Lecture 1-2

FUNDAMENTALS OF SOFTWARE ENGINEERING

Mehwish Mumtaz

Text Book

- Roger Pressman, Software Engineering: A Practitioner’s Approach(Selected Chapters)
- Shari Lawrence PFleeger and Joanne M. Atlee, Software Engineering Theory and Practice, Fourth Edition (Selected Chapters)
- Ian Sommerville, Software Engineering (Selected Chapters)

Why is SE Needed?

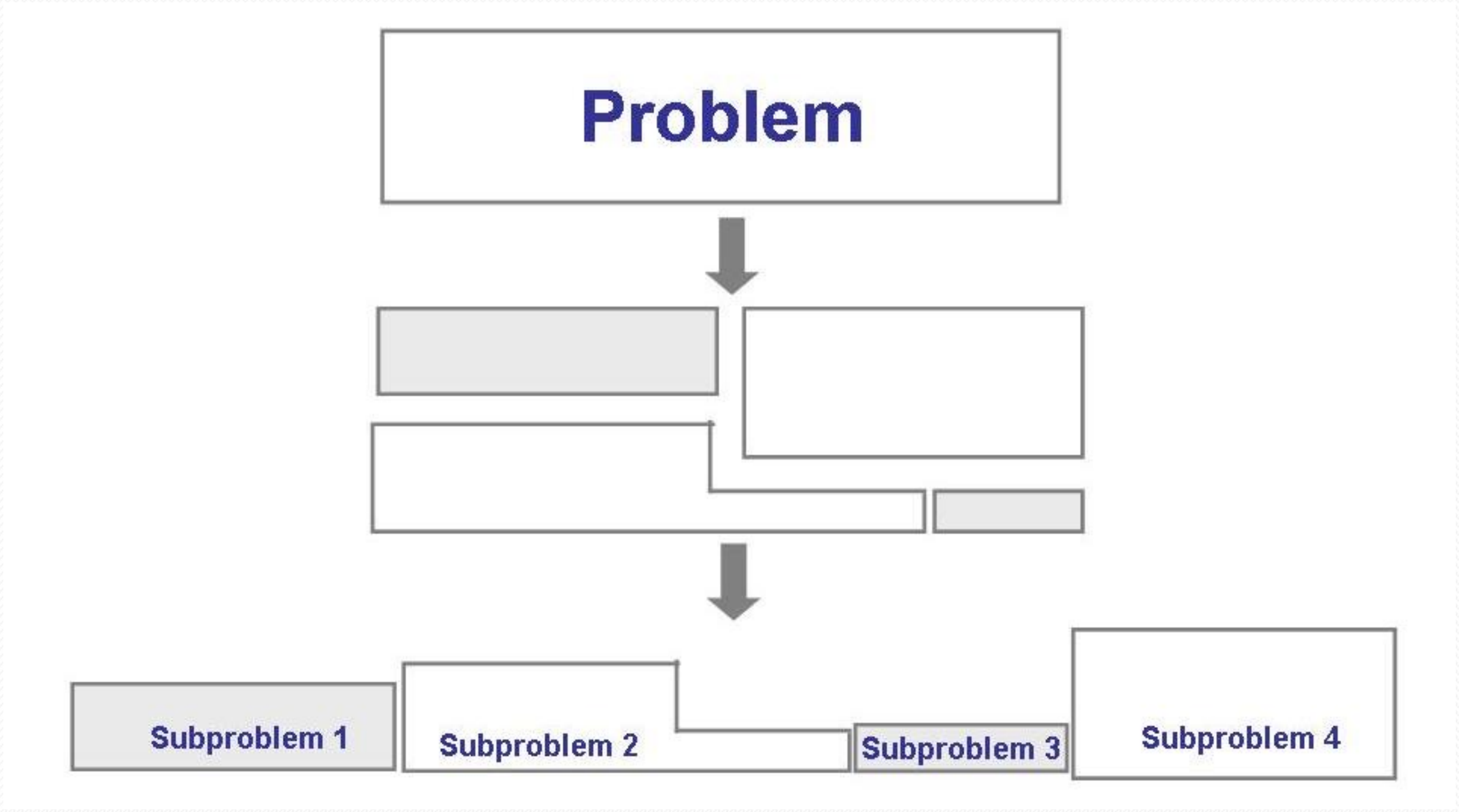
- Computers everywhere
  - Toaster, Microwave, Temperature control of A/C, Surgical Equipment…
- Computers need to be managed
  - Software runs on all computers
    - Make lives comfortable, efficient, effective…
    - E.g Document system. Estamping PITB
- SE practices ensure development of good software to improve our living standard

Why Do We Need to Study SE?

- What could be the benefits?

