Software Engineering

Module-1: Introduction to Software Engineering

Course Content

- Introduction to SE concepts, Software life cycle models, Software project management, Requirements analysis and specification, Software design, Function-oriented software design, Object modelling using UML
- Object-oriented software development, User interface design, Coding and Testing

Text Books

1. Pankaj Jalote. "An Integrated Approach to Software Engineering", 3rd Edition, Narosa, 2005
2. Rajiv Mall, Fundamentals of Software Engineering, 2014
3. Software Engineering: A Practitioner's Approach by Pressman, 7th edition.
4. Pfleeger Shari Lawrence, Software Engineering Theory and Practices, 4th edition.
5. Bernd Bruegge, Allen Dutoit: "Object-Oriented Software Engineering: Using UML, Patterns, and Java", Prentice Hall, 2003.
6. Blaha and Rumbaugh. "Object-Oriented Analysis and Modeling using UML, 2nd Edition, TMH 2005.

What is Software?

- **Software**, in its most general sense, is a set of instructions or programs instructing a computer to do specific tasks.
- Software is a **generic term** used to describe computer programs. Scripts, applications, programs and a set of instructions are all terms often used to describe software.
- The theory of software was first proposed by **Alan Turing** in 1935 in his essay "Computable numbers with an application to the Entscheidungs problem."
- However, the word software was coined by mathematician and statistician **John Tukey** in a 1958 issue of American Mathematical Monthly in which he discussed electronic calculators' programs.

What is Software Engineering?

- Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures.
- The outcome of software engineering is an efficient and reliable software product.
- Definitions
- IEEE defines software engineering as:
  - *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*
  - *The study of approaches as in the above statement.*

What is Software Engineering?

- **Fritz Bauer**, a German computer scientist, defines software engineering as:
  - *Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.*
- The discipline of software engineering was created to address poor quality of software, get projects exceeding time and budget under control, and ensure that software is built systematically, rigorously, measurably, on time, on budget, and within specification.
- Engineering already addresses all these issues, hence the same principles used in engineering can be applied to software.
- The widespread lack of best practices for software at the time was perceived as a "software crisis".

What is Software Engineering?

- **Who coined the term?**
- The earliest origin of the term "software engineering" was actually by Margaret Hamilton in the early 1960s while she was working on the Apollo space mission at Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for the Apollo space program.

Hamilton in 1969, standing next to listings of the software she and her MIT team produced for the Apollo project.

A talk of her at ICSE-2018 conference https://www.youtube.com/watch?v=ZbVOF0Uk5lU

Program Vs Product

- Many toy software are being developed by individuals such as students for their classroom assignments and hobbyists for their personal use.
- These are usually small in size and support limited functionalities and the author of a program is usually the sole user of the software.
- These toy software therefore usually lack good user-interface and proper documentation.
- Besides these may have poor maintainability, efficiency, and reliability.
- Since these toy software do not have any supporting documents such as users' manual, maintenance manual, design document, test documents, etc., we call these toy software as ***programs***.

Program Vs Product

- Professional software usually have multiple users and, have good user-interface, proper users' manuals, and good documentation support.
- Since, a software product has a large number of users, it is systematically designed, carefully implemented, and thoroughly tested.
- Professional software are often too large and complex to be developed by any single individual.
- It is usually developed by a group of developers working in a team.
- When developing small programs for personal use, rigid adherence to software engineering principles is often not worthwhile.

