Software Engineering

Lecture 01

Topic Covered

- Evolving Role of Software
- Hardware vs. Software
- Intro. to Software Engineering
- Software characteristics
- Software Components
- Software Applications
- Evolution of Software
- Software Myths
- Software Development Life Cycle (SDLC)
- SDLC Phases

Evolving Role of Software

Software is a product

- Transforms information - produces, manages, acquires, modifies, displays, or transmits information
- Delivers computing potential of hardware and networks

Software is a vehicle for delivering a product

- Controls other programs (operating system)
- Effects communications (networking software)
- Helps build other software (software tools & environments)

What is Software ?

Software can define as:

- Instruction – executed provide desire features, function & performance.
- Data structure – to adequately manipulate operation.
- Documents – operation and use of the program.

Software products may be developed for a particular customer or may be developed for a general market.

- Software products may be
  - **Generic** - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
  - **Bespoke** (**custom**) - developed for a single customer according to their specification.

Hardware vs. Software

Hardware

- Manufactured
- wear out
- Built using components
- Relatively simple

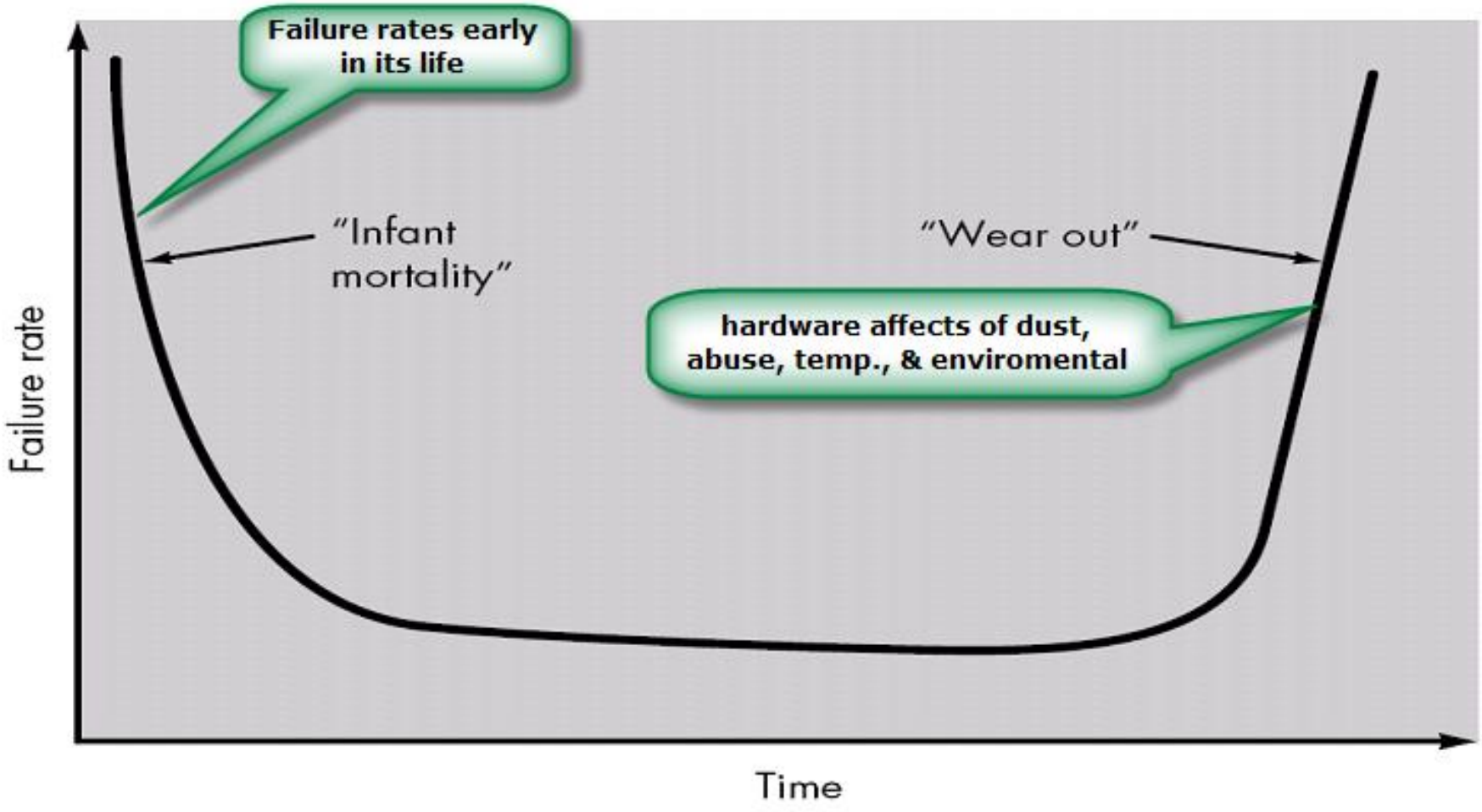
Software

- Developed/ engineered
- deteriorate
- Custom built
- Complex

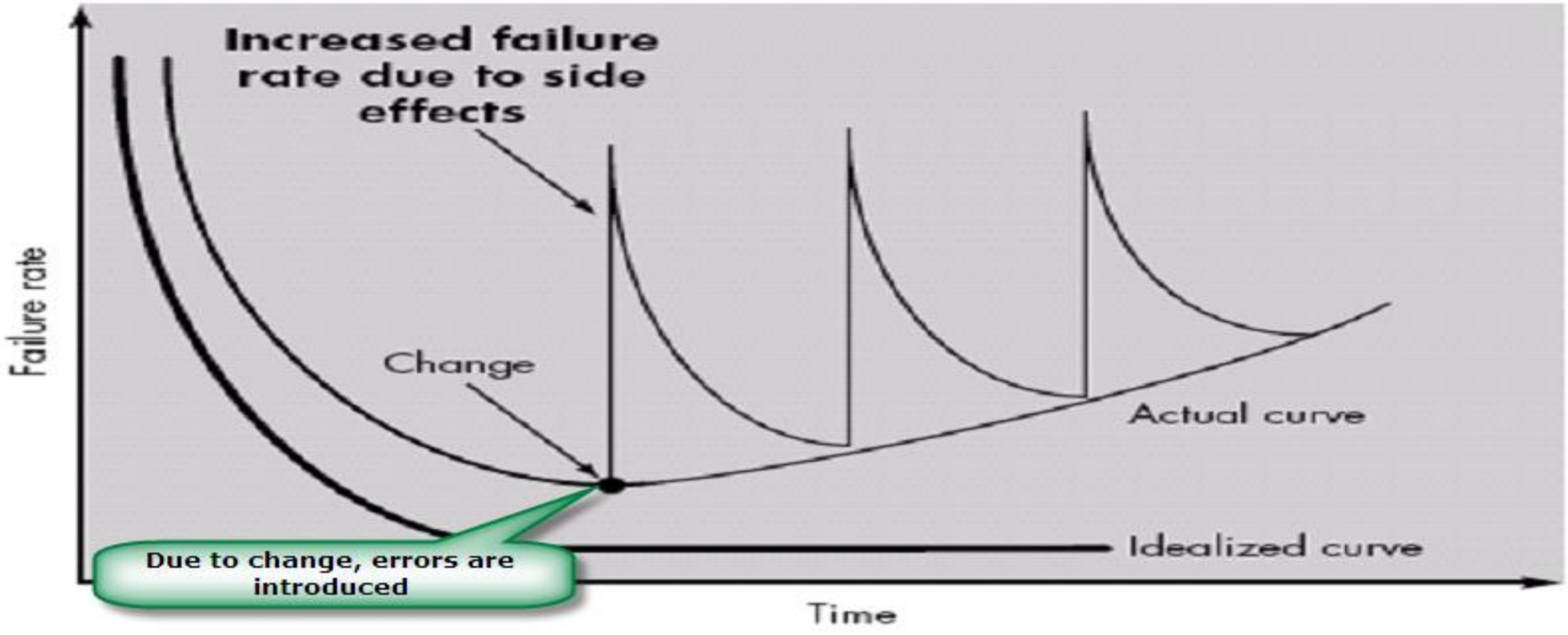
Manufacturing vs. Development

- Once a hardware product has been manufactured, it is difficult or impossible to modify. In contrast, software products are routinely modified and upgraded.
- In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.
- Unlike hardware, software costs are concentrated in design rather than production.

Failure curve for Hardware



Failure curve for Software