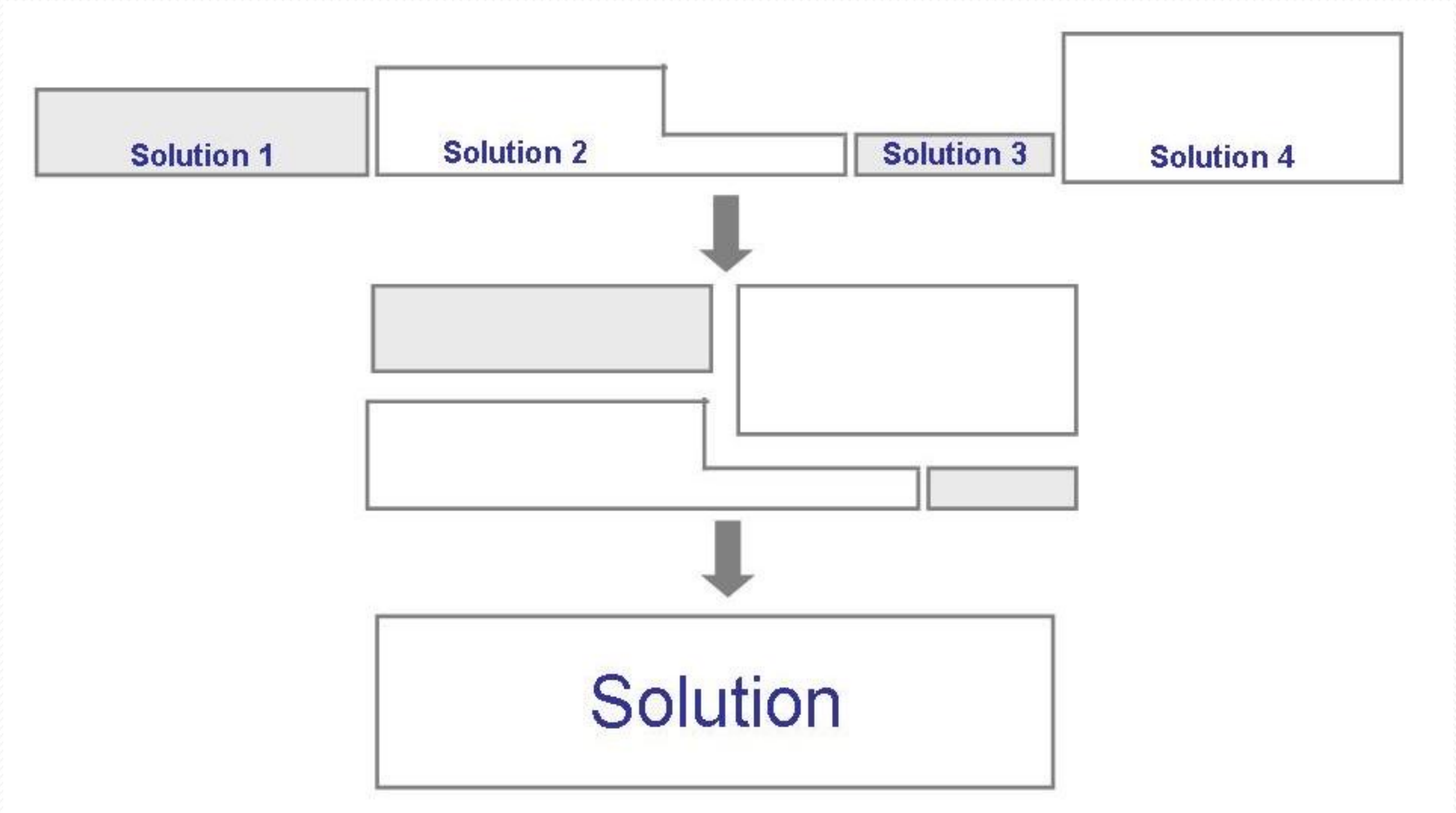
Solving Problems

- Analysis



Solving Problems (continued)

- Synthesis



Software Engineering

- Solving Problems
  - Computers
  - Computing

Software?

- Computer programs along with associated configuration data and documentation s.t. the programs are correctly operated
  - Configuration data helps set up the programs
  - System documentation helps understand structure of the system
  - User documentation explains how to use the system

Engineering?

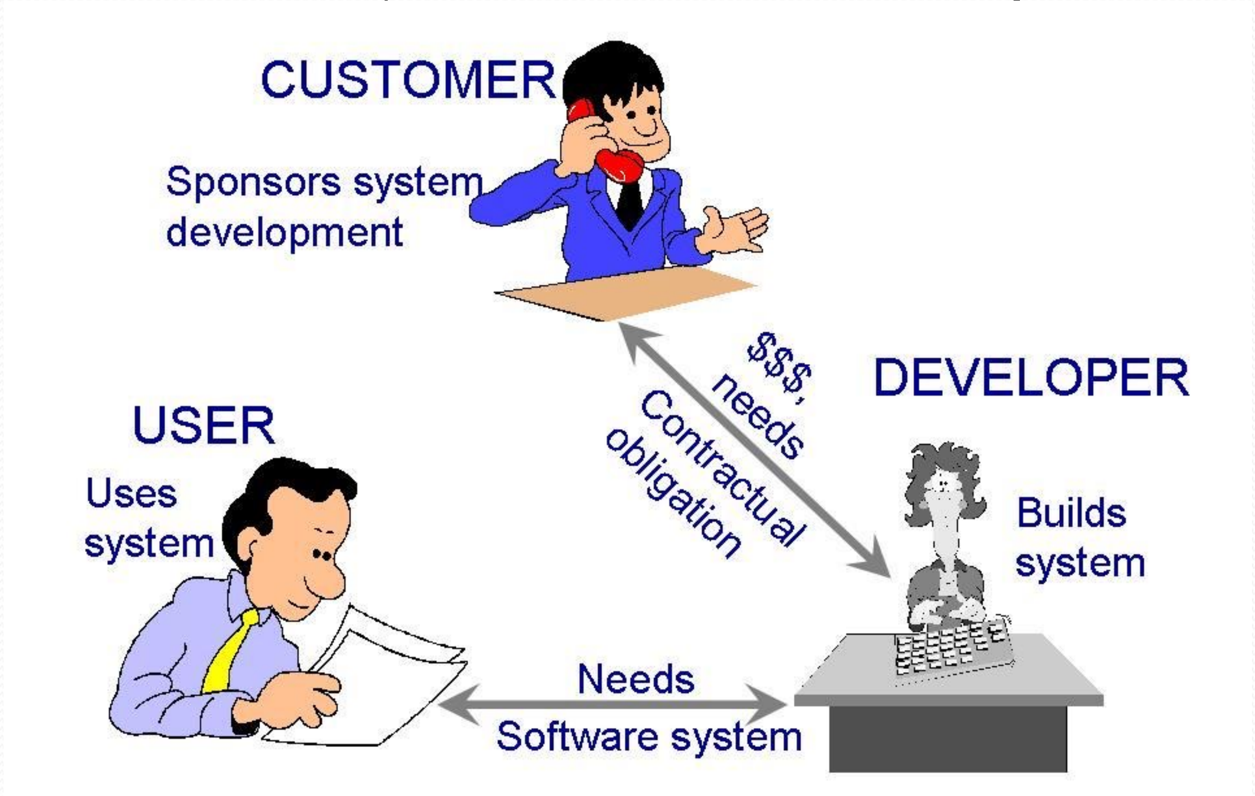
- Engineers' Job?
  - Make things work
  - Apply theories, methodologies, tools appropriately
  - Provide solutions in absence of applicable theories and methods
  - Realize financial and organizational constraints

Aspects of Software Production?