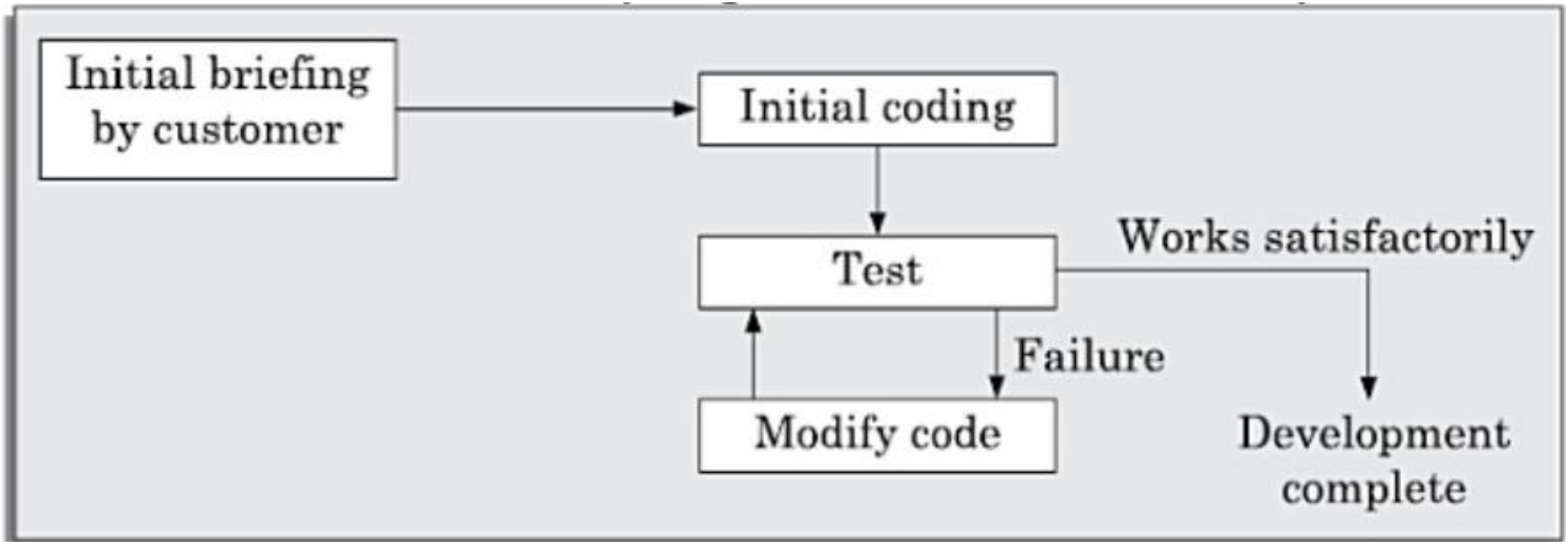
Programs versus Software Products

• Usually small in size	• Large
• Author himself is sole user	• Large number of users • Team of developers
• Single developer	• Well-designed interface
• Lacks proper user interface	• Well documented & user-manual prepared
• Lacks proper documentation	• Systematic development