When a hardware component wears out, it is replaced by a spare part.

There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity

Component Based vs. Custom Built

- Hardware products typically employ many standardized design components.
- Most software continues to be custom built.
- The software industry does seem to be moving (slowly) toward component-based construction.

What is Software Engineering?

- The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints
- Note:
  - Process, systematic (not ad hoc), evolutionary…
  - Constraints: high quality, cost, time, meets user requirements

Analysis of the Definition:

- Systematic development and evolution
  - An engineering process involves applying well understood techniques in a organized and disciplined way
  - Many well-accepted practices have been formally standardized
    - e.g. by the IEEE or ISO
  - Most development work is *evolutionary*

Analysis of the Definition:

- Large, high quality software systems
  - Software engineering techniques are needed because large systems cannot be completely understood by one person
  - Teamwork and co-ordination are required
  - Key challenge: Dividing up the work and ensuring that the parts of the system work properly together
  - The end-product that is produced must be of sufficient quality

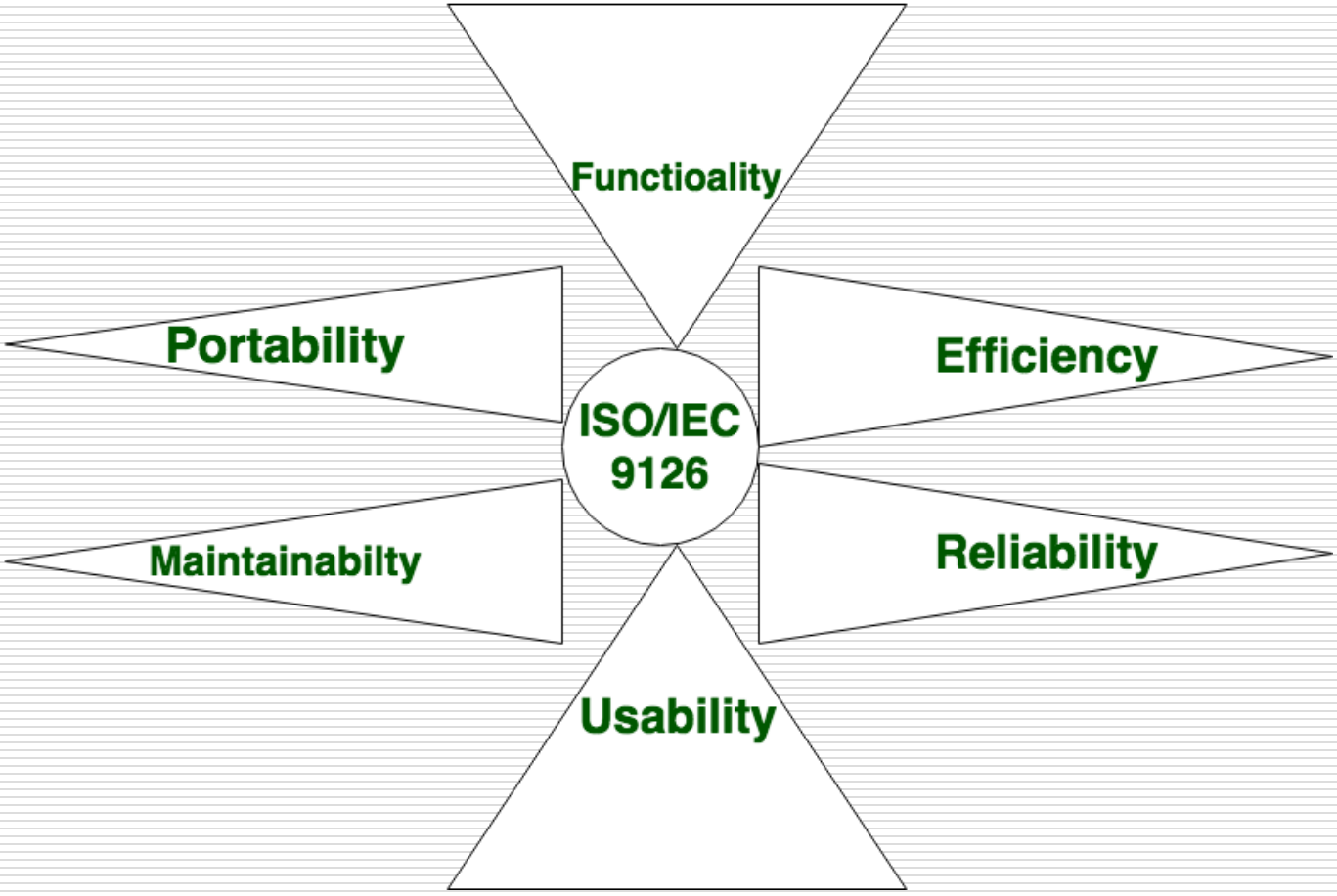
Analysis of the Definition:

- Cost, time and other constraints
  - Finite resources
  - The benefit must outweigh the cost
  - Others are competing to do the job cheaper and faster
  - Inaccurate estimates of cost and time have caused many project failures

Software characteristics

- Software is developed or engineered; it is not manufactured.
- Software does not “wear out” but it does deteriorate.
- Software continues to be custom built, as industry is moving toward component based construction.