- Technical process of developing software
- Activities such as management of project and teams
- Development of tools, theories, methods to support production of software

Who Does Software Engineering?

- Participants (stakeholders) in a software development project



Challenges for Software Industry

	2012	2013	2014	2015	2016
Successful Projects	27%	31%	28%	29%	32%
Challenged Projects	56%	50%	55%	52%	49%
Failed Projects	17%	19%	17%	19%	19%

Report for Year 2012 to 2016 (Standish 2016)

Factors Affecting Project Success Rate $\xrightarrow{\text{are}}$

- Project Size and Complexity
- Customer Satisfaction Level
- Development and Budget Process
- Skills of Developers and Managers

Development Team

- **Requirement analysts**: work with the customers to identify and document the requirements
- **Designers**: generate a system-level description of what the system is supposed to do
- **Programmers**: write lines of code to implement the design
- **Testers**: catch faults
- **Trainers**: show users how to use the system
- **Maintenance team**: fix faults that show up later
- **Librarians**: prepare and store documents such as software requirements
- **Configuration management team**: maintain correspondence among various artefacts

Dev

- Who

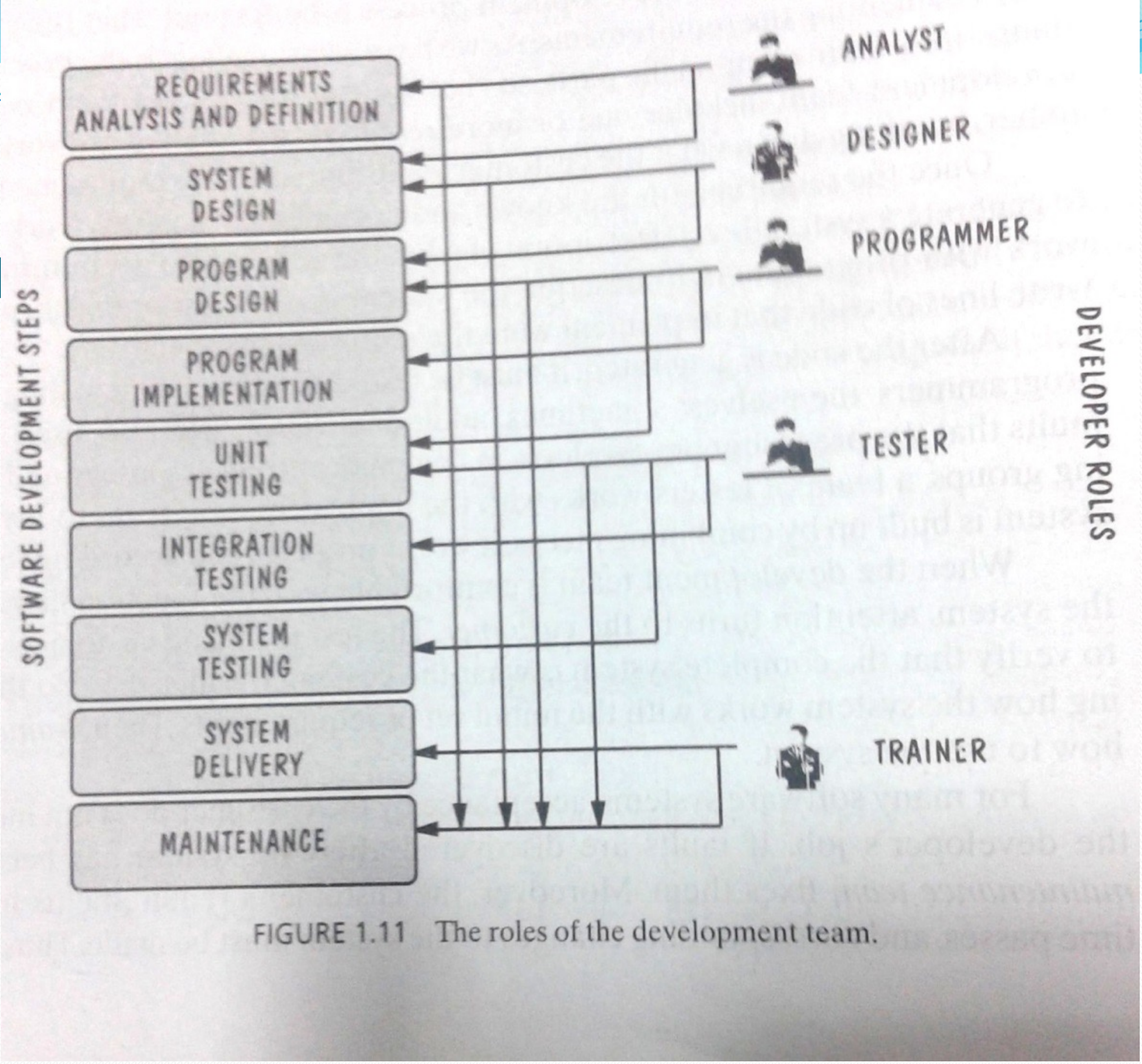
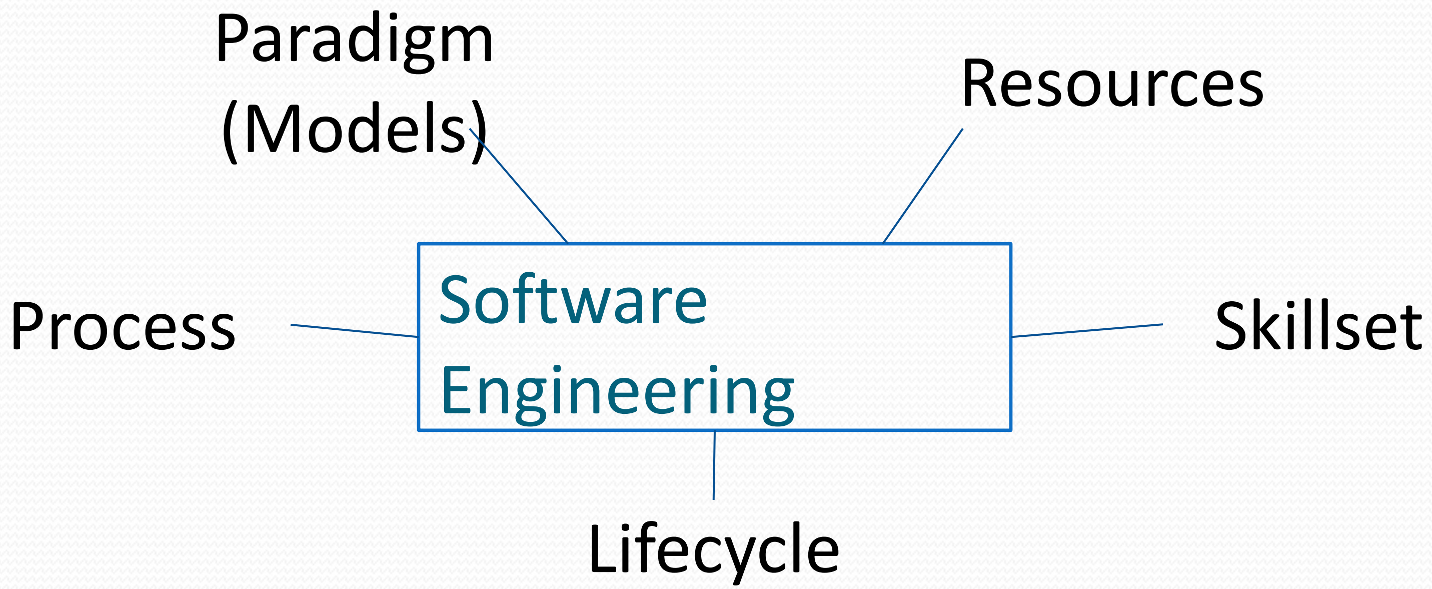