• Ad hoc development.	

Evolution of an Art into an Software Engineering Discipline

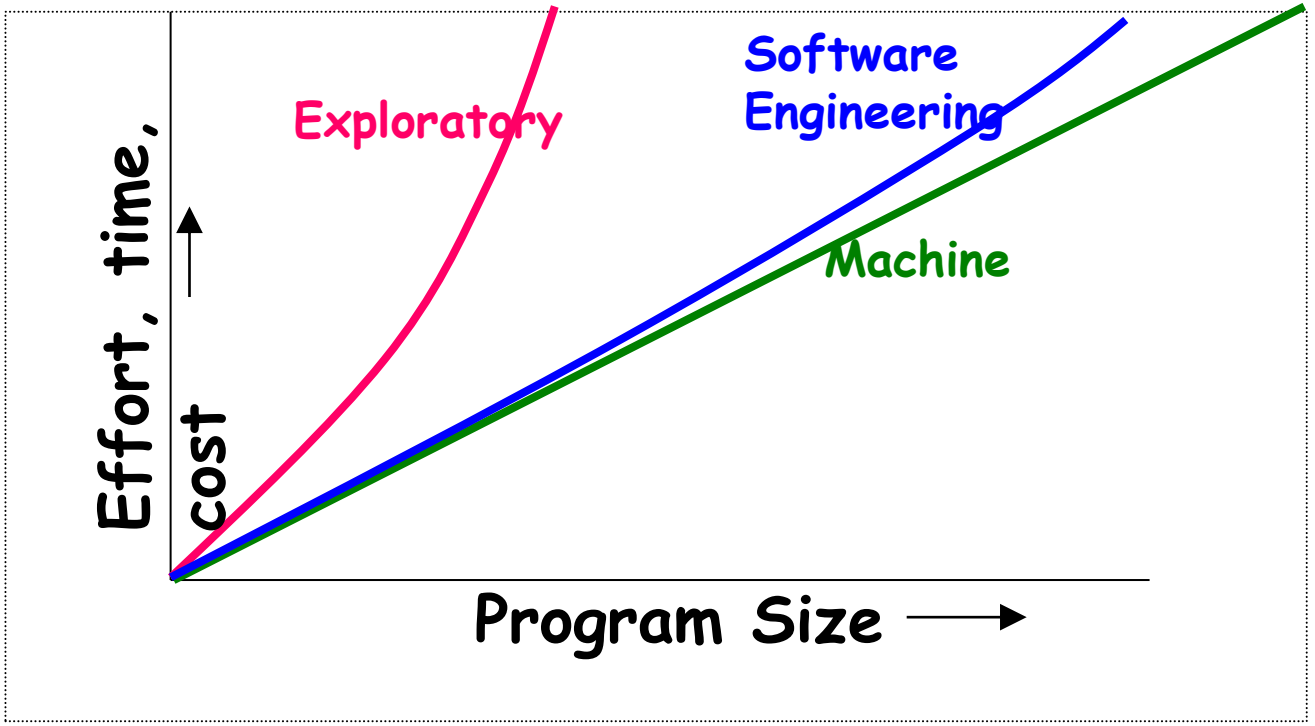
- The early programmers used an exploratory (also called build and fix) style.
  - In the build and fix (exploratory) style, normally a `dirty' program is quickly developed.
  - The different imperfections that are subsequently noticed are fixed.