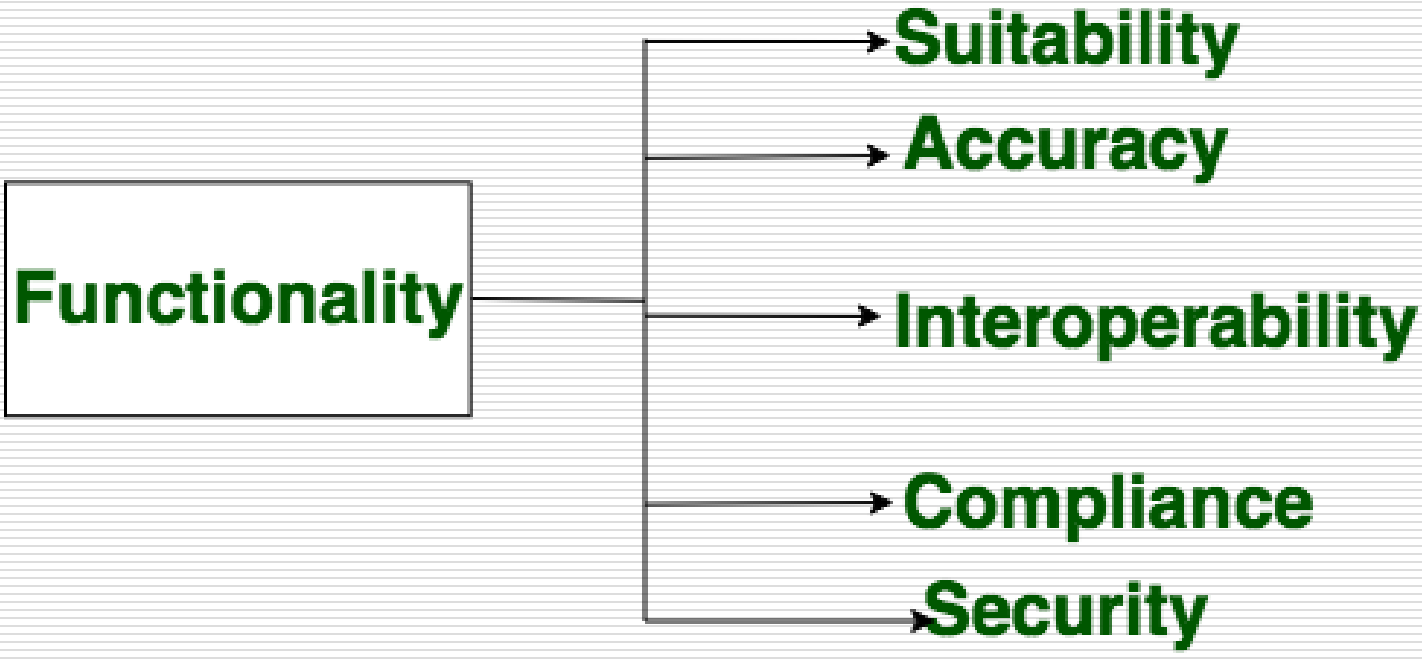
Software Components



Source: https://www.geeksforgeeks.org/software-engineering-software-characteristics/

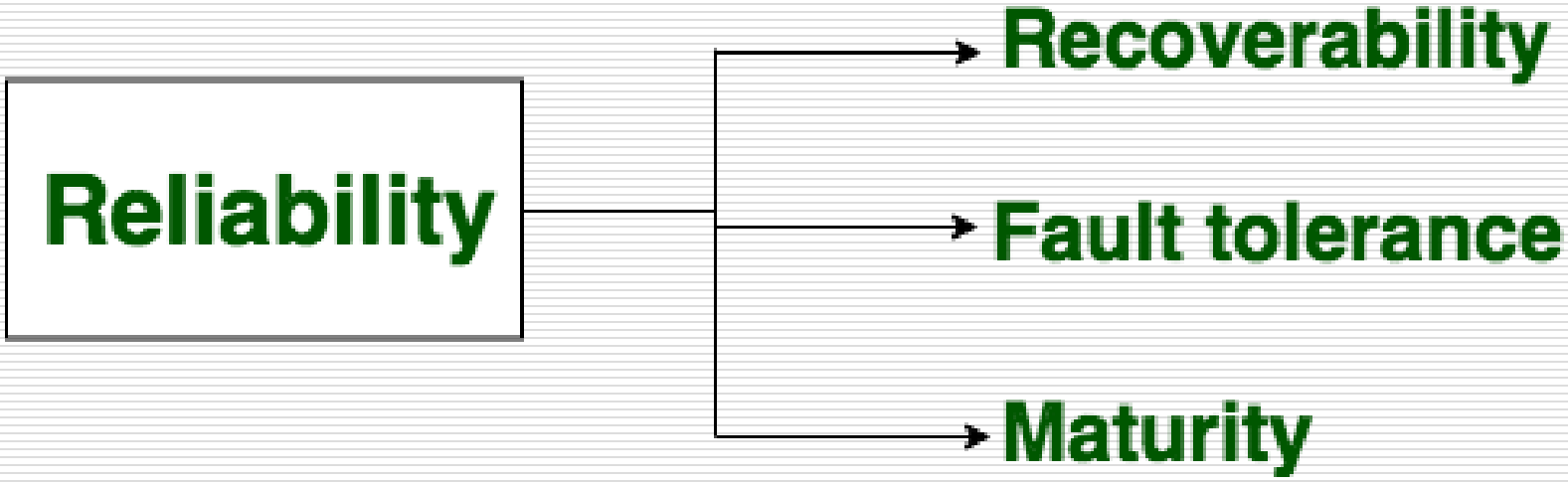
Functionality

- It refers to the degree of performance of the software against its intended purpose.



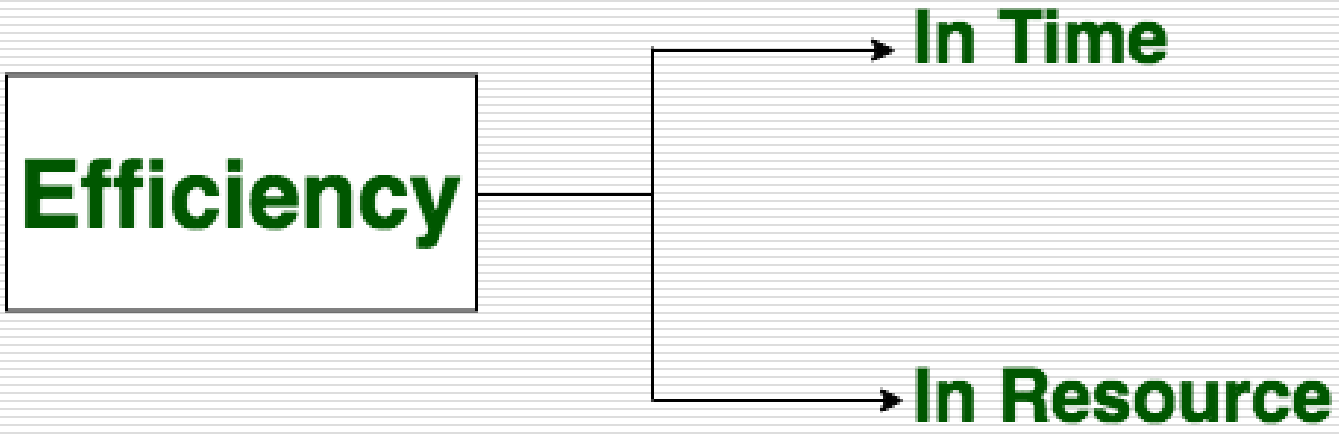
Reliability

- A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time.



