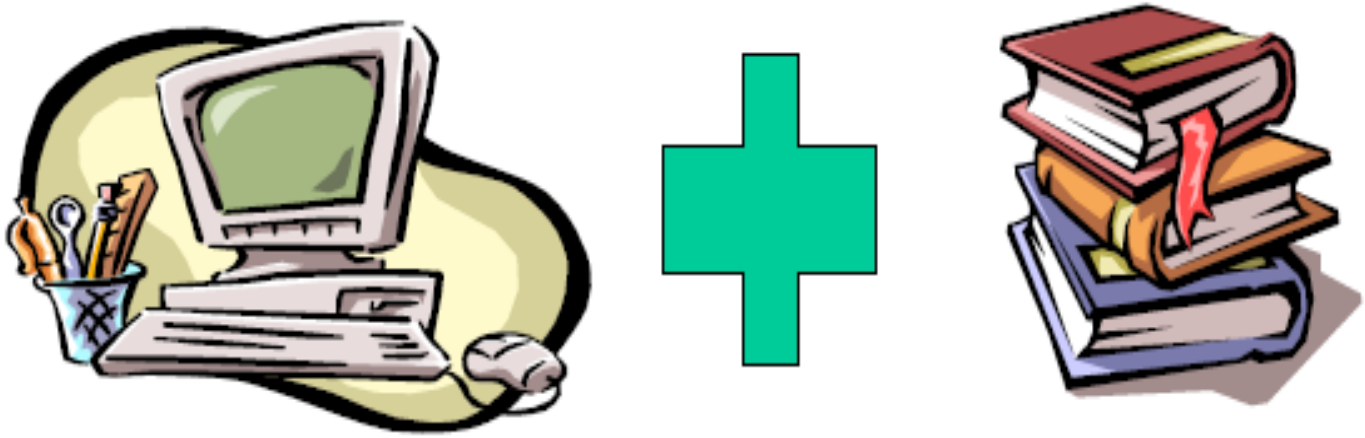
- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

# Software Engineering

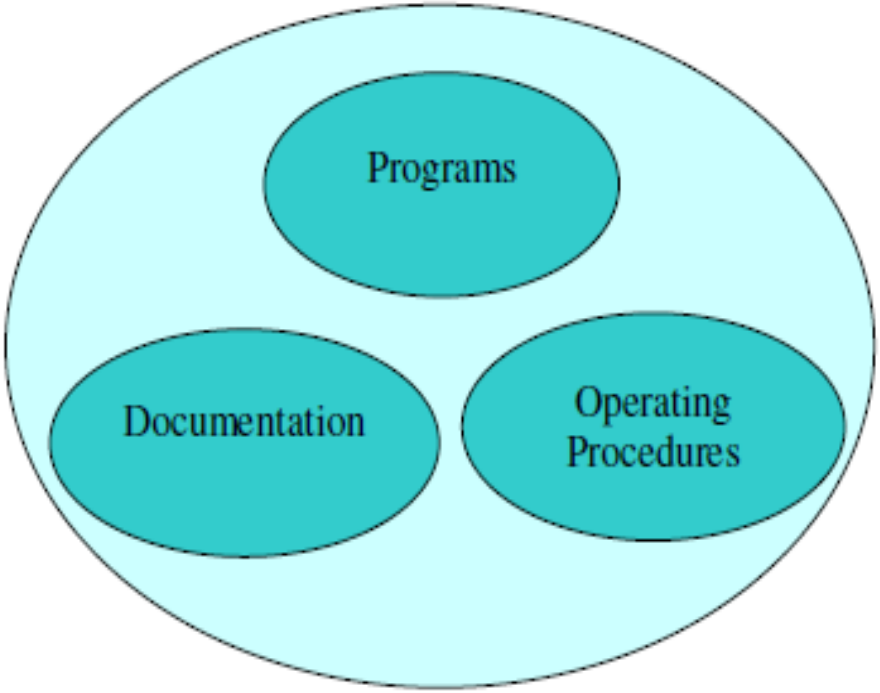
- In Software Engineering two words are there
  - Software
  - Engineering
- First I will discuss about Software and then Engineering

# What is software?

- Computer programs and associated documentation



# What is software?




Software=Program+Documentation+Operating Procedures

Components of software

# What is Software?

- Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information and

(3) documentation that describes the operation and use of the programs.

# What is Software?

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't "wear out."
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

# Engineering

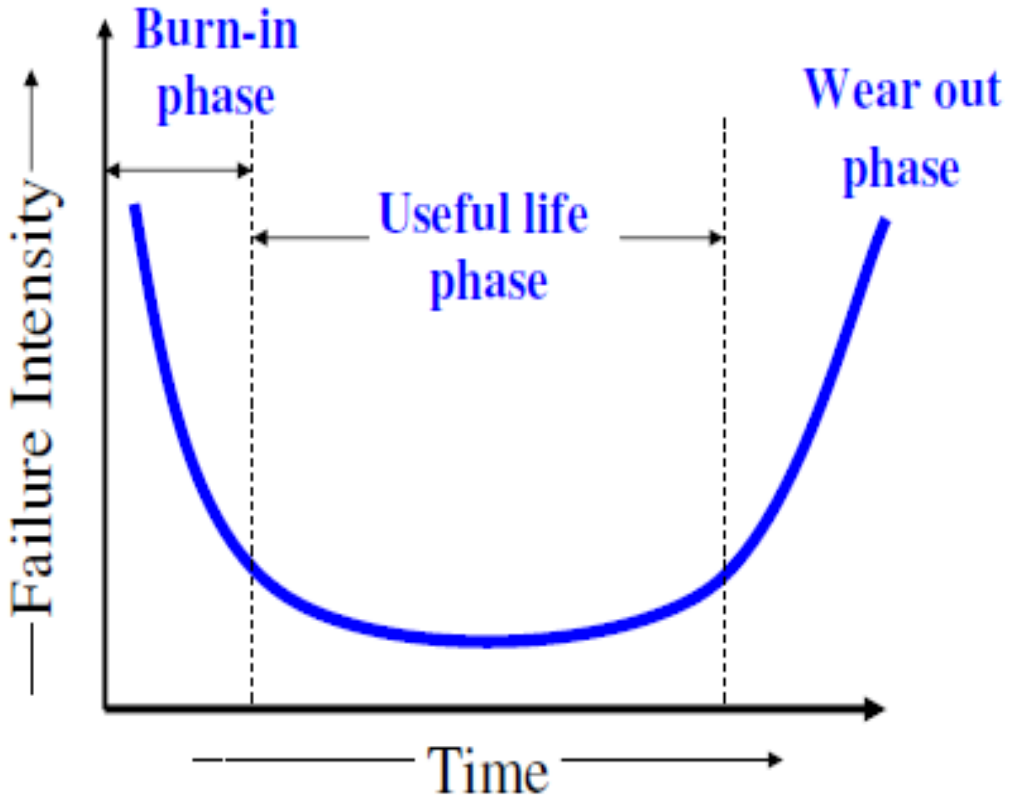
- Engineering is the systematic application of scientific principles, mathematical methods, and empirical evidence to innovate, design, develop, and maintain structures, machines, materials, systems, and processes. It involves the creative and analytical use of knowledge to solve practical problems, improve existing solutions, and meet human and societal needs in a safe, efficient, and sustainable manner.

# Scientist vs Engineer

- Computer Scientist
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time…
- Engineer
  - Develops a solution for an application-specific problem for a client
  - Uses computers & languages, tools, techniques and methods
- Software Engineer
  - Works in multiple application domains
  - Has only 3 months...
  - …while changes occurs in requirements and available technology

# Software Characteristics:

✓Software does not wear out.




This is often called the “bathtub curve”

# Three Phases of the Bathtub Curve

**1. Early Failure (Infant Mortality) Phase:**

- **Description:** This initial phase has a high failure rate, typically caused by design flaws, manufacturing defects, or errors in the deployment process. As these issues are identified and corrected, the failure rate decreases.
- **Relevance in Software Engineering:** In the context of software, this phase might correspond to the period after a software release where bugs are detected and fixed, known as the "shakedown" period.