Exploratory program development



What is Wrong with the Exploratory Style?

- Can successfully be used for very small programs only.



What is Wrong with the Exploratory Style?

- Besides the exponential growth of effort, cost, and time with problem size:
  - Exploratory style usually results in unmaintainable code.
  - It becomes very difficult to use the exploratory style in a team development environment.
  - Foremost difficulty is the exponential growth of development time and effort with problem size and large-sized software.

Principles Deployed by Software Engineering to Overcome Human Cognitive Limitations

- Mainly two important principles are deployed:
  - Abstraction
    - Simplify a problem by omitting unnecessary details.
    - Focus attention on only one aspect of the problem and ignore irrelevant details.
  - Decomposition
    - Decompose a problem into many small independent parts.
    - The small parts are then taken up one by one and solved separately.

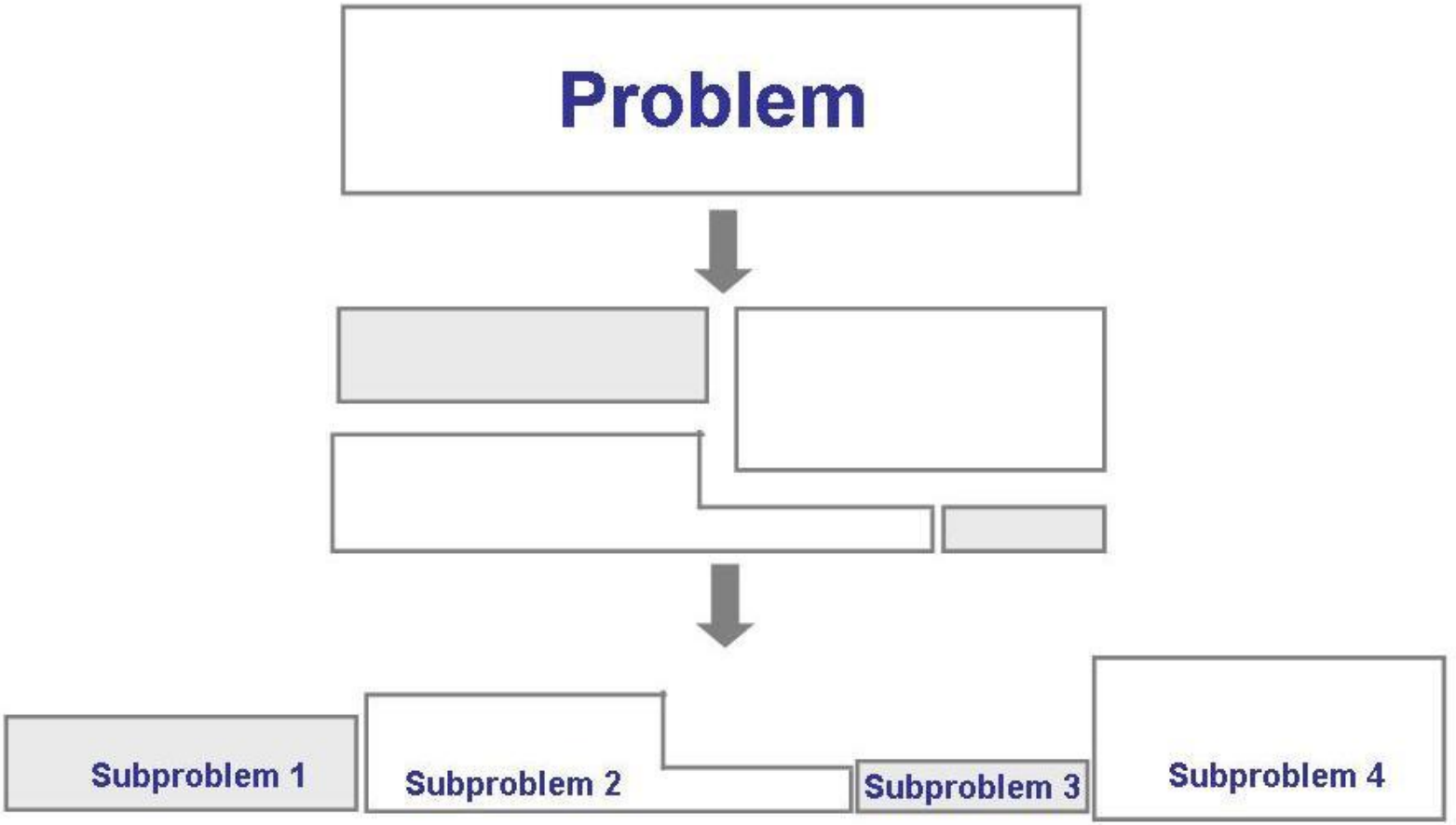
What is Software Engineering

- Software products are large and complex
- Development requires analysis and synthesis

- ***Analysis***: decompose a large problem into smaller, understandable pieces
  - abstraction is the key
- **Synthesis**: build (compose) a software from smaller building blocks
  - composition is challenging

