

Software Engineering

A thick, horizontal yellow brushstroke underline that spans the width of the slide, positioned directly beneath the title text.

What is Software Engineering?

Program vs. Software



What is Software Engineering?

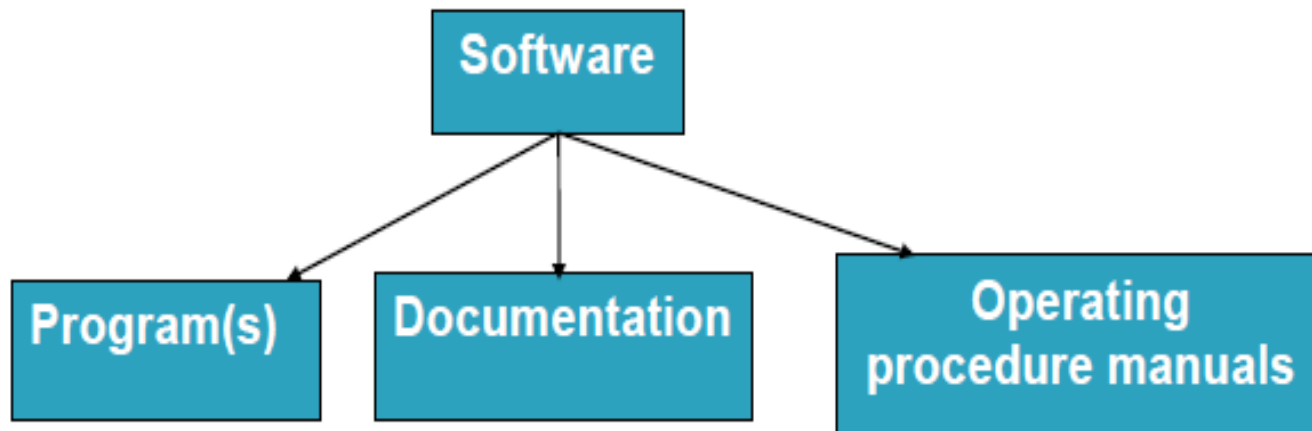
Program vs. Software

- o Program is set of instructions written for a specific purpose.
- o Software is the combination of program(s), documentation (documents produced during development) and operating procedure manuals (delivered with programs to customer at the time of release).



What is Software Engineering?

- o Program vs. Software



Object oriented concepts

- ✓ The size and complexity of software is increasing day by day. Conventional approaches of software design and implementation may not be effectively applicable.
- ✓ We want to simplify the development process and to produce high quality maintainable software.
- ✓ As we all know, development may take few years and same may have to be maintained for many years.
- ✓ A maintainable software may reduce the maintenance cost and high quality may enhance the sustainability of the software.

Object oriented concepts

- ✓ It is becoming popular to design, develop and maintain large size, complex and critical software systems using object oriented paradigm.
- ✓ Due to its popularity and acceptability in customers, companies are also releasing the object oriented versions of their existing software products.
- ✓ Many of the customers expect object oriented software solutions and request the same for their forthcoming projects.

Programs vs Software Products

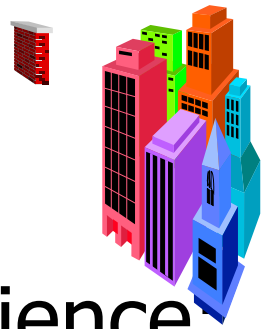
Characteristics	Program	SW product
Users	self	Others
Number of user	Self/few	Large number
Size	small	Large
Functionality	limited	Large
Interfaces	Ok	Well designed
Environment	One	Several
System	Used by itself	Works with other systems
User background	Similar	Varied
Presence of bugs	Not a major concern	Major concern
Documentation	Minimal	Exhaustive
Testing	Minimal	Exhaustive
Cost/user	High	low
Developers	One /few	Many
Use of standards, etc	Not essential	essential