# Three Phases of the Bathtub Curve

**2. Normal Life (Useful Life) Phase:**

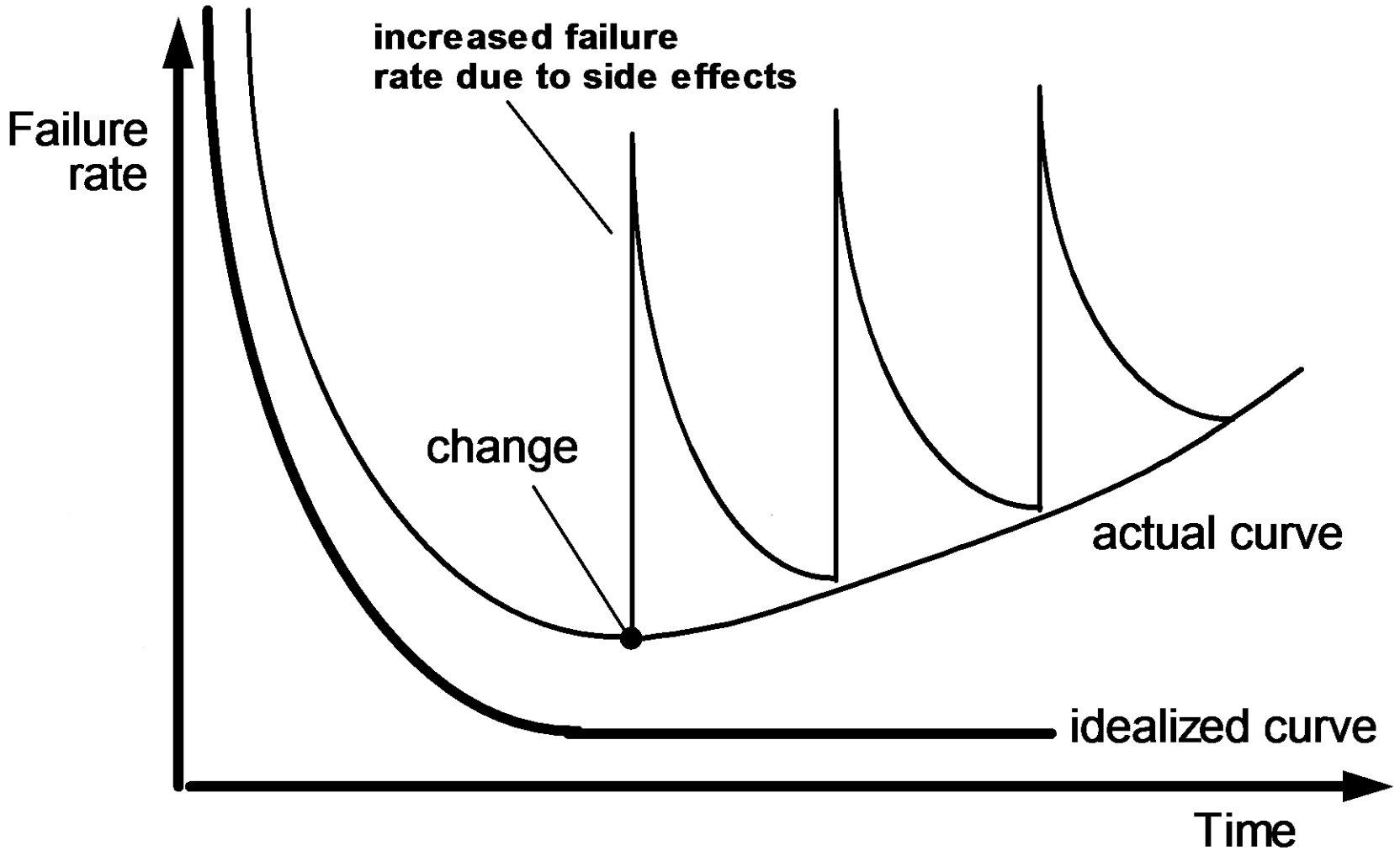
- **Description:** After the early failures are resolved, the product enters a period of stability where failures occur randomly and at a relatively low, constant rate. This phase represents the majority of the product's lifecycle.
- **Relevance in Software Engineering:** During this phase, the software runs smoothly with occasional issues, often related to unforeseen edge cases or minor defects. Maintenance and updates are performed as needed.

# Three Phases of the Bathtub Curve

**3. Wear-Out Phase:**

- **Description:** Eventually, the failure rate begins to increase again as the product ages and components start to wear out or become obsolete.
- **Relevance in Software Engineering:** In software, this phase might occur when the technology becomes outdated, or when accumulated technical debt leads to increasing instability and failures, necessitating significant refactoring, redevelopment, or replacement.

# Wear vs. Deterioration



# Wear vs. Deterioration

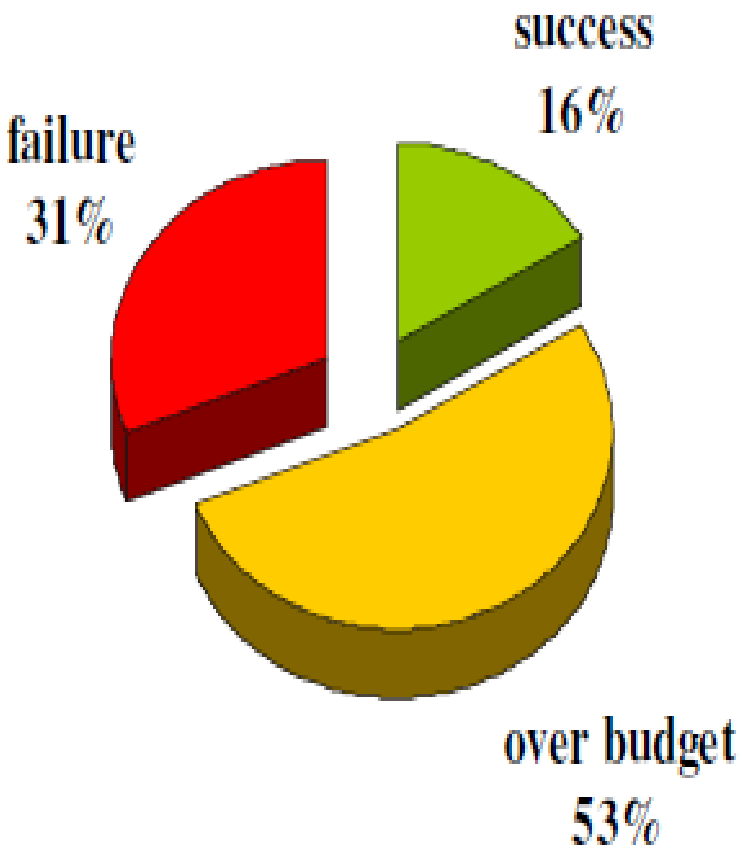
**Software never wear outs. It has deterioration.**

At its infant state ,software has high failure rate same as hardware . By time, After customization & repairing the defects , it becomes idealized or gets into the steady state or idealized state. Software and hardware defects are not same. Software defects may be happened by the unfulfilled user demands, slow, bugs and many more. Here the Idealized Curve shows the Idealized State. And the Actual Curve shows the increased failure rate gradually due to the defeats during software customization and modification (Change). This (change) causes Software deterioration. So, software does not wear out, it just have deterioration.