FIGURE 1.11 The roles of the development team.



Requirements Analysis	Design	Coding	Testing	Deployment	Maintenance

Typical phases in lifecycle of software

Software Lifecycle

- Phases
  - Requirements analysis and definition
  - System (architecture) design
  - Program (detailed/procedural) design
  - Writing programs (coding/implementation)
  - Testing: unit, integration, system
  - System delivery (deployment)

  - Maintenance

Software Engineers?

- Adopt a systematic and organized approach, effectively, to produce high quality software
- May have to use Ad hoc(Necessary) approaches to develop software
    - Some real complex problems may not be solved using elegant theories of CS

What is a Good Software Product?

- Good software engineering must always include a strategy for producing quality software
- Product Quality?
  - Multiple facets…

What is a Good Software Product?

- Users judge external characteristics (e.g., correct functionality, number of failures, type of failures)
- Designers and maintainers judge internal characteristics (e.g., ease of modification)
- Thus different stakeholders may have different criteria
- Need quality models

What is a Good Software Product?
McCall's Quality Factors

1. **Correctness**

   The extent to which a software meets its requirements specification.(Fulfill Customer Objective).

   1. **Functional Requirements:** Directly specify e.g Mobile application
   2. **Non functional Requirements**: Loading time of application, Performance of the application

2. **Efficiency**

   The amount of hardware resources and code the software, needs to perform a function.

3. **Integrity**

   The extent to which the software can control an unauthorized person from the accessing the data or software.

4. **Reliability**

   The extent to which a software performs its intended functions without failure.(Not a serious error. Therac 25. We expect errors)

5. **Usability**

   The extent of effort required to learn, operate and understand the functions of the software.

What is a Good Software Product? McCall's Quality Factors

6. Maintainability –
The effort required to detect and correct an error during maintenance phase. (Modification in Initial release, bug fixing phase, software evolution phase, old software and reverse engineering)

7. Flexibility
The effort needed to improve an operational software program.

8. Testability –
The effort required to verify a software to ensure that it meets the specified requirements.

9. Portability –
The effort required to transfer a program from one platform to another.

10. Re-usability –
The extent to which the program's code can be reused in other applications. E.g Car

11. Interoperability –
The effort required to integrate two systems with one another.

Lehman’s Laws of Software Evolution

- (1974) "Continuing Change" — A system must be continually adapted or it becomes progressively less satisfactory. It happens so until it becomes economical to replace it by a new or a restructured version.-Add new features e.g Soneri Bank software
- (1974) "Increasing Complexity/Entropy" —Complexity/entropy of a system increases with time, unless work is done to maintain or reduce it-lines of code
- (1991) "Continuing Growth" — the functional content of a system must be continually increased to maintain user satisfaction over its lifetime
- (1996) "Declining Quality" — the quality of a system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes

Software Engineering

- Disciplined, consistent, and systematic effort to construct (design + build) and maintain good quality software in timely and cost-effective manner

How Successful Have We Been

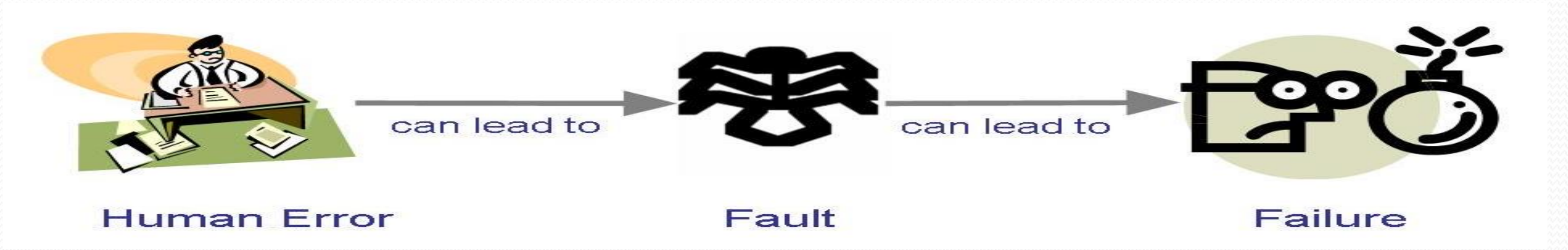
- Perform tasks more quickly and effectively
  - Word processing, spreadsheets, e-mail
- Support advances in medicine, agriculture, transportation, multimedia education, and most other industries
- Many good stories
- However, software is not without problems (recall the Standish report on challenged and failed projects???)