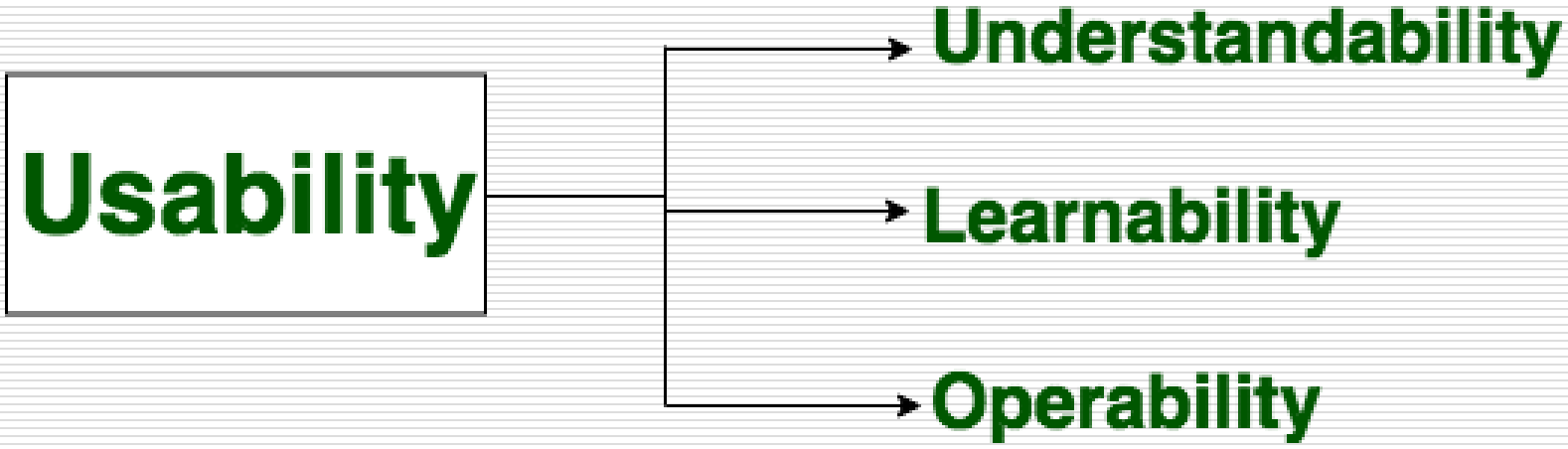
Efficiency

- It refers to the ability of the software to use system resources in the most effective and efficient manner. The software should make effective use of storage space and executive command as per desired timing requirements.



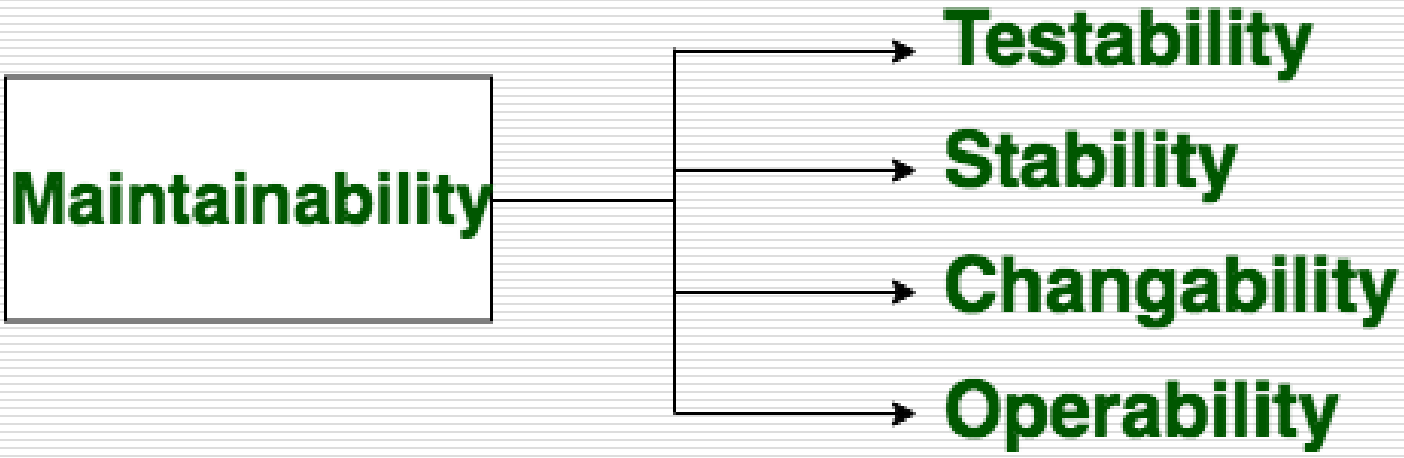
Usability

- It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software.