# Software Crisis (Need of Software Engineering)

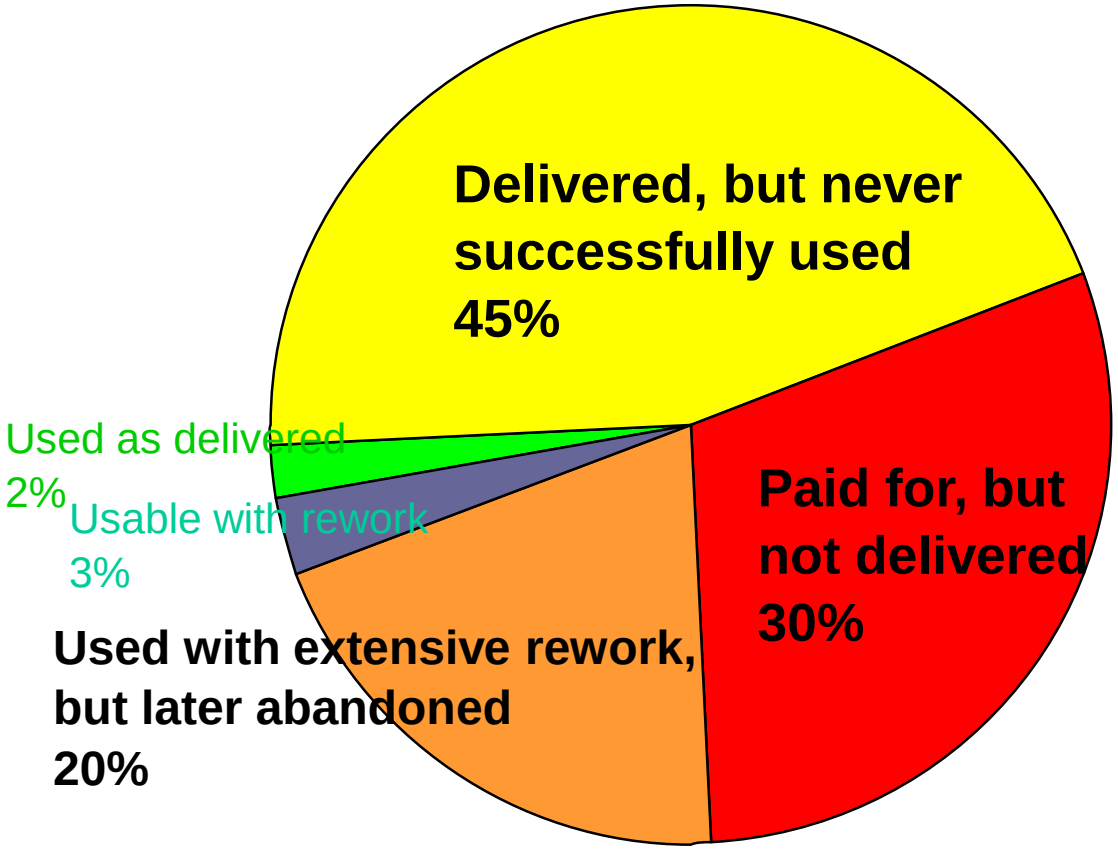
- A survey conducted by IBM in 2000 reports that
  - 31% of the projects get cancelled before completion.
  - 53% of the projects exceed to the budget schedule even up to 189%.
  - 94% of the projects have many restarts.
- Software failures receive a lot more publicity than software engineering success stories.

- Software industry is in Crisis!



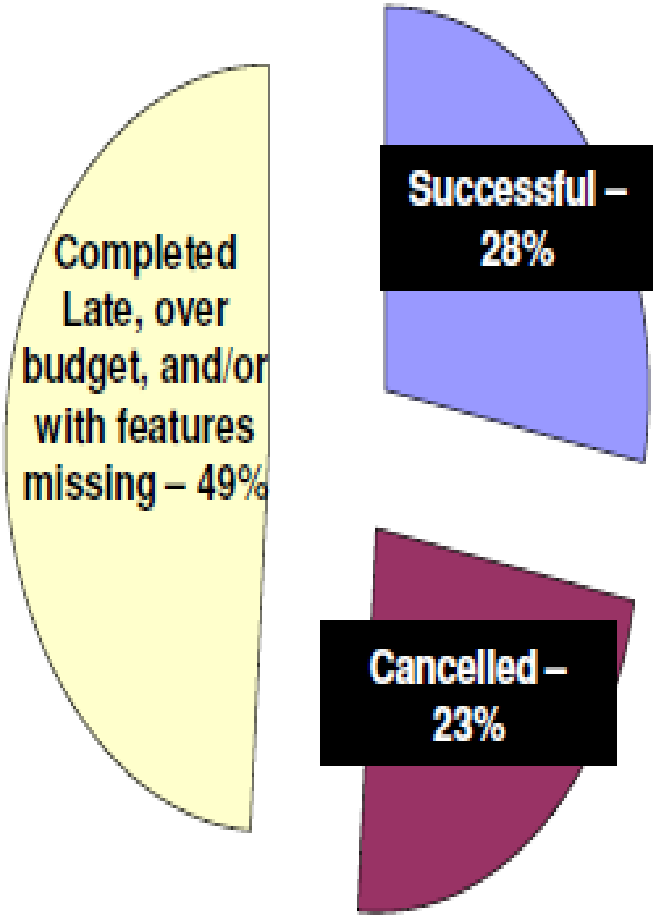
# Why Software Engineering?

*9 software projects totaling $96.7 million: Where The Money Went*
*[Report to Congress, Comptroller General, 1979]*




Take a look at the Standish Report (The "Chaos" Report)

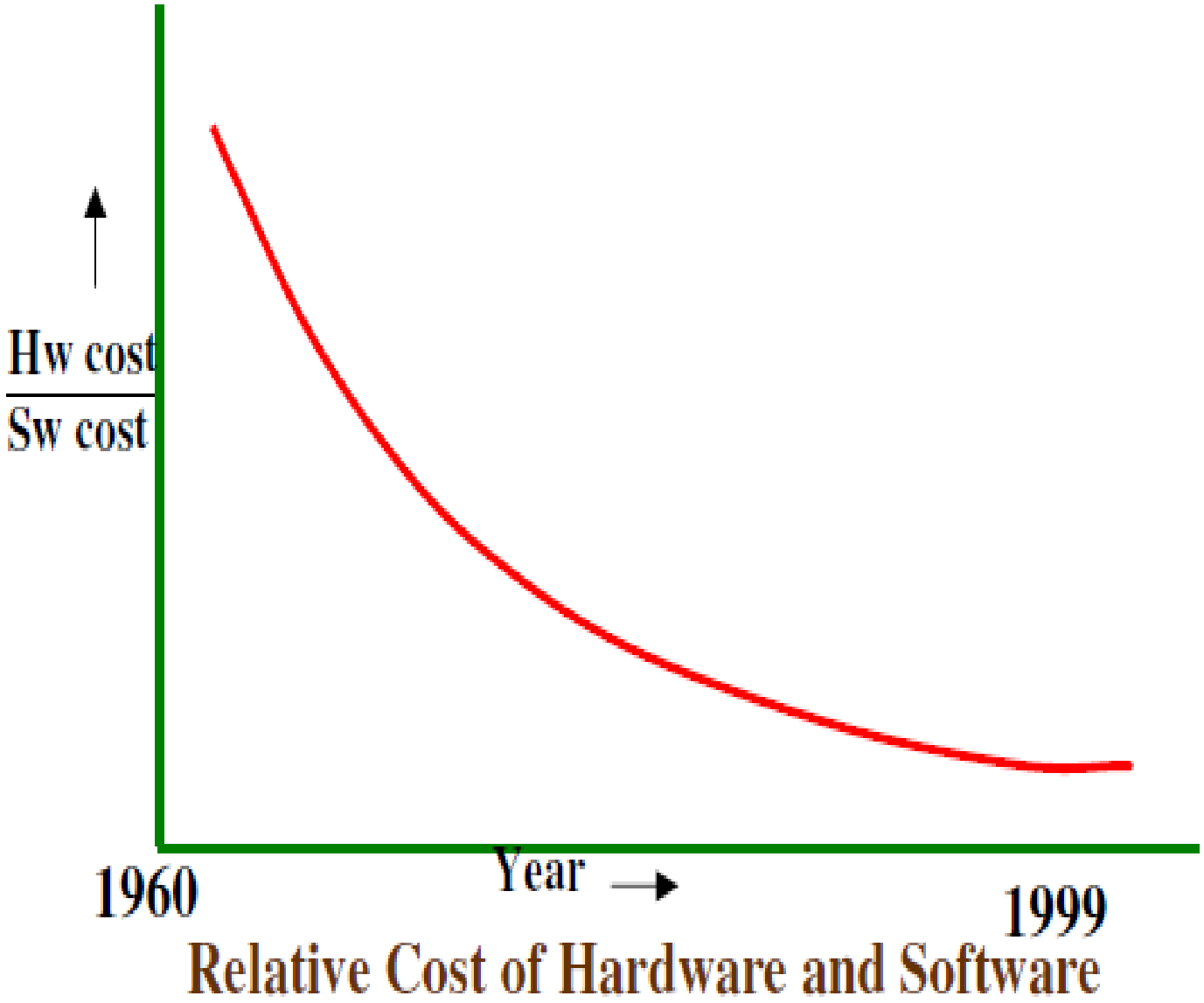
# This is the SORRY state of Software Engineering Today!