How Successful Have We Been

- Malfunctioning in Therac-25 killed several people
- Reliability constraints have caused cancellation of many *safety critical* systems
  - *Safety-critical*: something whose failure poses a threat to life or health

Terminology for Describing Bugs

- **A fault**: occurs when a human makes a mistake, called **an error**, in performing some software activities
- **A failure**: is a departure from the system's required behaviour. A failure indicates that the system is not performing as required. Any type of failure in a system can open it up to fuzz testing. Fuss testing "is a quality assurance technique used to discover coding errors and security loopholes in software, operating systems or networks. It involves inputting massive amounts of random data, called fuzz, to the test subject in an attempt to make it crash.



Terminology for Describing Bugs

- **Defect:**
  - If the actual result of the software deviates from the one expected and anticipated by the team of testers, while testing the software, then it results into a defect
- Examples
  - An error in coding or logic that impacts the software and causes it to malfunction or perform in an inaccurate manner.
  - Any deviation from the customer requirements also leads to defects in the software.
  - Troubles and mistakes in the external behaviour and internal structure and design.
  - Providing wrong and inaccurate inputs.
- Bugs
  - Bugs are the most integral part of a software system and can be termed as the errors, flaws, and faults present in the computer program that impact the performance as well as the functionality of the software can cause it to deliver incorrect and unexpected results. These not only impact the performance of the software, but also cause it to behave in an unanticipated way.
  - Reasons for Bugs:
  - From errors and mistakes made in the source code of the software, to issues and flaws found in the software design,
  - Error or flaw found in the development environment of the software.
  - Issue in the software or hardware that leads it to malfunction.
  - Discrepancies in the Operating System used by the program.
  - Few bugs are even caused due to incorrect codes produced by compilers.

An Engineering Approach

- Idea to build a house
- Asking someone to build the house
- Explaining requirements
- Getting designs
- Modifying + Approving designs
- Inspecting the construction
- Adding new features
- Testing household components
- Moving in
- Getting issues fixed after moving in

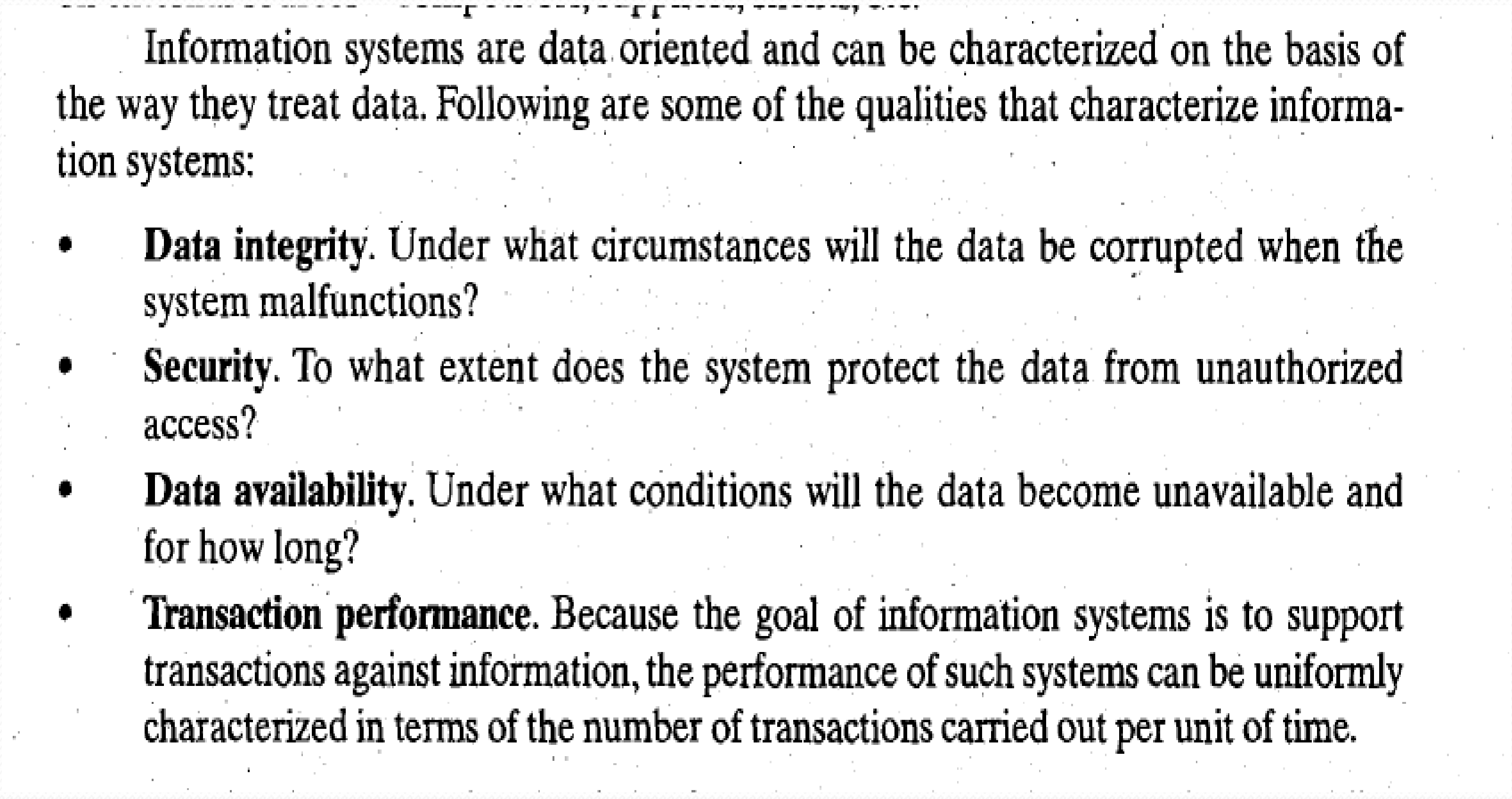
Building a House

An Engineering Approach

- Requirement analysis and definition
- System design
- Program design
- Writing the programs
- Unit testing
- Integration testing
- System testing
- System delivery
- Maintenance