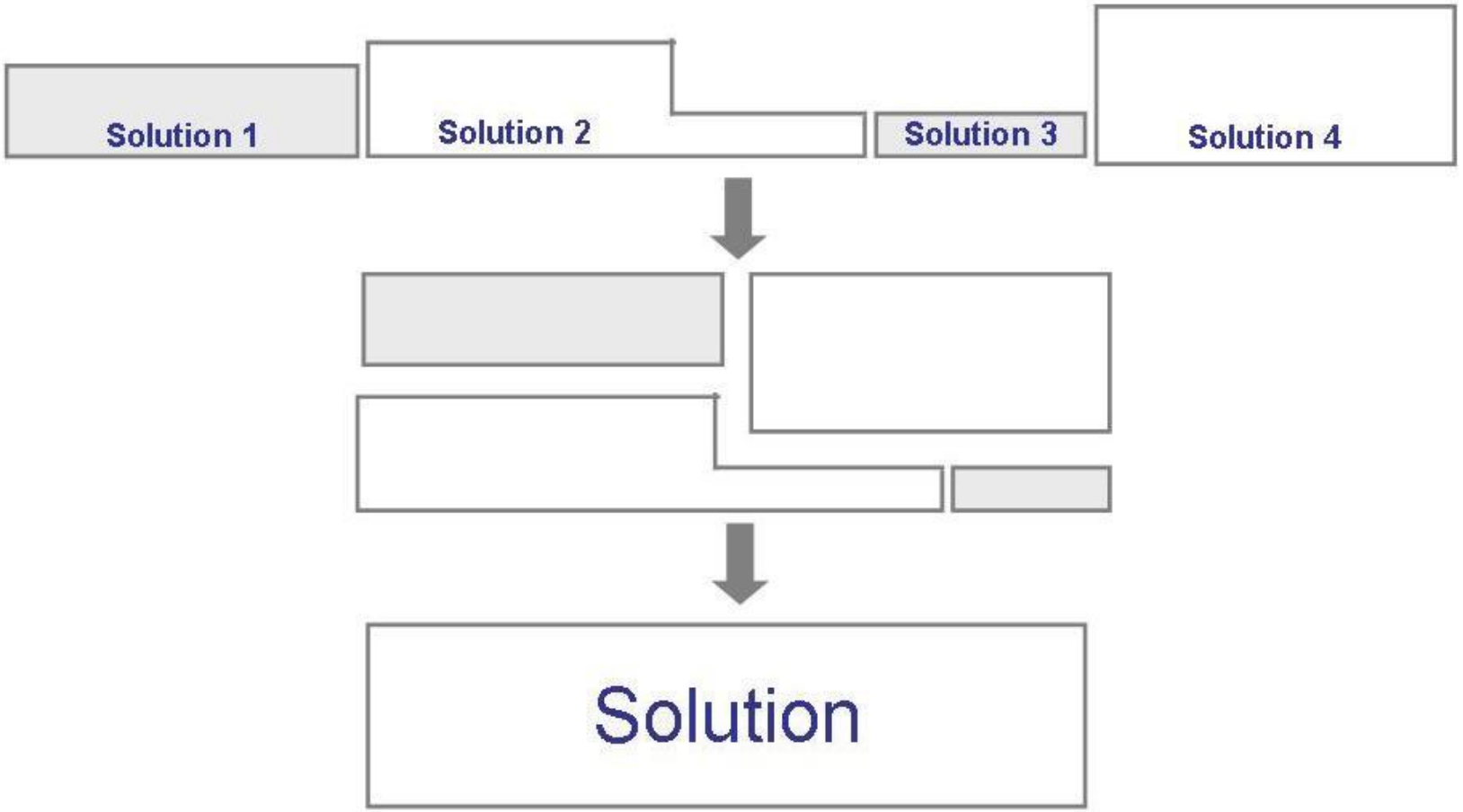
What is Software Engineering

- The analysis process



What is Software Engineering

- The synthesis process



Why Study Software Engineering?

- To acquire skills to develop large programs.
  - Exponential growth in complexity and difficulty level with size.
  - The ad hoc approach breaks down when size of software increases.
  - In software engineering we are not dealing with programs that people build to illustrate something or for hobby.

Why Study Software Engineering?

- Instead we refer to this software as ***industrial strength software***.
- An *industrial strength software system* is built to solve some problem of a client.
- It is used by the clients organization for operating some part of business.
- Ex. To manage inventories, finances, monitor patients, air traffic control, etc.)
- A malfunction of such a system can have huge impact in terms of financial or business loss, inconvenience to users, or loss of property
  and life.
- The software system needs to be of high quality with respect to properties like dependability, reliability, user-friendliness, etc.

Software Failure Cases

- **Faulty Soviet early warning system nearly causes WWIII (1983) :** The Russians' system told them that the US had launched five ballistic missiles.
- The trigger for the near apocalyptic disaster was traced to a fault in software that was supposed to filter out false missile detections caused by satellites picking up sunlight reflections off cloud-tops.

Software Failure Cases

- **The AT&T network collapse (1990):** In 1990, 75 million phone calls across the US went unanswered after a single switch at one of AT&T's 114 switching centers suffered a minor mechanical problem, which shut down the center.
- When the center came back up soon afterwards, it sent a message to other centers, which in turn caused them to trip and shut down and reset.

- Many more can be read here **https://www.zdnet.com/article/the-top-10-it-disasters-of-all-time/**

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.

Factors Contributing to the Software Crisis

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

The Software Engineering Challenges

- **Scale:** Development of a very large system requires a very different set of methods compared to developing a small system.
- The methods that are used for developing small systems generally *do not scale up* to large systems.
- When dealing with a small software project, the engineering capability required is low.
- *Small project* > 10 KLOC, *Medium project* > 100 KLOC, *Large project* > Million LOC, and *very large* if the size is many million LOC.