- Data on 28,000 projects completed in 2000

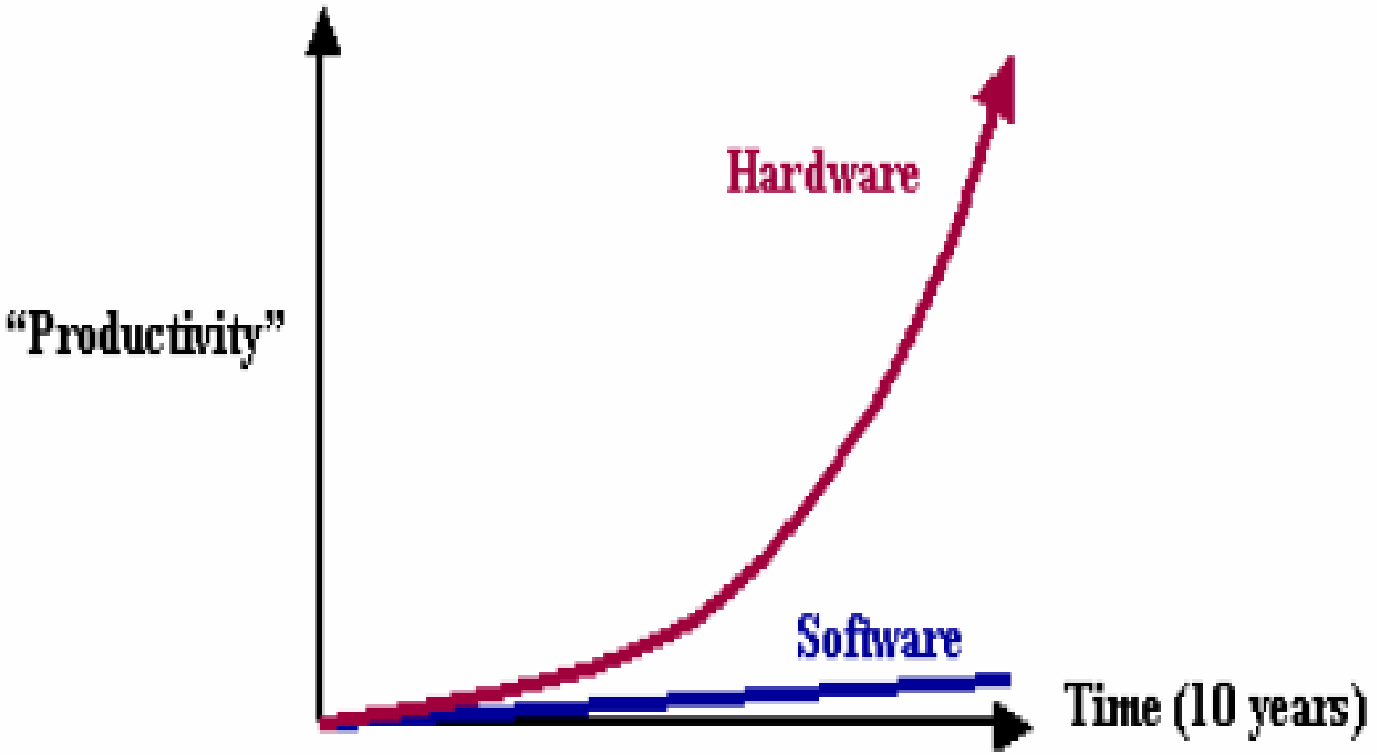
# Software Crisis

- Software cost has not dropped as rapidly as the hardware cost.
- Developers in H/W industry have significantly affected cost, quality, productivity and reliability of hardware.
- In S/W industry, there is no single development either in technology or in mgmt. technique that by itself promises even one order of magnitude improvement in the productivity, cost and reliability.
- Use of CASE tools was thought to be a development giving boost to above factors but the results are highly unsatisfactory.




Relative Cost of Hardware and Software

- Unlike Hardware
  - Moore's law: processor speed/memory capacity doubles every two years



# Software Engineering: A Problem Solving Activity

- **Analysis**: Understand the nature of the problem and break the problem into pieces
- **Synthesis**: Put the pieces together into a large structure

- For problem solving we use
- Techniques (methods):
  - Formal procedures for producing results using some well-defined notation
- Methodologies:
  - Collection of techniques applied across software development and unified by a philosophical approach
- Tools:
  - Instrument or automated systems to accomplish a technique

# Software Engineering: Definition

Software Engineering is a collection of techniques, methodologies and tools that help with the production of

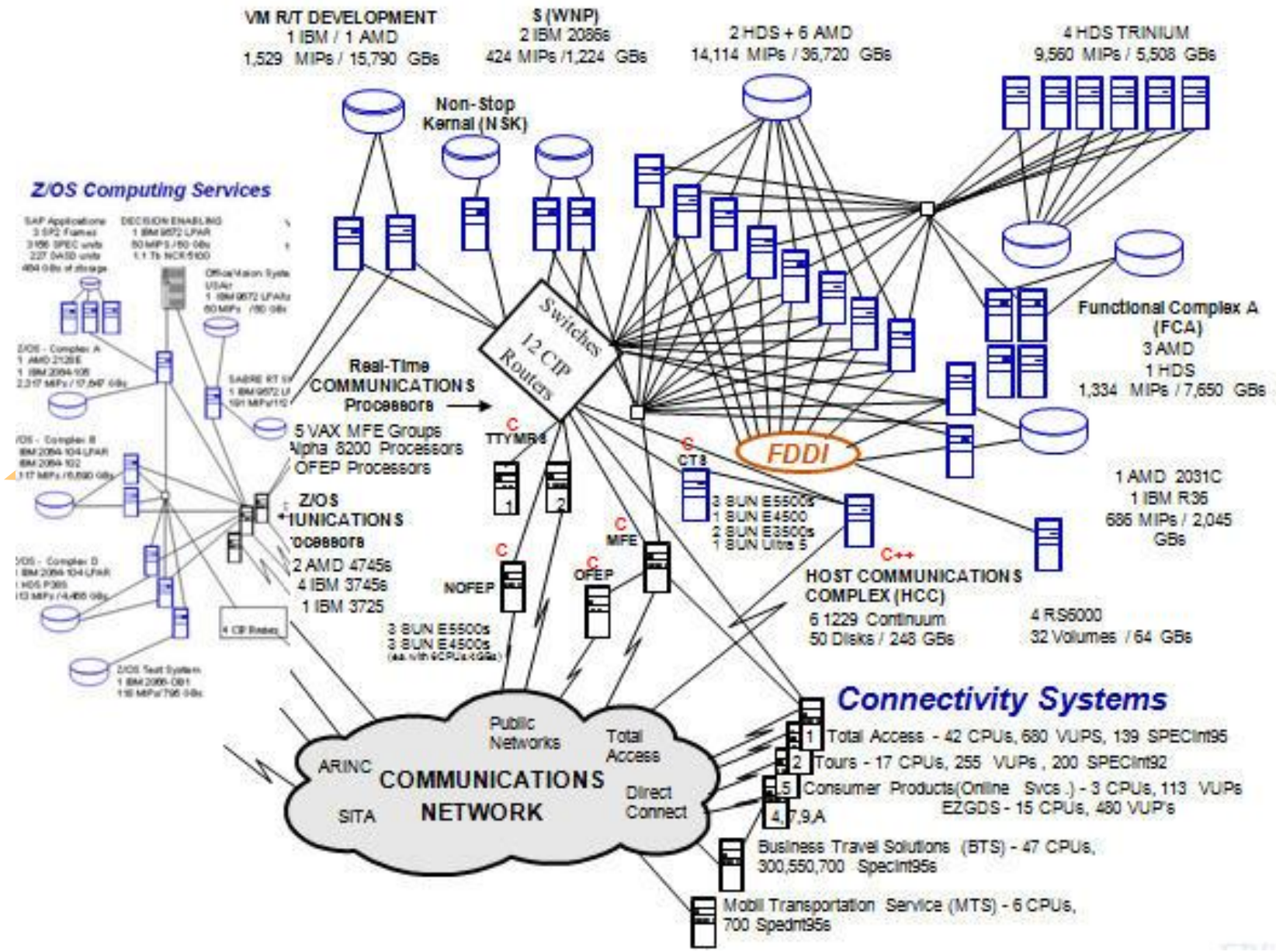
- a high quality software system
- with a given budget
- before a given deadline

while change occurs.