Building a System

Information Systems-Quality

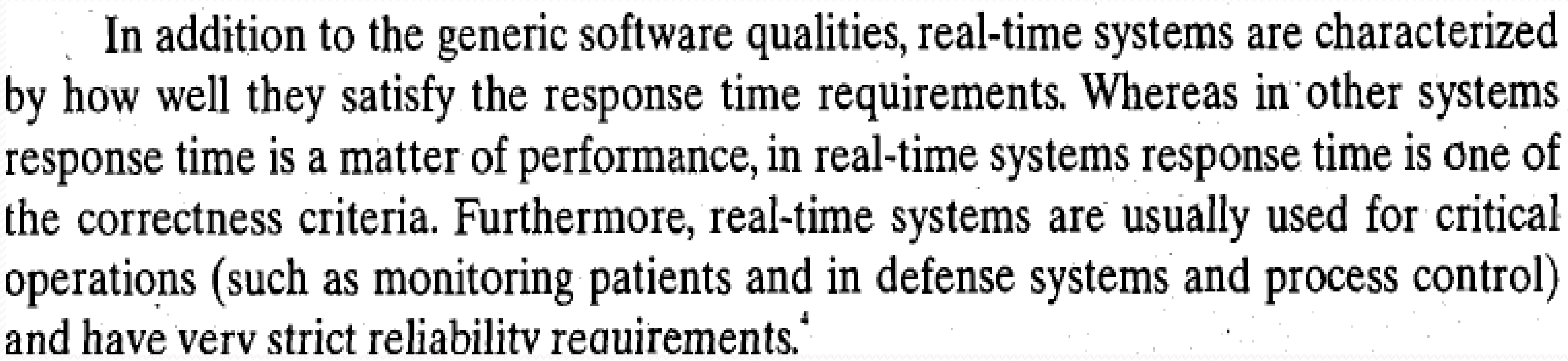


Information systems are data oriented and can be characterized on the basis of the way they treat data. Following are some of the qualities that characterize information systems:

- **Data integrity**. Under what circumstances will the data be corrupted when the system malfunctions?
- **Security**. To what extent does the system protect the data from unauthorized access?
- **Data availability**. Under what conditions will the data become unavailable and for how long?
- **Transaction performance**. Because the goal of information systems is to support transactions against information, the performance of such systems can be uniformly characterized in terms of the number of transactions carried out per unit of time.

Real Time Example

- Factory system respond to a high temperature.
- Mouse clicking

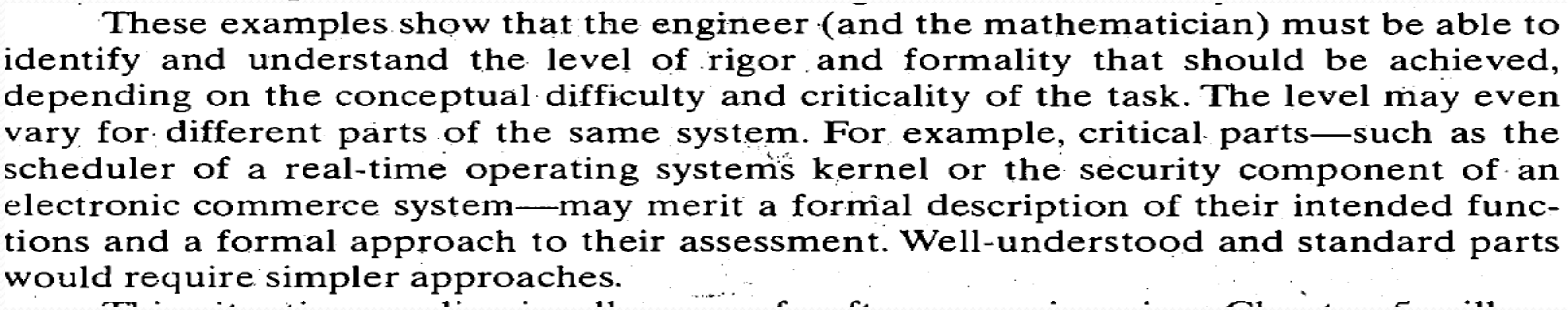


In addition to the generic software qualities, real-time systems are characterized by how well they satisfy the response time requirements. Whereas in other systems response time is a matter of performance, in real-time systems response time is one of the correctness criteria. Furthermore, real-time systems are usually used for critical operations (such as monitoring patients and in defense systems and process control) and have very strict reliability requirements.[4]

Software Engineering Principles Successful Software Development

Some principles are:

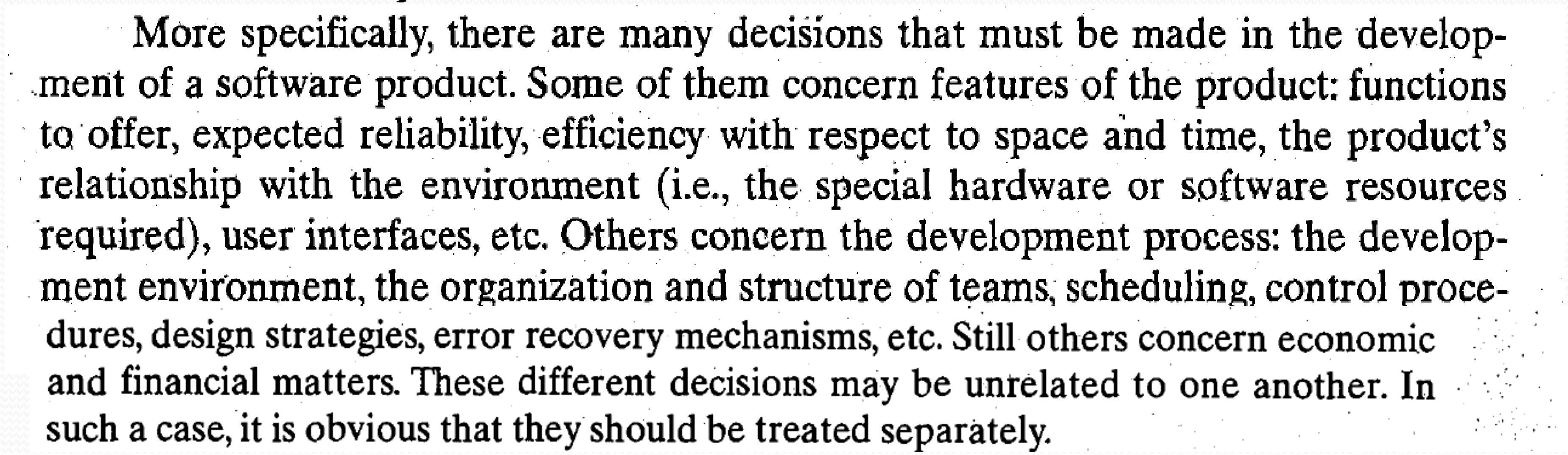
- Rigor and Formality
- Software Development -Creative Activity
- Rigor Approach produce desirable products
- Applied methods and techniques and then find the results
- Not always to be formal methods-past experience but if not experience then use mathematical model
  - Precision and exactness
  - Assessment of engineering activity and the results.
  - Example



These examples show that the engineer (and the mathematician) must be able to identify and understand the level of rigor and formality that should be achieved, depending on the conceptual difficulty and criticality of the task. The level may even vary for different parts of the same system. For example, critical parts—such as the scheduler of a real-time operating systems kernel or the security component of an electronic commerce system—may merit a formal description of their intended functions and a formal approach to their assessment. Well-understood and standard parts would require simpler approaches.

Software Engineering Principles Successful Software Development

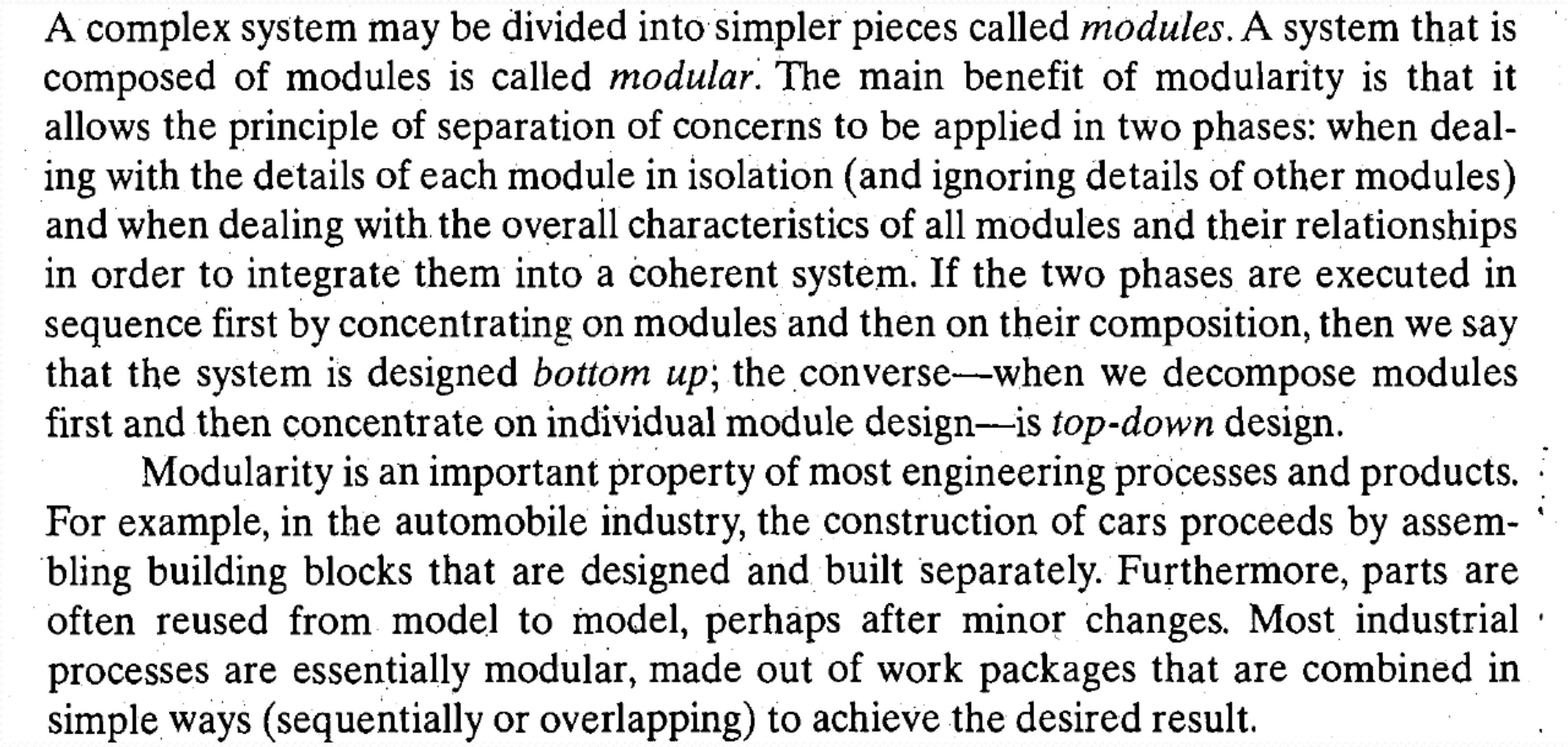
- Separation of Concerns
- Dealing with the problem
  - Correctness and efficiency
  - Parts of system



More specifically, there are many decisions that must be made in the development of a software product. Some of them concern features of the product: functions to offer, expected reliability, efficiency with respect to space and time, the product's relationship with the environment (i.e., the special hardware or software resources required), user interfaces, etc. Others concern the development process: the development environment, the organization and structure of teams, scheduling, control procedures, design strategies, error recovery mechanisms, etc. Still others concern economic and financial matters. These different decisions may be unrelated to one another. In such a case, it is obvious that they should be treated separately.