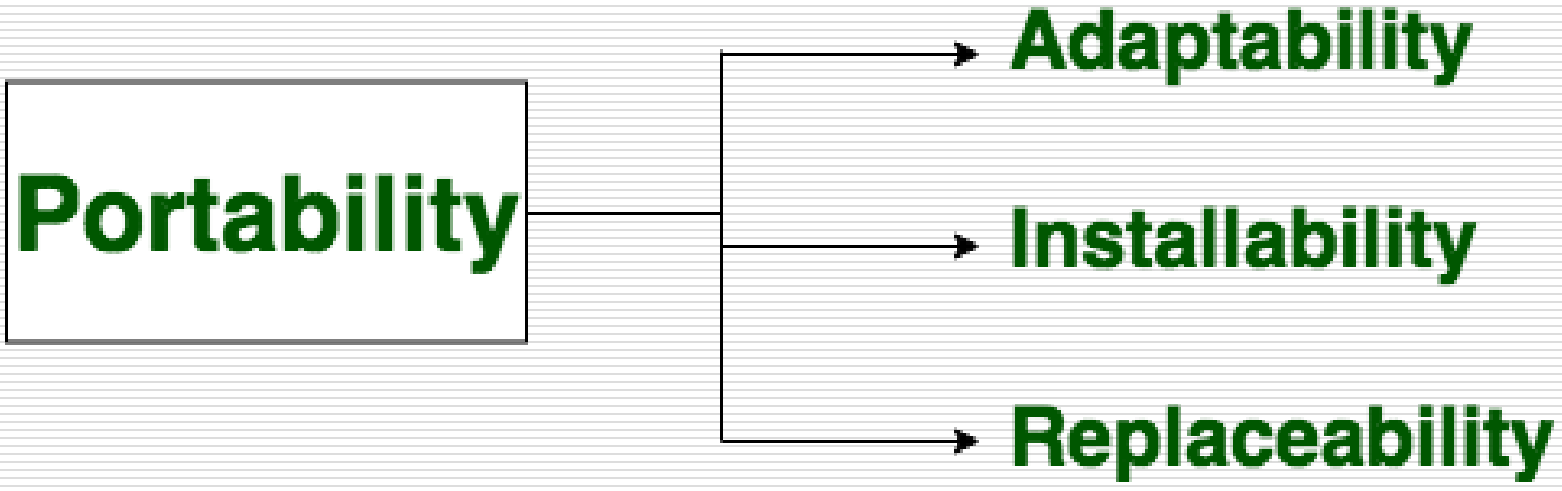
Maintainability

- It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.



Portability

- A set of attributes that bears on the ability of software to be transferred from one environment to another, without or minimum changes.



Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product line software
- Web applications
- Artificial intelligence software

System Software:

- System software is a collection of programs written to service other programs.
- It is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces.

Ex. Compilers, operating system, drivers etc.

Application Software :

- Application software consists of standalone programs that solve a specific business need.
- Application software is used to control the business function in real-time.

Engineering /Scientific software:

- Characterized by "number crunching" algorithms.
- Applications range from astronomy to volcano logy, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

Ex. Computer Aided Design (CAD), system stimulation etc.

Embedded Software:

- It resides in read-only memory and is used to control products and systems
- Embedded software can perform limited and esoteric functions.

Ex. keypad control for a microwave oven.

Product line software:

- Designed to provide a specific capability for use by many different customers, product line software can focus on a limited and esoteric marketplace.

Ex. Word processing, spreadsheet, CG, multimedia, etc.

Web Applications:

- Web apps can be little more than a set of linked hypertext files.
- It evolving into sophisticated computing environments that not only provide standalone features, functions but also integrated with corporate database and business applications.

Artificial Intelligence software

- AI software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis

Ex. Robotics, expert system, game playing, etc.

Software Crisis Problem

- Increasing cost of Computers
- Increasing product complexity
- Lack of programmers
- Slow programmer's productivity growth
- Lack of funding for software engineering research
- Rising demand for software
- Lack of caffeine in software development organizations

Software Crisis Problem

- Increasing cost of Computers
- **Increasing product complexity**
- Lack of programmers
- **Slow programmer's productivity growth**
- Lack of funding for software engineering research
- **Rising demand for software**
- Lack of caffeine in software development organizations

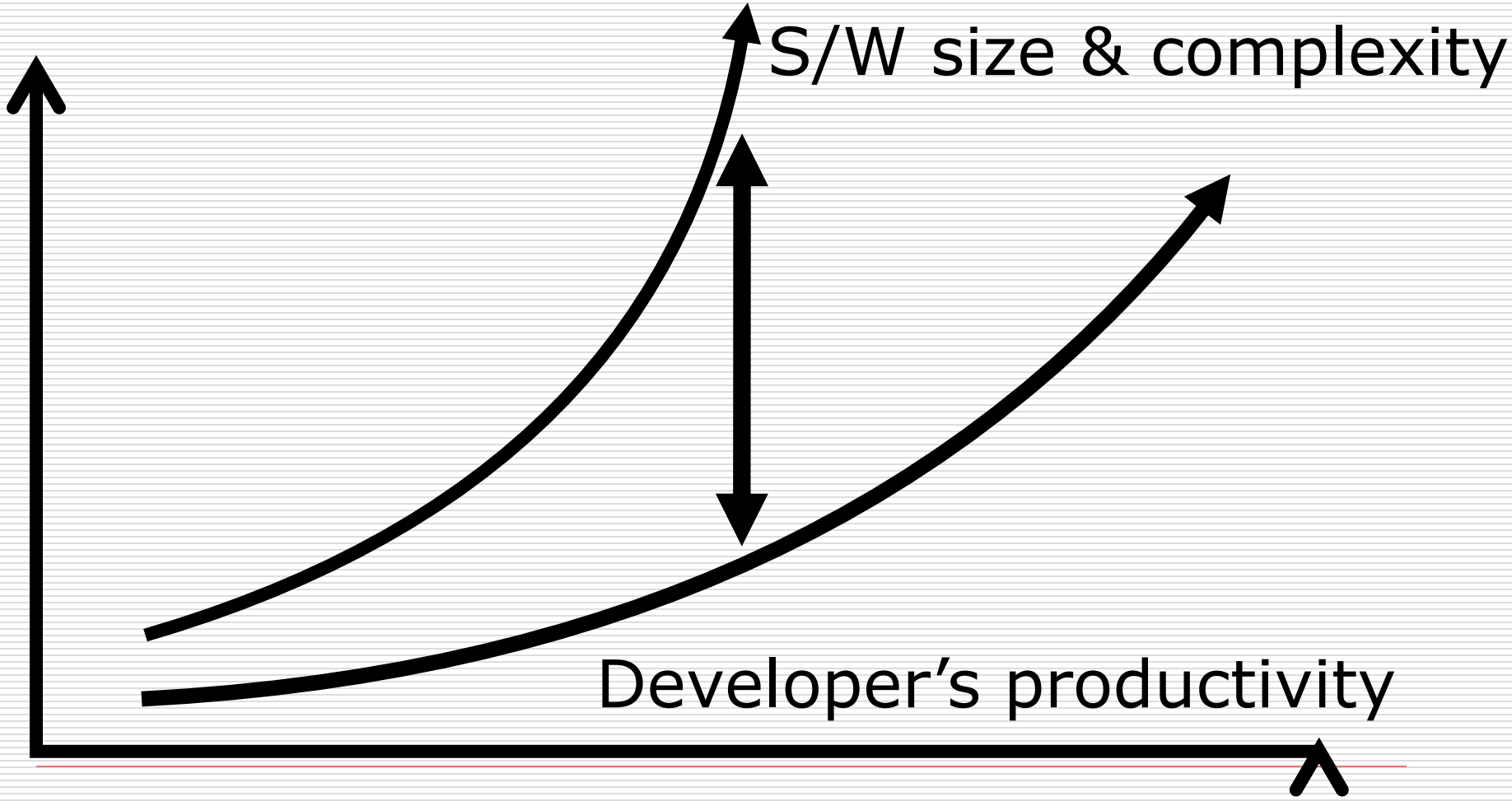
Development Effort

Size (LOC)	Example
10^2	Class exercise
10^3	Small Project
10^4	Term Project
10^5	Word processor
10^6	Operating System
10^7	Distributed System