Software + Engineering

- **Software** is considered to be a collection of executable programming code, associated libraries and documentations.
- Software, when made for a specific requirement is called **software product**.
- **Engineering** on the other hand, is all about developing products, using well-defined, scientific principles and methods.
- **Software engineering as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures.**
- ***The outcome of software engineering is an efficient and reliable software product.***

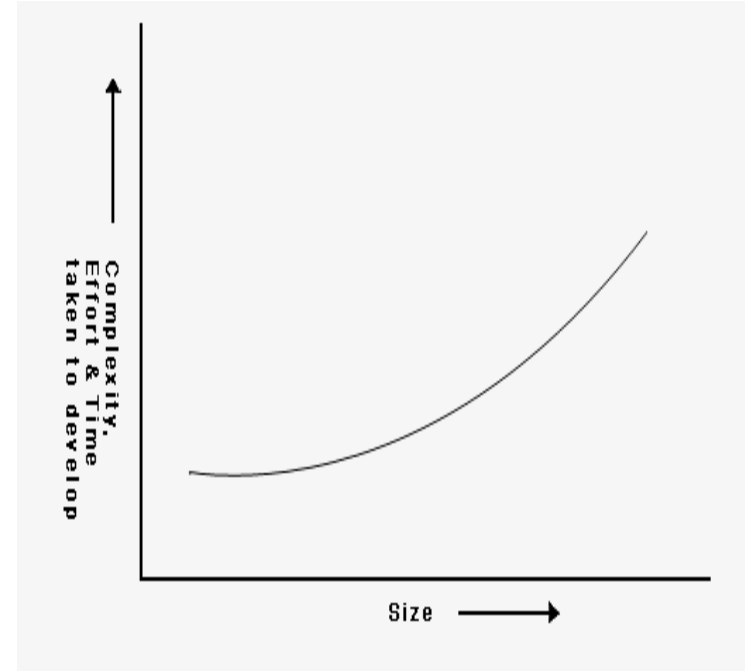
What is Software Engineering?

- Engineering approach to develop software.
 - Building Construction Analogy.
- Systematic collection of past experience:
 - techniques,
 - methodologies,
 - guidelines.




Scope and necessity of software engineering

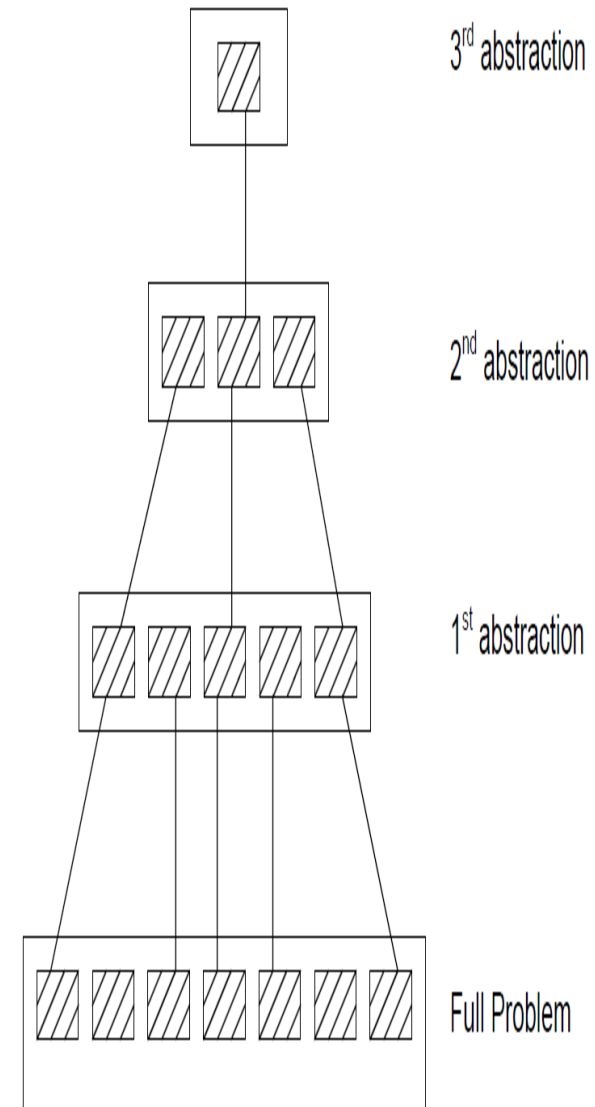
- Ex- **Difference between building a Wall and a Multistoried Building.**
- In industry it is usually needed to develop large programs to accommodate multiple functions.
- Problem with developing such large commercial programs is that the **complexity and difficulty levels of the programs increase exponentially with their sizes**