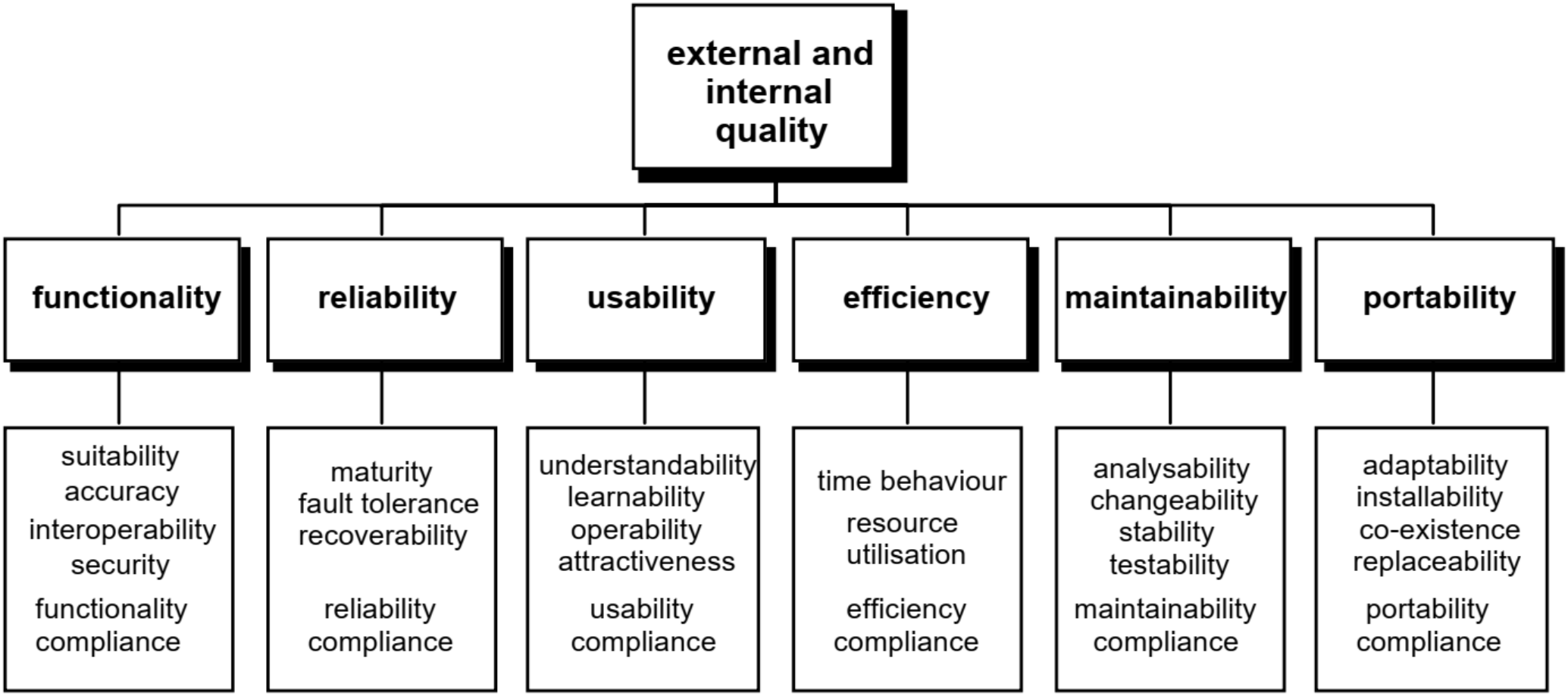
The Software Engineering Challenges

- **Quality and Productivity:** The cost of developing a system is the cost of the resources used for the system
- In the case of software, is dominated by the manpower cost, as development is largely labor-intensive.
- Productivity in terms of output (KLOC) per person-month can adequately capture both cost and schedule concerns.

The Software Engineering Challenges

- **Consistency and Repeatability:** A key challenge that software engineering faces is how to ensure that successful results can be repeated, and there can be some degree of consistency in quality and productivity.
- **Changes**: Rapid change has a special impact on software. As software is easy to change due to its lack of physical properties that may make changing harder, the expectation is much more from software for change.

The Software Engineering Challenges (ISO/IEC-9126)



The Software Engineering Challenges (ISO/IEC-9126)

- **Functionality:** The capability to provide functions which meet stated
  and implied needs when the software is used.
- **Reliability:** The capability to maintain a specified level of performance
- **Usability:** The capability to be understood, learned, and used
- **Efficiency:** The capability to provide appropriate performance relative to the amount of resources used
- **Maintainability:** The capability to be modified for purposes of making corrections, improvements, or adaptation
- **Portability:** The capability to be adapted for different specified enviornments without applying actions or means other than those provided for this purpose in the product

The Software Engineering Challenges (ISO/IEC-9126)

- **Functionality:** The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
  - **Suitability:** The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.
  - **Accuracy:** The capability of the software product to provide the right or agreed results or effects with the needed degree of precision.
  - **Interoperability:** The capability of the software product to interact with one or more specified systems.
  - **Security:** The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them, and authorized persons or systems are not denied access to them.

The Software Engineering Challenges (ISO/IEC-9126)

- **Reliability:** The capability of the software product to maintain a specified level of performance when used under specified conditions.
  - **Maturity:** The capability of the software product to avoid failure as a result of faults in the software.
  - **Fault tolerance:** The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
  - **Recoverability:** The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure.
- **Usability:** The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.
  - **Understandability:** The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
  - **Learnability:** The capability of the software product to enable the user to learn its application.
  - **Operability:** The capability of the software product to enable the user to operate and control it.
  - **Attractiveness**: The capability of the software product to be attractive to the user.

- Each quality sub-characteristic (e.g. adaptability) is further divided into attributes. An attribute is an entity which can be verified or measured in the software product.