Programming effort

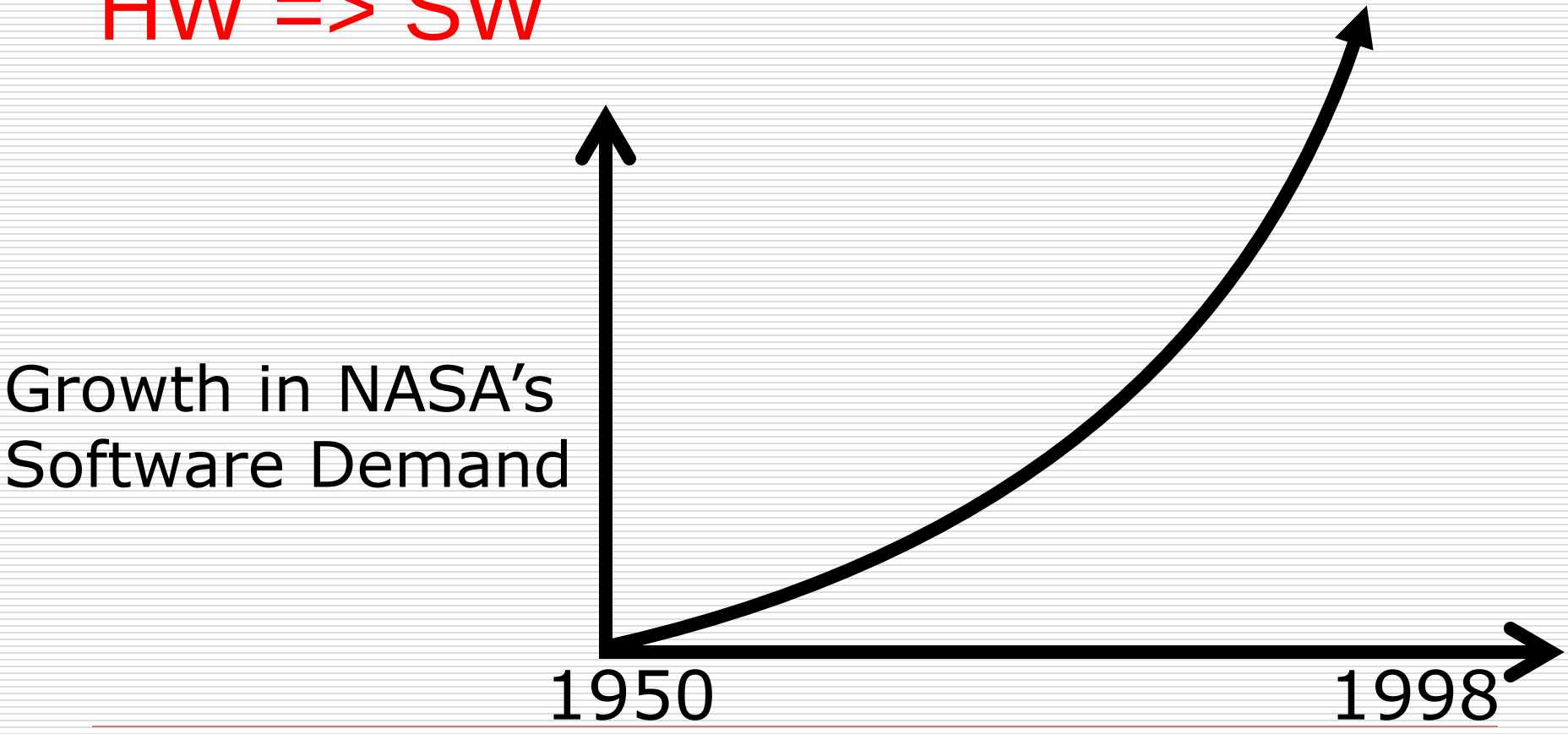
Software Engineering effort

Developers Productivity Growth



Rising Demand for Software

HW => SW



Software Evolution

- The Law of Continuing Change (1974): E-type (Real world implemented) systems must be continually adapted else they become progressively less satisfactory.
- The Law of Increasing Complexity (1974): As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.
- The Law of Self Regulation (1974): The E-type system evolution process is self-regulating with distribution of product and process measures close to normal.
- The Law of Conservation of Organizational Stability (1980): The average effective global activity rate in an evolving E-type system is invariant over product lifetime.

Software Evolution

- The Law of Conservation of Familiarity (1980): As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behavior to achieve satisfactory evolution.
- The Law of Continuing Growth (1980): The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.
- The Law of Declining Quality (1996): The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
- The Feedback System Law (1996): E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

Software Myths

Definition: Beliefs about software and the process used to build it. Myths have number of attributes that have made them insidious (i.e. dangerous).

- Misleading Attitudes - caused serious problem for managers and technical people.

Management myths

Managers in most disciplines, are often under pressure to maintain budgets, keep schedules on time, and improve quality.

Myth1: We already have a book that's full of standards and procedures for building

software, won't that provide my people with everything they need to know?

Reality :

- Are software practitioners aware of existence standards?
- Does it reflect modern software engineering practice?
- Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality?

Myth2*:* If we get behind schedule, we can add more programmers and catch up

Reality*:* Software development is not a mechanistic process like manufacturing. Adding people to a late software project makes it later.

- People can be added but only in a planned and well-coordinated manner

Myth3*:* If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality*:* If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsource software projects