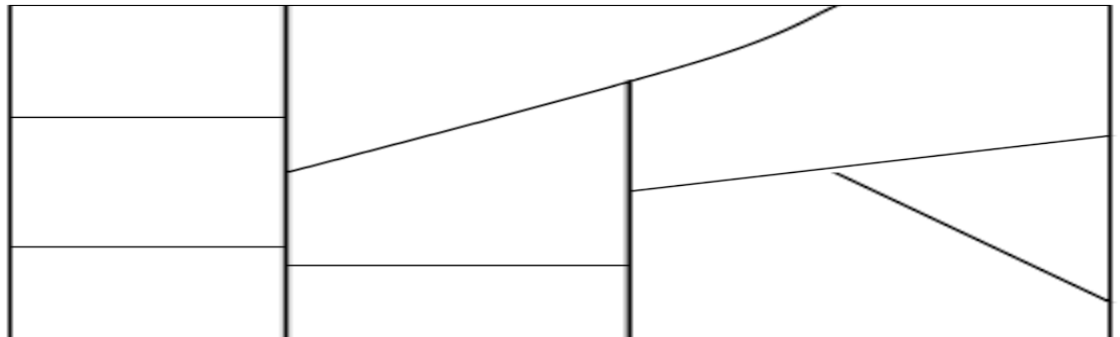
Increase in development time and effort with problem size

- For example, a program of size 1,000 lines of code has some complexity.
- 
- But a program with 10,000 LOC is not just 10 times more difficult to develop, but may as well turn out to be 100 times more difficult unless software engineering principles are used.
 - In such situations software engineering techniques come to rescue to reduce the programming complexity.
 - Software engineering principles use two important techniques to reduce problem complexity: **abstraction and decomposition.**

- The principle of abstraction implies that a **problem can be simplified by omitting irrelevant details.**
- Consider only those aspects of the problem that are relevant for certain purpose and suppress other aspects that are not relevant for the given purpose.
- Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on.
- Abstraction is a powerful way of reducing the complexity of the problem.



- The other approach to tackle problem complexity is **decomposition**. A complex problem is divided into several smaller problems and then the smaller problems are solved one by one.
- However, in this technique any random decomposition of a problem into smaller parts will not help.
- The problem has to be decomposed such that each component of the **decomposed problem can be solved independently** and then the solution of the different components can be combined to get the full solution.
- **Challenge:** A good decomposition of a problem as should minimize interactions among various components.



NEED OF SOFTWARE ENGINEERING

Because of higher rate of change in user requirements and environment on which the software is working.

Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

Cost- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.

NEED OF SOFTWARE ENGINEERING

Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which the user works.