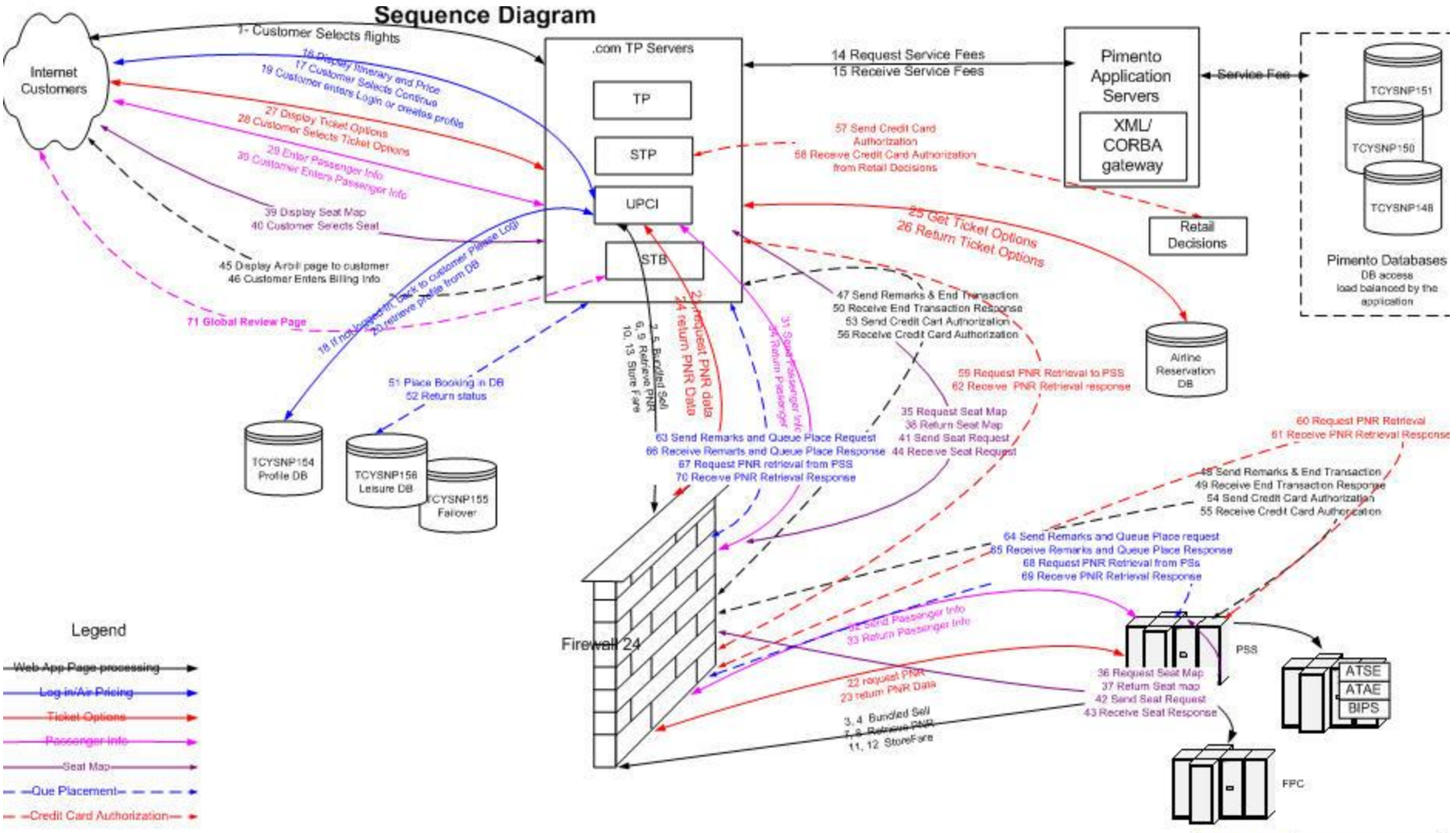
# Factors affecting the quality of a software system

- **Complexity:**
  - The system is so complex that no single programmer can understand it anymore
  - The introduction of one bug fix causes another bug
- **Change:**
  - The “Entropy” of a software system increases with each change: Each implemented change erodes the structure of the system which makes the next change even more expensive (“Second Law of Software Dynamics”).
  - As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base.

# Complex Server Connections



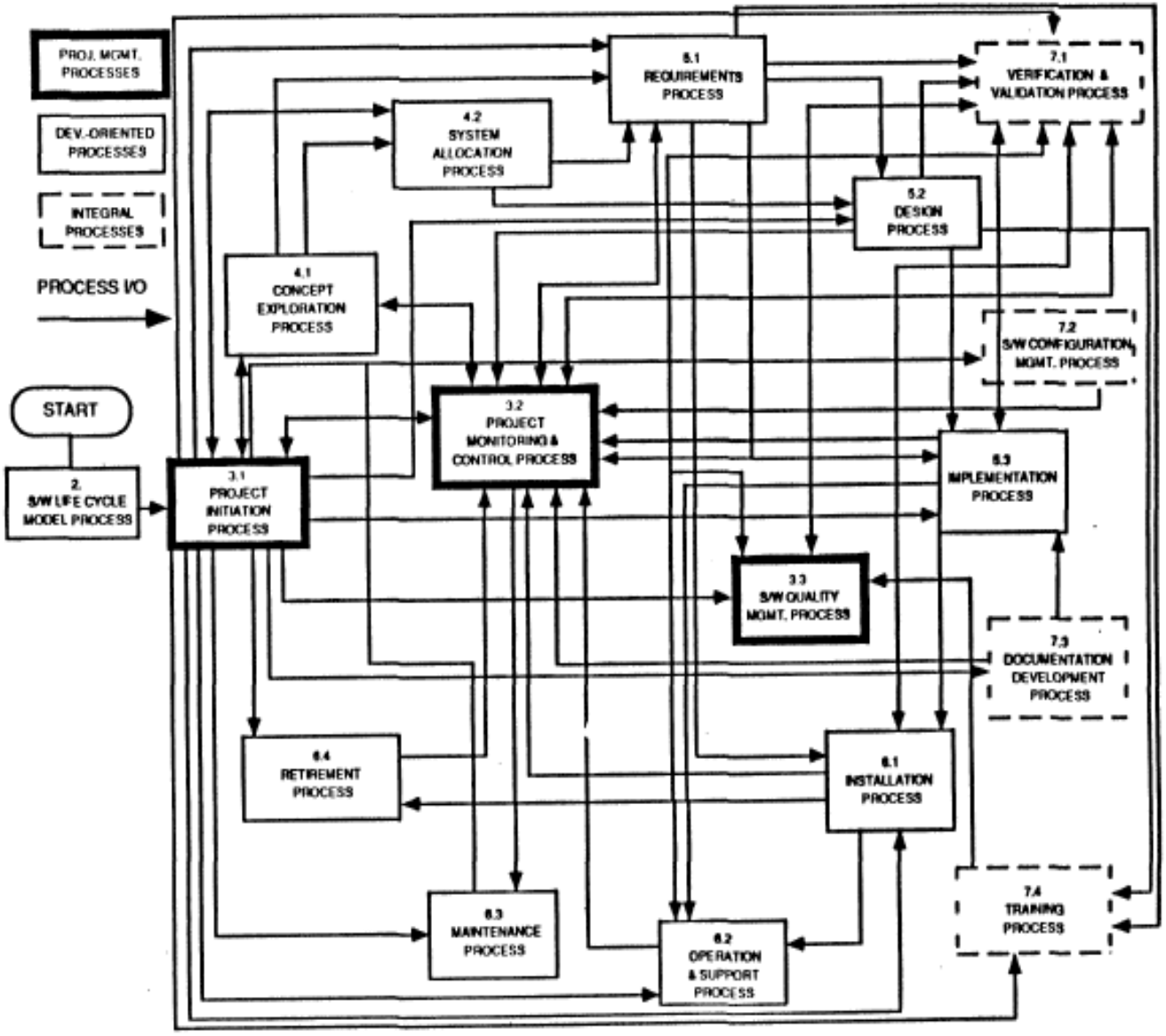
# Complex Message Flow



# Dealing with Complexity

1. Abstraction
2. Decomposition
3. Hierarchy

*What is this?*



# Introduction to the notion of software engineering as a product

Software engineering as a discipline often revolves around the idea that software is not just a set of code or a technical solution, but a **product** that must meet certain standards, fulfill specific requirements, and deliver value to its users. This perspective is crucial because it shifts the focus from merely writing code to ensuring that the software is reliable, maintainable, and evolves effectively over time.