Customer Myths

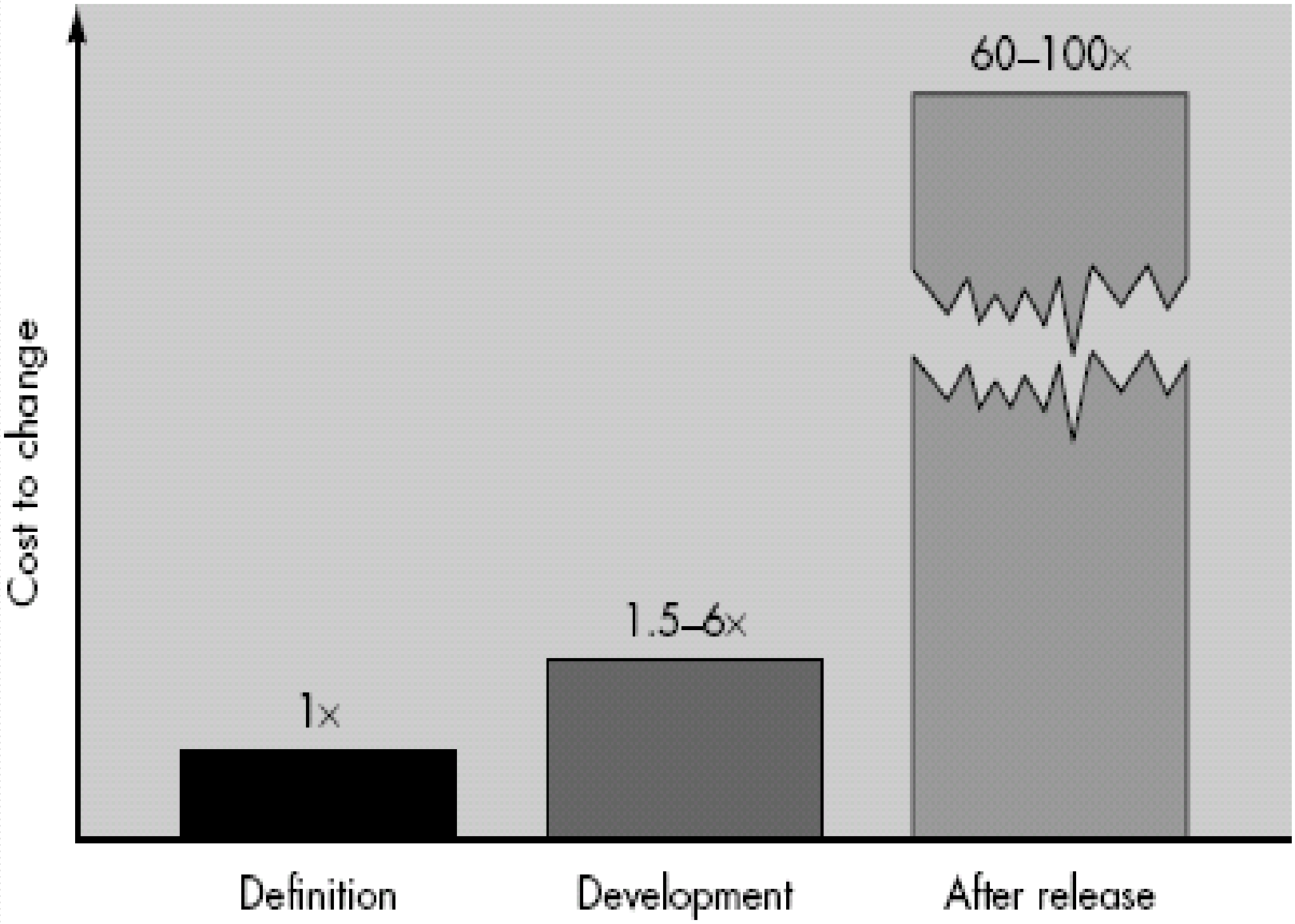
Customer may be a person from inside or outside the company that has requested software under contract.

Myth*:* A general statement of objectives is sufficient to begin writing programs— we can fill in the details later.

Reality*:* A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.

Myth*:* Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: Customer can review requirements and recommend modifications with relatively little impact on cost. When changes are requested during software design, the cost impact grows rapidly. Below mentioned *figure* for reference.


Cost to change
60–100×
1.5–6×
1×
Definition
Development
After release

Practitioner's myths

Myth1*:* Once we write the program and get it to work, our job is done.

Reality*:* Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth2*:* Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the *formal technical review.*

Myth3: The only deliverable work product for a successful project is the working program.

Reality*:* A working program is only one part of a *software configuration* that includes many elements. Documentation provides a foundation for successful engineering and, more important, guidance for software support.

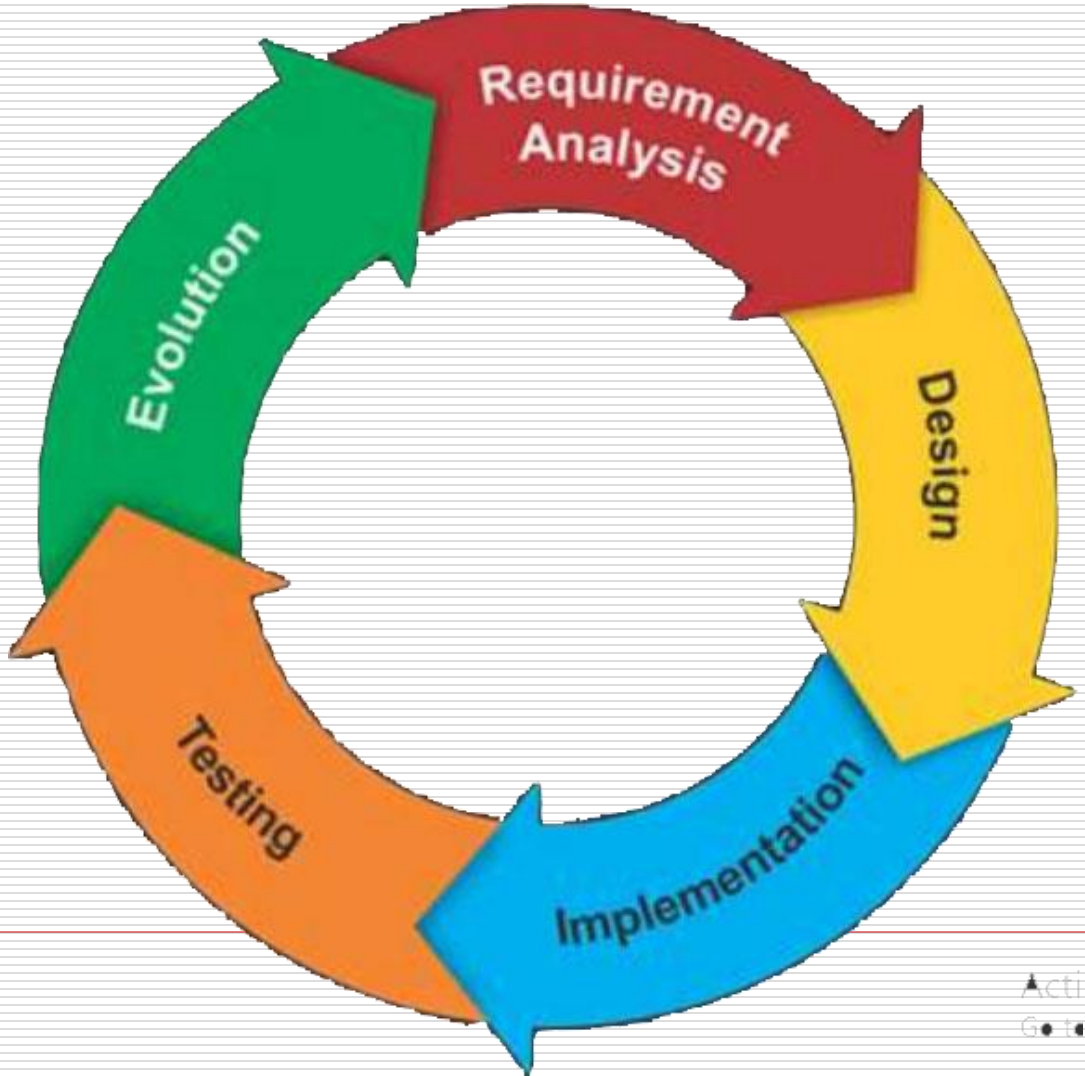
Myth4 *:* Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

Reality*:* Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Software Development Life Cycle (**SDLC**)

- The SDLC is a framework that describes the activities performed at each stage of a software development project.
- SDLC process is used by the software industry to design, develop and test high quality software. It aims to produce the quality software that meets or exceeds customer expectations, reaches completion within time and budget.