If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

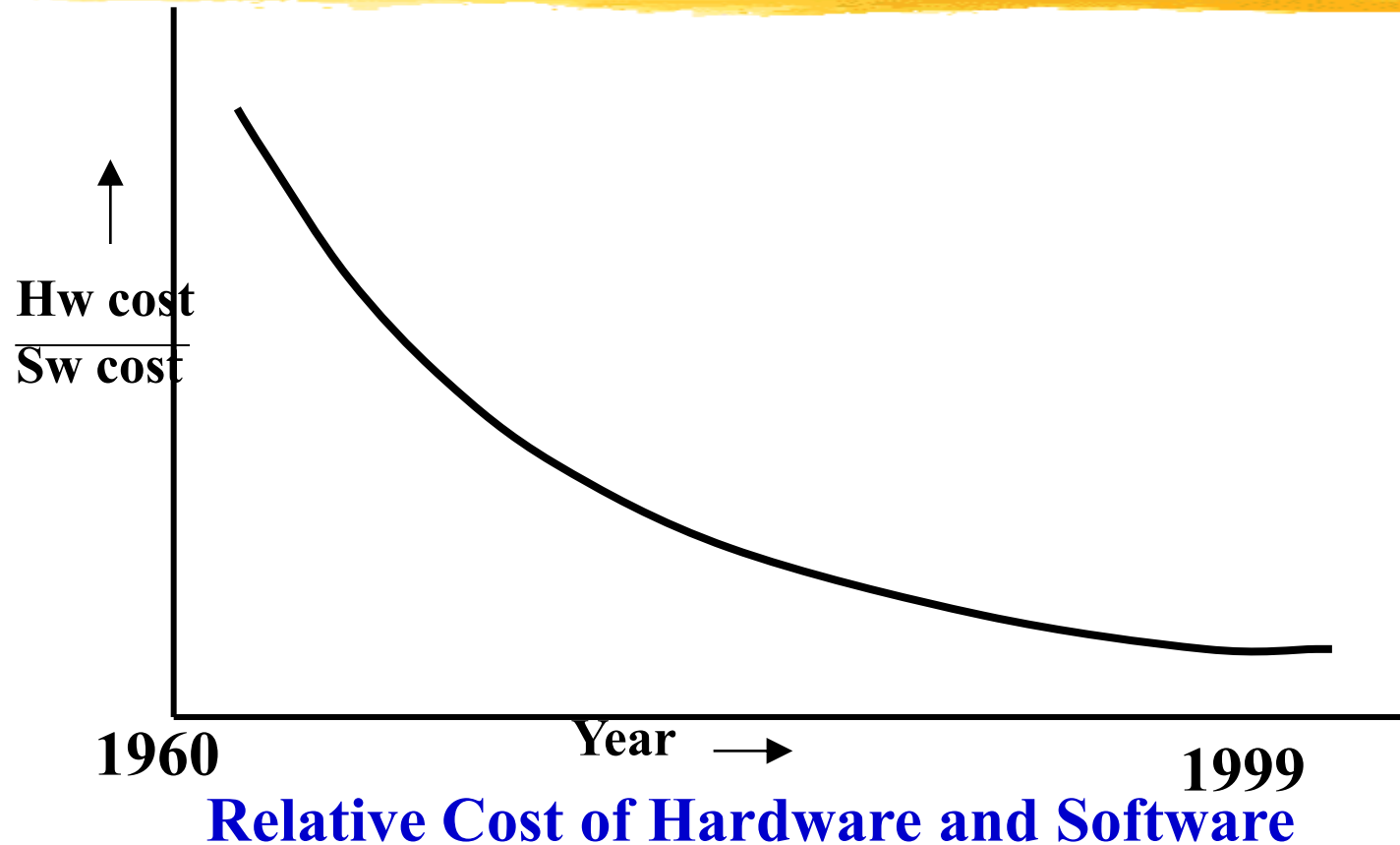
Quality Management- Better process of software development provides better and quality software product.

Software Crisis



- Software products:
 - fail to meet user requirements.
 - frequently crash.
 - expensive.
 - difficult to alter, debug, and enhance.
 - often delivered late.
 - use resources non-optimally.

Software Crisis (cont.)



Factors contributing to the software crisis



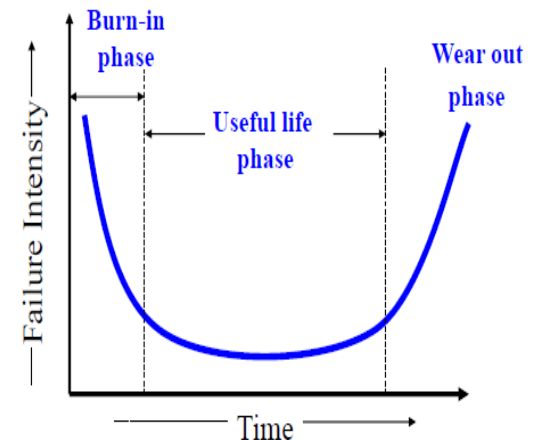
- Larger problems,
- Lack of adequate training in software engineering,
- Increasing skill shortage,
- Low productivity improvements.