**Key Aspects**

1. **Product Lifecycle:** Just like any other product, software has a lifecycle that includes planning, design, development, testing, deployment, maintenance, and eventual retirement. Each stage is critical in ensuring the software meets user needs and business objectives.

# Introduction to the notion of software engineering as a product

2. **Requirements Gathering**: Understanding the needs of the users and stakeholders is fundamental in software engineering. This phase involves defining what the software should do, its features, and how it should interact with other systems. These requirements form the blueprint for the product.
3. **Design and Architecture:** The software's architecture and design are like the blueprint of a physical product. This phase involves creating the structure that will support the software's functionalities and ensure it can scale, integrate, and perform as needed.
4. **Development:** The actual coding or construction of the software product happens during this phase. Developers work to translate the design into functional software, ensuring that each component aligns with the specified requirements.

# Introduction to the notion of software engineering as a product

5. **Quality Assurance (QA):** Before the product is released, it undergoes rigorous testing to ensure it is free from defects, performs as expected, and provides a good user experience. QA is essential in ensuring the software product is reliable and meets the desired quality standards.
6. **Maintenance and Support:** Once the software is deployed, it continues to be a living product. This phase involves fixing bugs, updating features, and ensuring the software adapts to new requirements or technologies. Maintenance is critical for the longevity and relevance of the product.
7. **User Experience (UX):** A software product must not only function correctly but also be user-friendly. The design and functionality should cater to the end-users, ensuring that the product is easy to use, efficient, and satisfies the user's needs.

# Introduction to the notion of software engineering as a product

8. **Product Management:** Managing a software product involves overseeing its entire lifecycle, making decisions about when to add new features, when to retire old ones, and how to align the software with business goals and market demands.
9. **Versioning and Evolution:** Software products typically undergo multiple versions and updates. Versioning helps in tracking changes, while the product's evolution ensures it remains competitive and relevant as technology and user needs change.
10. **Delivery Models:** The software product can be delivered in various models, such as standalone applications, web-based platforms, or as a service (SaaS). The choice of delivery model impacts how the product is developed, maintained, and consumed by users.

# Essential characteristics of Well-Engineered Software Product

A well-engineered software product should possess the following essential characteristics:

1. **Efficiency:**
   The software should not make wasteful use of system resources such as memory and processor cycles.
2. **Maintainability:**
   It should be possible to evolve the software to meet the changing requirements of customers.
3. **Dependability:**
   It is the flexibility of the software that ought to not cause any physical or economic injury within the event of system failure. It includes a range of characteristics such as reliability, security and safety.

# Essential characteristics of Well-Engineered Software Product

4. **In time:**
   Software should be developed well in time.
5. **Within Budget:**
   The software development costs should not overrun and it should be within the budgetary limit.
6. **Functionality:**
   The software system should exhibit the proper functionality, i.e, it should perform all the functions it is supposed to perform.
7. **Adaptability:**
   The software system should have the ability to get adapted to a reasonable extent with the changing requirements.

# Introduction to the Engineering aspects of Software Products

The engineering aspects of software products involve the systematic application of engineering principles to the design, development, testing, and maintenance of software. This approach ensures that software is not only functional but also reliable, scalable, and maintainable over time. The discipline of software engineering applies these principles to manage the complexity of software development and ensure the end product meets both user and business needs.

Key Engineering Aspects

1. Systematic Design and Architecture:
    - **Design Principles:** Engineering software products begins with designing a robust architecture that serves as the backbone for the software. This involves selecting appropriate design patterns, modularizing the system, and ensuring the architecture supports scalability and flexibility.

# Introduction to the Engineering aspects of Software Products

- **Scalability:** The design should accommodate future growth in terms of users, data, and functionality. This means thinking ahead and engineering a system that can scale without requiring significant rework.

2. Reliability and Robustness:
   - **Error Handling:** Ensuring that the software can handle unexpected situations gracefully without crashing. This involves thorough error detection, correction mechanisms, and recovery procedures.
   - **Testing:** A critical engineering aspect involves rigorous testing at multiple levels (unit, integration, system, acceptance) to identify and fix bugs before the software is released. Automated testing and continuous integration practices are often employed to maintain high reliability.

# Introduction to the Engineering aspects of Software Products

3. **Performance Optimization:**
   - **Efficiency:** Engineering software products requires a focus on optimizing performance. This includes efficient use of resources such as memory and processing power, as well as minimizing latency and maximizing throughput.
   - **Profiling and Optimization:** Tools and techniques are used to profile software performance, identify bottlenecks, and optimize code for better performance.
4. **Maintainability and Extensibility:**
   - **Code Quality:** High-quality code is essential for maintaining and extending a software product. This includes adhering to coding standards, writing clear and concise code, and documenting the codebase effectively.
   - **Modular Design:** Software should be engineered in a modular way, allowing components to be updated or replaced without affecting the entire system. This makes the software easier to maintain and adapt to new requirements.

# Introduction to the Engineering aspects of Software Products

5. Security:
   - **Security Engineering:** Protecting the software from malicious attacks is a fundamental engineering concern. This involves implementing security best practices, such as data encryption, secure authentication, and regular security audits.
   - **Vulnerability Management:** Identifying and addressing security vulnerabilities promptly to protect the software and its users from potential threats.
6. User Experience (UX) Engineering:
   - **Human-Centered Design:** Engineering software products includes ensuring they are user-friendly. This involves considering the user experience during the design phase, conducting usability testing, and iterating on the design based on user feedback.
   - **Accessibility:** Ensuring that the software is accessible to users with different abilities, complying with accessibility standards.

# Introduction to the Engineering aspects of Software Products

7. Project and Process Management:
   - **Agile and DevOps:** Modern software engineering often involves iterative development processes like Agile and continuous delivery practices enabled by DevOps. These approaches help manage the complexity of software projects and ensure continuous improvement and rapid delivery.
   - **Risk Management:** Identifying, assessing, and mitigating risks throughout the software development lifecycle to ensure that potential issues do not derail the project.
8. Documentation and Knowledge Transfer:
   - **Technical Documentation:** Maintaining thorough documentation is essential for knowledge transfer, onboarding new team members, and ensuring that future maintenance and updates can be performed smoothly.
   - **User Documentation:** Providing end-users with clear and helpful documentation to maximize the software's usability and effectiveness.

# Introduction to the Engineering aspects of Software Products

7. Integration and Interoperability:
   - **System Integration:** Engineering software products often involves integrating them with other systems, services, or APIs. Ensuring that these integrations are seamless and reliable is a key engineering challenge.
   - **Interoperability:** The software should be designed to work across different platforms, devices, and environments, ensuring broad compatibility and usability.
8. Lifecycle Management:
   - **Version Control:** Engineering practices include robust version control to manage changes in the software, enabling traceability and facilitating collaboration among developers.
   - **Continuous Improvement:** Engineering doesn't stop at the initial release; it involves ongoing improvements and updates to address new requirements, fix issues, and enhance the software's capabilities.

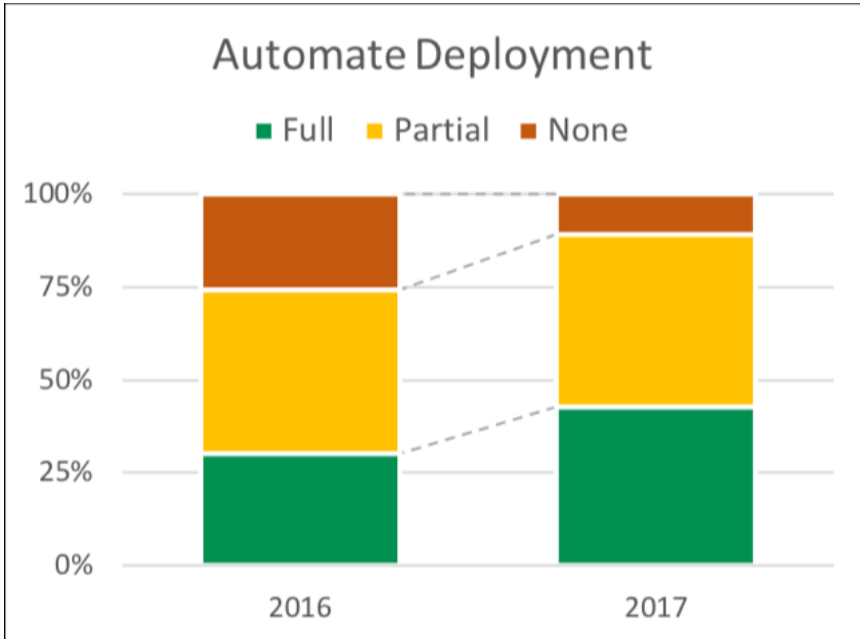
# Necessity of automation

- End-to-end automation of the software development process has gone mainstream. While automation of individual programming tasks (compilation, testing, documentation, etc.) has been part and parcel of our discipline for many years, we now see that all modern software development teams are striving to automate as much of the software development process as possible.
- Automation helps to increase development speed, limit knowledge dissipation, and build quality into every step.