SDLC Phases


Requirement Analysis
Design
Implementation
Testing
Evolution

SDLC Phases

1. Planning and Requirements Analysis
2. Defining Requirements
3. Designing the Software
4. Building or Developing the Software
5. Testing the Software
6. Deployment and Maintenance

1. Planning & Requirement Analysis

- Requirement analysis is the most important and fundamental stage in SDLC.
- It is performed by the senior members of the team with inputs from all the stakeholders and domain experts or SMEs in the industry.
- Planning for the quality assurance requirements and identification of the risks associated with the project is also done at this stage.

Requirements Analysis (cont.)

- Business Requirements
- Stakeholder Requirements
- Solution Requirements
  - Functional Requirements
  - Non-functional Requirements
- Transition Requirements

2. Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the software requirements and get them approved from the project stakeholders.

This is done through '**SRS**' – Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

Defining Requirements (cont.)

- Enterprise Analysis
- Business Analysis Planning & Monitoring
- Elicitation
- Requirements Analysis
- Requirements Management & Communication
- Solution Assessment & Validation

3. Designing the Software

- Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.
- This DDS is reviewed by all the stakeholders and based on various parameters as risk assessment, design modularity , budget and time constraints , the best design approach is selected for the software.

4. Developing the Software

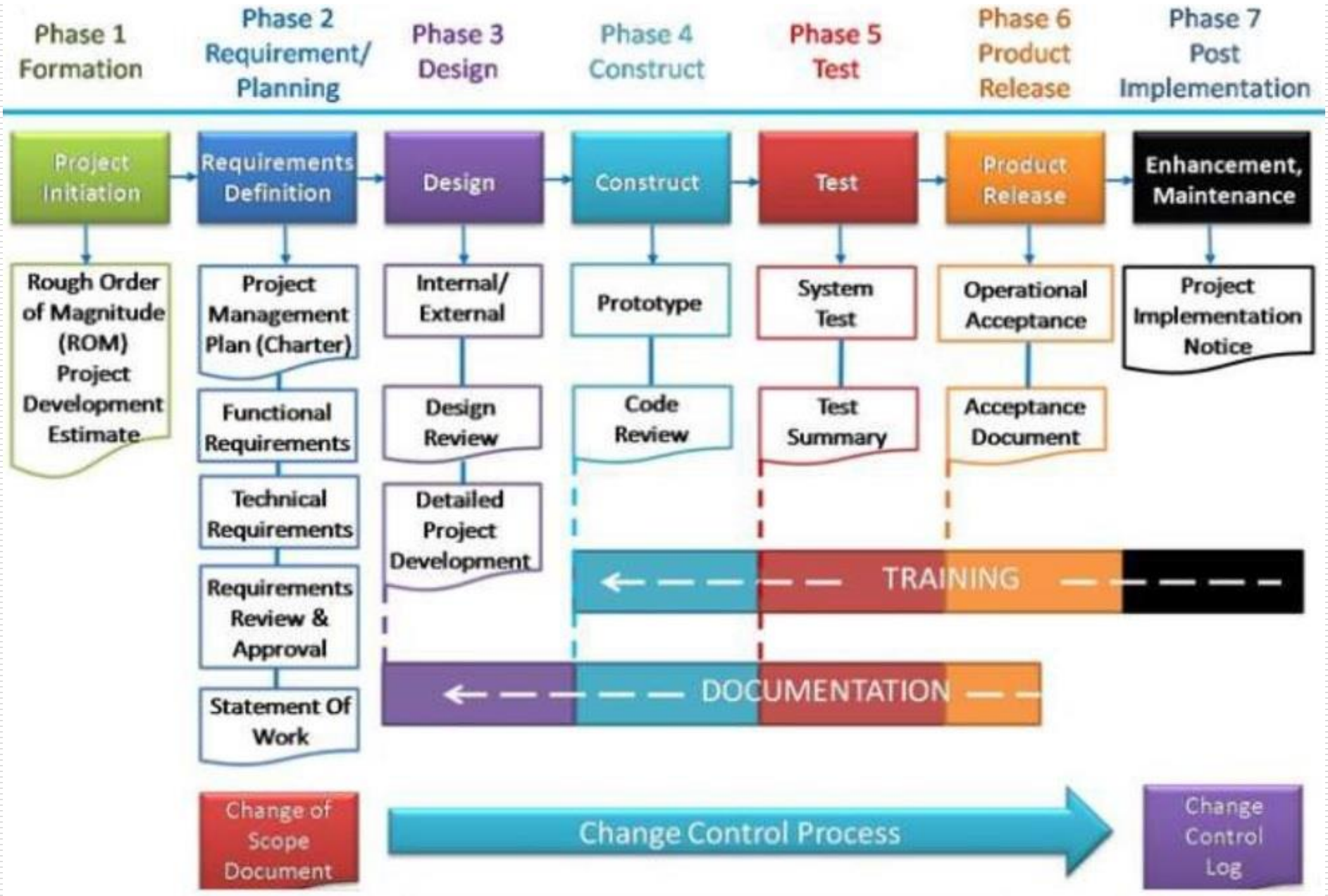
- In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage.
- Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate and implement the code.

5. Testing the Software

- This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC.
- However this stage refers to the testing only that stage of the software where defects are reported, tracked, fixed and retested, until the software reaches the quality standards defined in the SRS.

6. Deployment and Maintenance

- Once the software is tested and no bugs or errors are reported then it is deployed.
- Then based on the feedback, the software may be released as it is or with suggested enhancements in the target segment.
- After the software is deployed then its maintenance starts.


Phase 1
Formation
Phase 2
Requirement/
Planning
Phase 3
Design
Phase 4
Construct
Phase 5
Test
Phase 6
Product
Release
Phase 7
Post
Implementation
Project Initiation
Requirements Definition
Design
Construct
Test
Product Release
Enhancement, Maintenance
Rough Order of Magnitude (ROM) Project Development Estimate
Project Management Plan (Charter)
Functional Requirements
Technical Requirements
Requirements Review & Approval
Statement Of Work
Internal/ External
Design Review
Detailed Project Development
Prototype
Code Review
System Test
Test Summary
Operational Acceptance
Acceptance Document
Project Implementation Notice
TRAINING
DOCUMENTATION
Change of Scope Document
Change Control Process
Change Control Log

SDLC Stages & Documents