Software Vs Hardware

- Depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects);
- Defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time.
- As time passes, the failure rate rises again as hardware components suffer from the cumulative affects of dust, vibration, temperature extremes, and many other environmental conditions. Hardware begins to wear out.

Software Characteristics

✓ Software does not wear out.

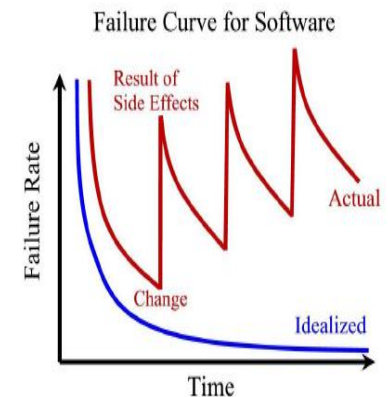


Software doesn't "wear out."

- Software is not influenced to the environmental conditions that cause hardware to wear out.
- The failure rate curve for software should take the form of the "idealized curve". Undiscovered defects will cause high failure rates early in the life of a program.
- However, these are corrected (ideally, without introducing other errors) and the curve flattens as shown.
- As changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike.
- Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

Software Characteristics

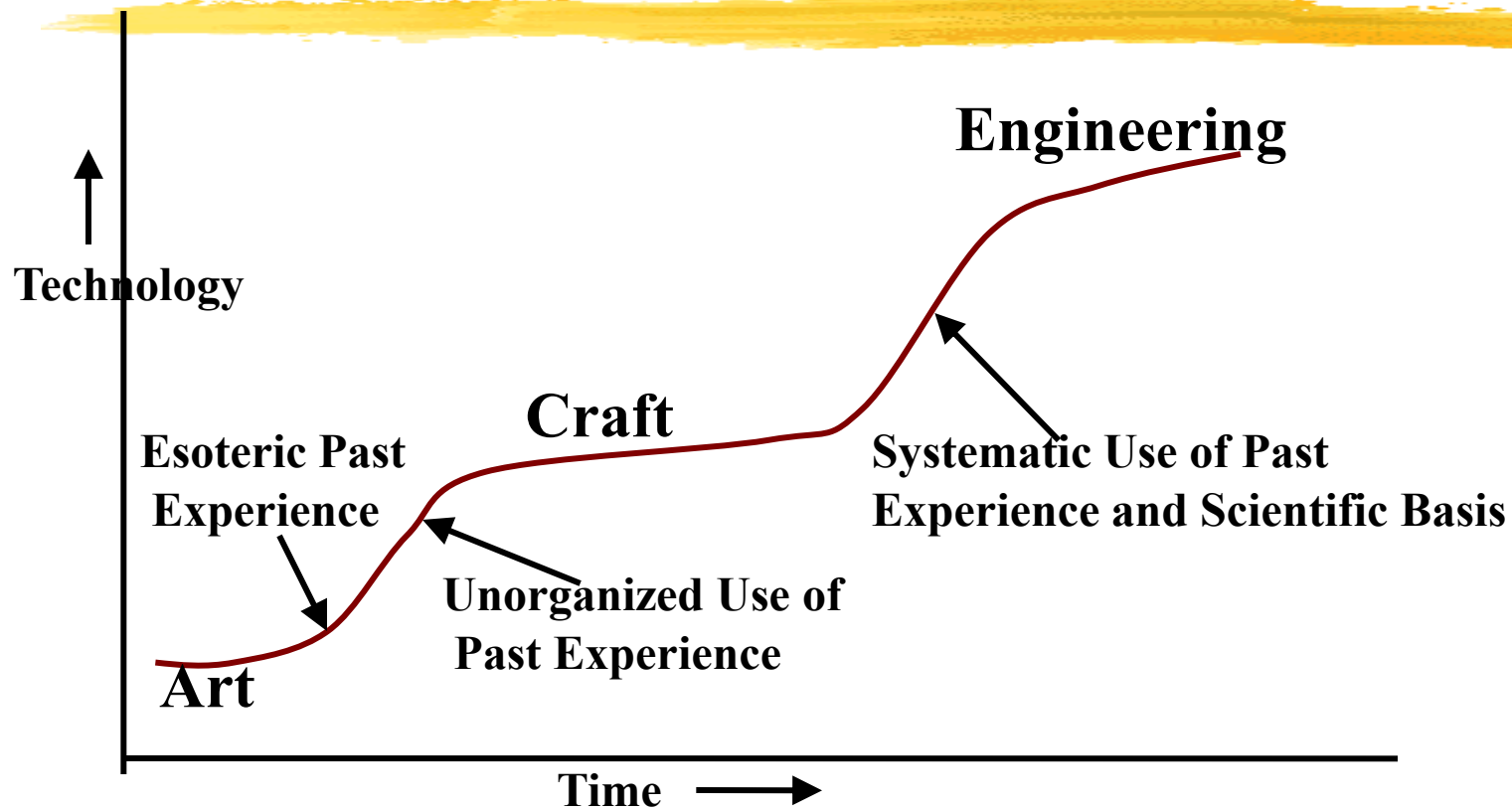
- ✓ Flexibility of Software
- ✓ Reusability of Software