# Necessity of automation

**Trend 1: The percentage of teams that do not automate deployment shrank from 26% in 2016 to 11% in 2017**

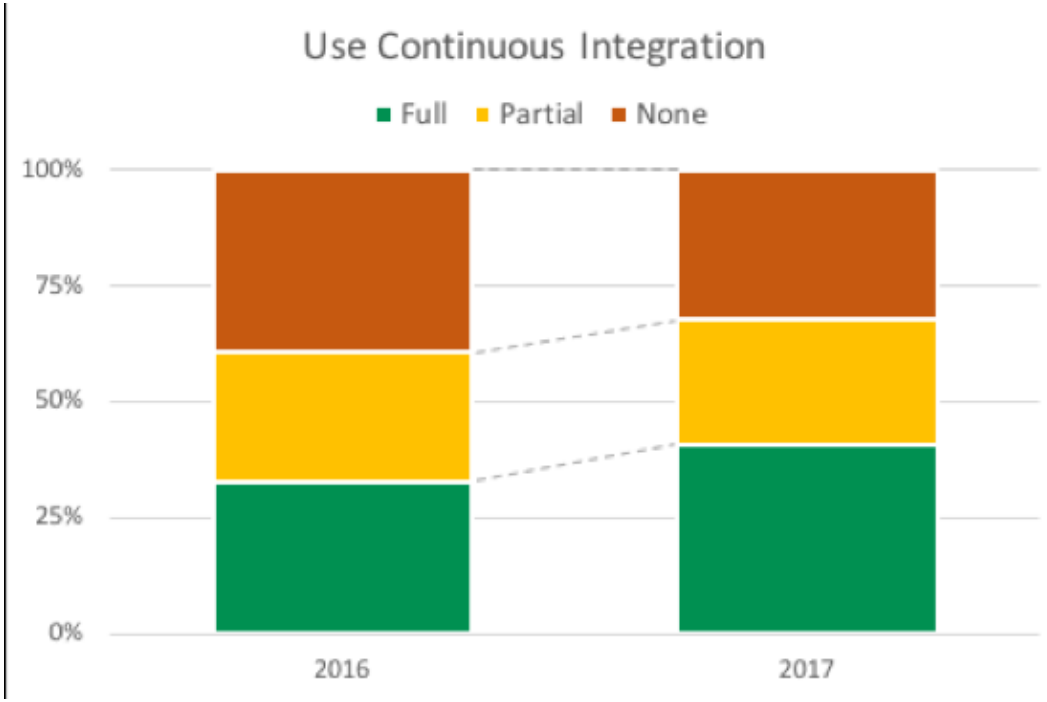
- For example, a team that deploys each new release with a single push of a button would receive a perfect score on this practice, codified as fully applied. But a team that needs to go through a limited number of well-documented manual steps would be scored as partially applied.



# Necessity of automation

**Trend 2: Teams that fully apply continuous integration (CI) now outnumber those that don't (41% versus 32% in 2017; was 33% versus 39% in 2016)**

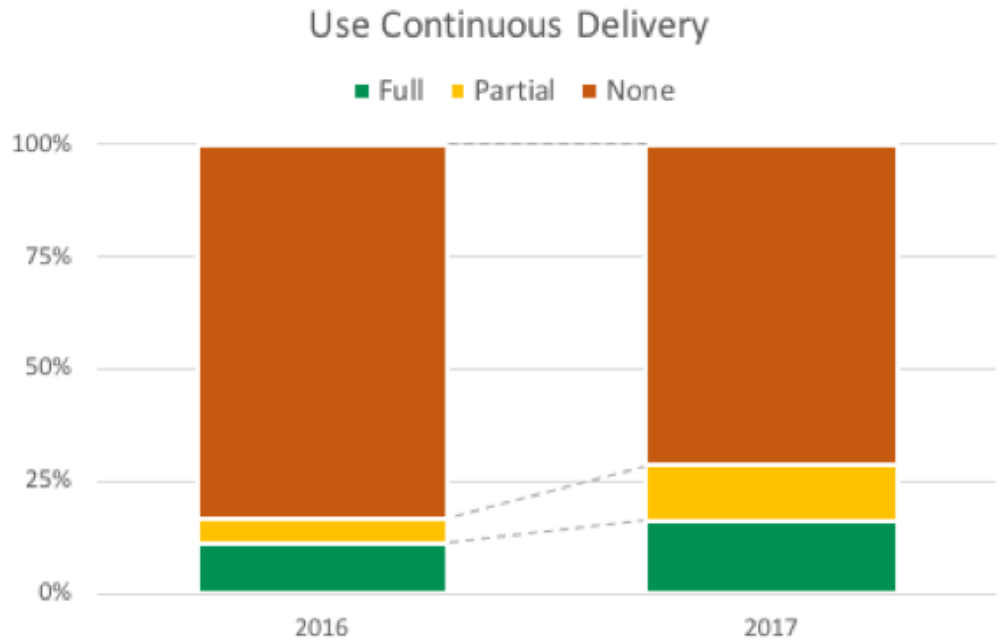
- The trend for continuous integration (automatic compilation and testing after each change) lags behind the trend for deployment automation, but overall, it shows a similar improvement.



# Necessity of automation

**Trend 3: The number of teams that apply continuous delivery (CD) is a small but growing minority (16% in 2017 versus 11% in 2016)**

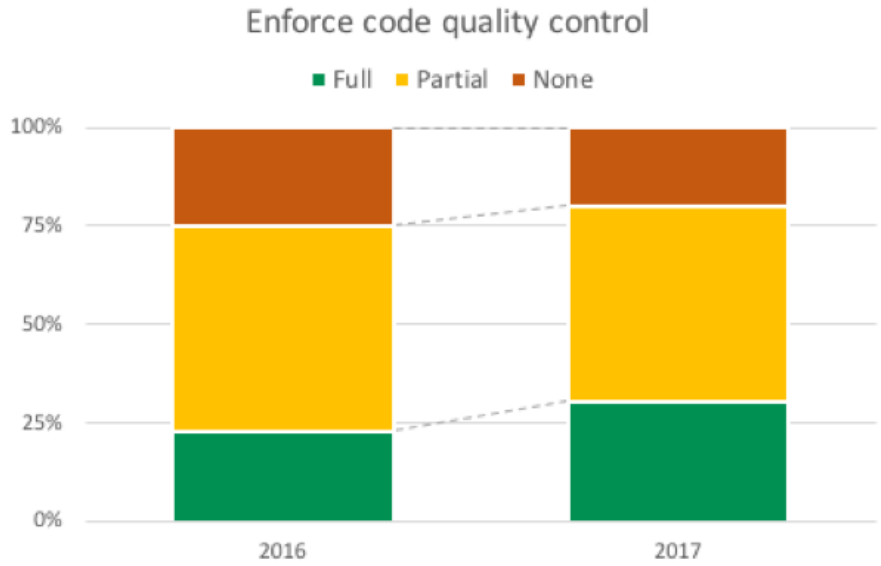
- The trend for continuous integration (automatic compilation and testing after each change) lags behind the trend for deployment automation, but overall, it shows a similar improvement.