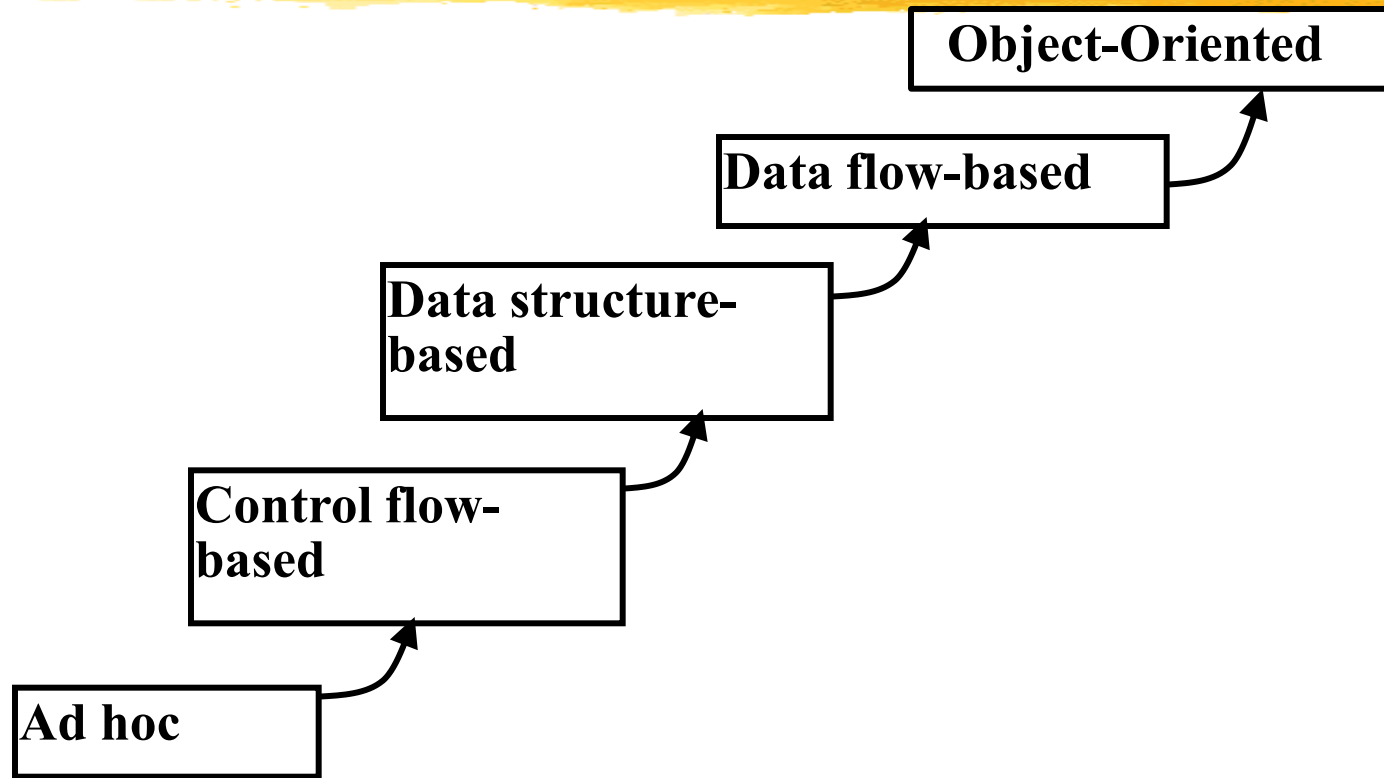


Evolution of Technology

Technology Development Pattern



Evolution of Design Techniques



Evolution of software design techniques over the last 50 years.

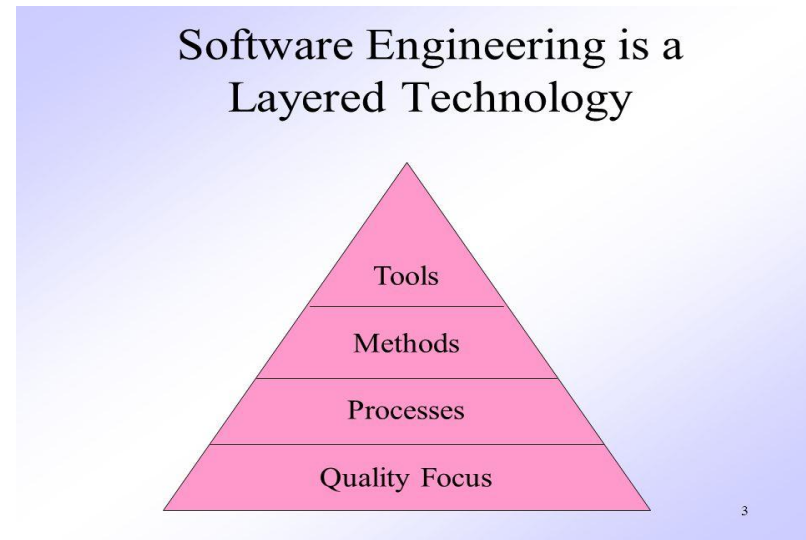
- During the 1950s, most programs were being written in assembly language.
- These programs were limited to about a few hundreds of lines of assembly code, i.e. were very small in size.
- **Every programmer developed programs in his own individual style - based on his intuition.**
- This type of programming was called **Exploratory Programming.**

Exploratory style vs. modern style of software development.

Exploratory style	Modern style
Emphasis on error correction	Emphasis on error prevention
Coding was considered synonymous with software development.	Coding is regarded as only a small part of the overall software development activities.
Believed in developing a working system as quickly as possible and then successively modifying it until it performed satisfactorily.	There are several development activities such as design and testing which typically require much more effort than coding.

Software Engineering as Layered Technology

- Software engineering is totally a layered technology.
- It means that, to develop a software one will have to go from one layer to another layer.
- The all layers are related and each layer fulfill the all requirements of the previous layer.