# Necessity of automation

**Trend 4: More teams are enforcing full quality control (31% in 2017, up from 23% in 2016)**

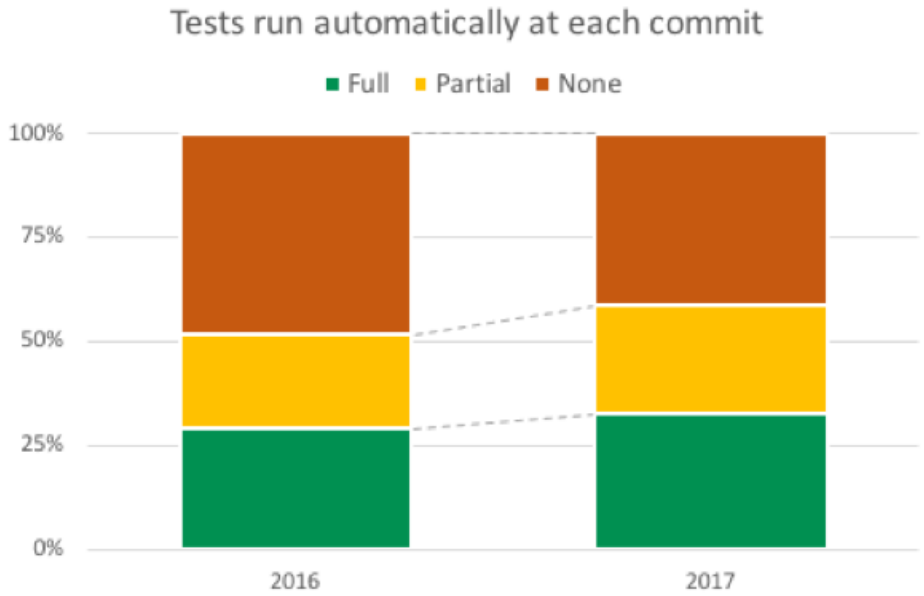
- To assess code quality control, we observe whether a team works with clear coding standards, systematically reviews code (against these standards and against principles of good design), and whether they perform automated code quality checks. Full adoption of code quality control has increased somewhat (31%, up from 23%), but 20% of teams are still producing code without adequate quality control in place.



# Necessity of automation

**Trend 5: The number of teams that change their code without proper regression testing is declining but was still a staggering 41% in 2017 (down from 48% in 2016)**

- To assess testing practices, we observe whether teams have an automated regression test suite that is being executed consistently after each change. Full adoption of this practice is increasing (33%, up from 29%). But changing code without proper regression testing is still uncannily common (41%; was 48%).



# Software Developer duties and responsibilities

- A standard Software Developer job description should include, but not be limited to:
  - Researching, designing, implementing and managing software programs
  - Testing and evaluating new programs
  - Identifying areas for modification in existing programs and subsequently developing these modifications
  - Writing and implementing efficient code
  - Determining operational practicality
  - Developing quality assurance procedures
  - Deploying software tools, processes and metrics
  - Maintaining and upgrading existing systems
  - Training users
  - Working closely with other developers, UX (User Experience) designers, business and systems analysts

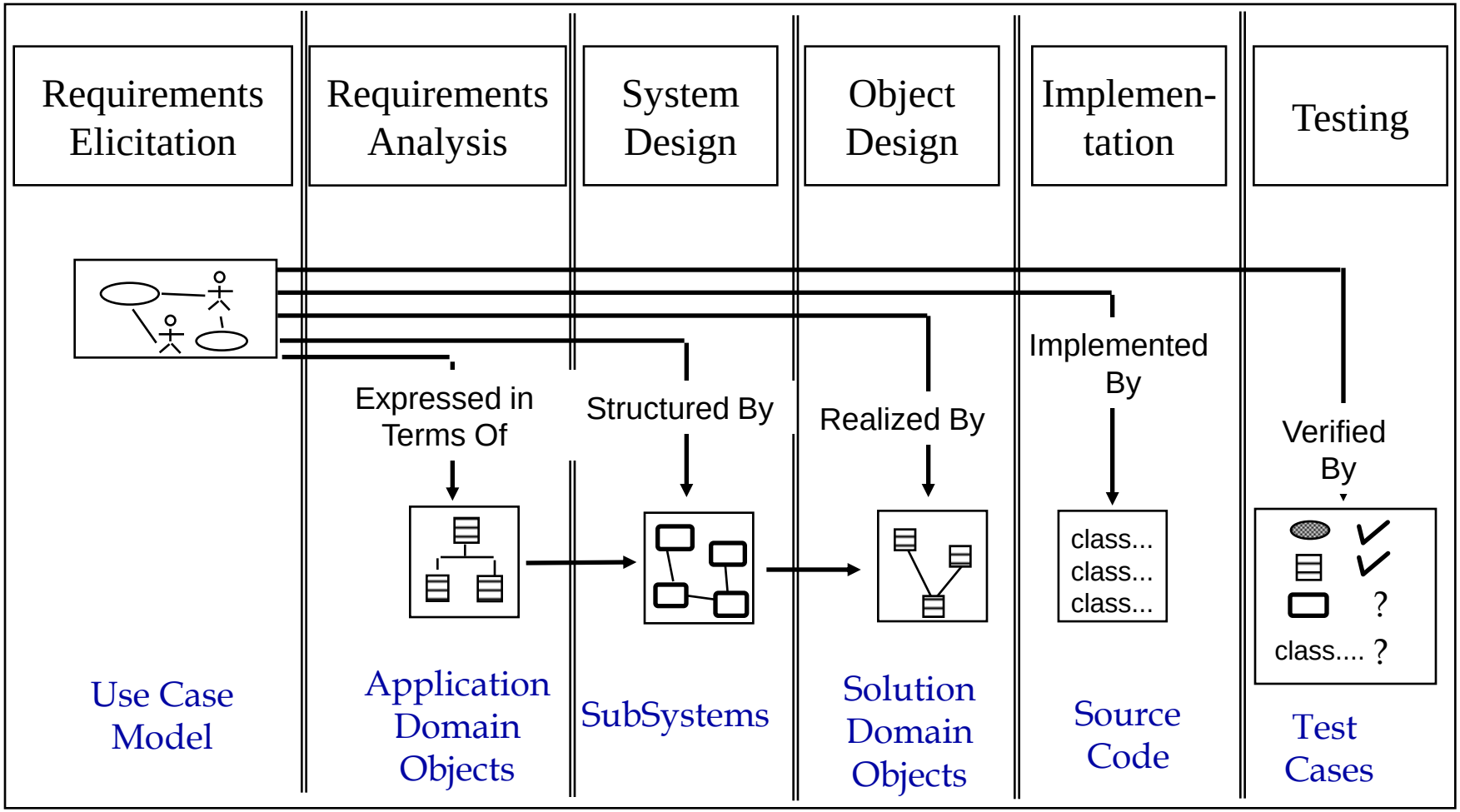
# Software Lifecycle Definition

- Software lifecycle:
  - Set of activities and their relationships to each other to support the development of a software system

- Typical Lifecycle questions:
  - Which activities should I select for the software project?
  - What are the dependencies between activities?
  - How should I schedule the activities?

# Software Lifecycle Activities

| Deliverable 0 | | | | | |
|---|---|---|---|---|---|
| Deliverable 1 | Deliverable 2 | Deliverable 3 | Deliverable 4 | Deliverable 5 | Deliverable 6 |




**Each activity produces one or more models**

# Software Engineering Phases

- Definition phase
  - focuses on what (information engineering, software project planning, requirements analysis).
- Development phase
  - focuses on how (software design, code generation, software testing).
- Support phase
  - focuses on change (corrective maintenance, adaptive maintenance, perfective maintenance, preventive maintenance).

# Software Life Cycle Phases

1. Requirements, analysis, and design phase.
2. System design phase.
3. Program design phase.
4. Program implementation phase.
5. Unit testing phase.
6. Integration testing phase.
7. System testing phase.
8. System delivery.
9. Maintenance.

# Software Development Process Models

- In software development life cycle, various models are designed and defined. These models are called as **Software Development Process Models.**
- On the basis of project motive, the software development process model is selected for development.

# Software Development Process Models

- **Following are the different software development process models:**

  1) Big-Bang model
  2) Code-and-fix model
  3) Waterfall model
  4) V model
  5) Incremental model
  6) RAD model
  7) Agile model
  8) Iterative model
  9) Spiral model
  10) Prototype model

# End of Chapter 1

Questions?