- **Quality layer:** An engineering approach must rest on a quality. A product should meet its specification. The "**Bed Rock**" that supports software engineering is a quality focus.
- The **process** is the key that **keeps all the layers together**. It is the glue that holds the technology together and enables rational and timely development of computer software.
- The **Method** is an answer to all the **HOW'S that are asked during the process**. It is collection of all tasks starting from the **requirement analysis, design, construction, support and testing phase**.
- **Tools** provide automated or semi-automated supports for the process and the methods. When the tools are integrated so that the information created by one tool can be used by another, eg. computer-aided software engineering (CASE). CASE combine software, hardware, and software engineering database.

What is a Software Process

- A set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals)
- As an organization matures, the software process becomes better defined and more consistently implemented throughout the organization
- Software process maturity is the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective



Software Life Cycle

- Software life cycle (or software process):
 - series of identifiable stages that a software product undergoes during its life time:
 - * Feasibility study
 - * requirements analysis and specification,
 - * design,
 - * coding,
 - * testing
 - * maintenance.

Life Cycle Model

- A software life cycle model (or process model):
 - a descriptive and diagrammatic model of software life cycle:
 - identifies all the activities required for product development,
 - establishes a precedence ordering among the different activities,
 - Divides life cycle into phases.

Why Model Life Cycle ?

- A written description:
 - forms a common understanding of activities among the software developers.
 - helps in identifying inconsistencies, redundancies, and omissions in the development process.
 - Helps in tailoring a process model for specific projects.

Life Cycle Model (CONT.)



- The development team must identify a suitable life cycle model:
 - and then adhere to it.
 - Primary advantage of adhering to a life cycle model:
 - * helps development of software in a systematic and disciplined manner.

Life Cycle Model (CONT.)



- When a program is developed by a single programmer
 - he has the freedom to decide his exact steps.
- When a software product is being developed by a team:
 - there must be a precise understanding among team members as to when to do what,
 - otherwise it would lead to chaos and project failure.

Life Cycle Model (CONT.)

- A life cycle model:
 - defines entry and exit criteria for every phase.
 - A phase is considered to be complete:
 - * only when all its exit criteria are satisfied.
- The phase exit criteria for the software requirements specification phase:
 - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.
- A phase can start:
 - only if its phase-entry criteria have been satisfied.

Life Cycle Model (CONT.)



- Many life cycle models have been proposed.
- We will confine our attention to a few important and commonly used models.
 - classical waterfall model
 - iterative waterfall,
 - evolutionary,
 - prototyping, and
 - spiral model

CHARACTERESTICS OF GOOD SOFTWARE



- A software product can be judged by what it offers and how well it can be used.
- This software must satisfy on the following grounds:
 - **Operational**
 - **Transitional**
 - **Maintenance**

Operational

- This tells us how well software works in operations. It can be measured on:
 - **Budget** : in terms of cost, manpower
 - **Usability** : ease of use
 - **Efficiency** : minimum expenditure of time and effort
 - **Correctness** : with respect to a specification
 - **Functionality** : having a practical use
 - **Dependability** : extent to which a critical system is trusted by its users
 - **Security** : degree of resistance or protection from

Transitional

- This aspect is important when the software is moved from one platform to another:
 - **Portability** : usability of the same software in different environments
 - **Interoperability** : ability of a system or a product to work with other systems
 - **Reusability** : use of existing assets in some form within the software product development process
 - **Adaptability** : able to change or be changed in order to fit or work better in some situation

Maintenance

- This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:
 - **Modularity:** degree to which a system's components may be separated and recombined
 - **Maintainability:** degree to which an application is understood, repaired, or enhanced
 - **Flexibility:** ability for the solution to adapt to possible or future changes in its requirements
 - **Scalability:** ability of a program to scale

Summary



- Software engineering is:
 - systematic collection of decades of programming experience
 - together with the innovations made by researchers.

Summary



- A fundamental necessity while developing any large software product:
 - adoption of a life cycle model.

Summary

- Adherence to a software life cycle model:
 - helps to do various development activities in a systematic and disciplined manner.
 - also makes it easier to manage a software development effort.

Reference



- R. Mall, "Fundamentals of Software Engineering," Prentice-Hall of India, 1999, CHAPTER 1.