Software Engineering Principles Successful Software Development

- Modularity

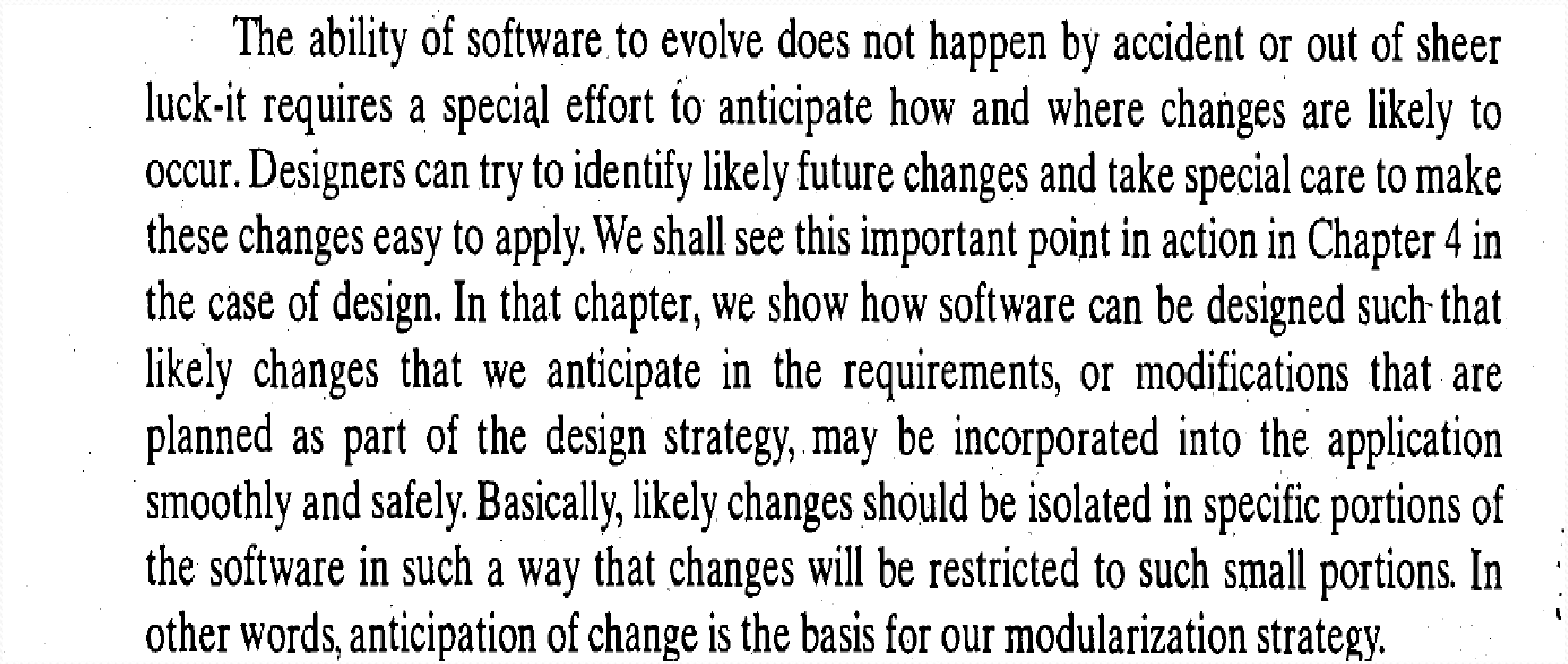


A complex system may be divided into simpler pieces called *modules*. A system that is composed of modules is called *modular*. The main benefit of modularity is that it allows the principle of separation of concerns to be applied in two phases: when dealing with the details of each module in isolation (and ignoring details of other modules) and when dealing with the overall characteristics of all modules and their relationships in order to integrate them into a coherent system. If the two phases are executed in sequence first by concentrating on modules and then on their composition, then we say that the system is designed *bottom up*; the converse—when we decompose modules first and then concentrate on individual module design—is *top-down* design.

Modularity is an important property of most engineering processes and products. For example, in the automobile industry, the construction of cars proceeds by assembling building blocks that are designed and built separately. Furthermore, parts are often reused from model to model, perhaps after minor changes. Most industrial processes are essentially modular, made out of work packages that are combined in simple ways (sequentially or overlapping) to achieve the desired result.

Software Engineering Principles Successful Software Development

- Anticipation of Change



The ability of software to evolve does not happen by accident or out of sheer luck-it requires a special effort to anticipate how and where changes are likely to occur. Designers can try to identify likely future changes and take special care to make these changes easy to apply. We shall see this important point in action in Chapter 4 in the case of design. In that chapter, we show how software can be designed such that likely changes that we anticipate in the requirements, or modifications that are planned as part of the design strategy, may be incorporated into the application smoothly and safely. Basically, likely changes should be isolated in specific portions of the software in such a way that changes will be restricted to such small portions. In other words, anticipation of change is the basis for our modularization strategy.

A Systems Approach

- Hardware/Software
- People
  - Interactions

A Systems Approach (Contd.)

- Objects
  - i.e. things
- Activities
  - Actions taken
  - Input / Outputs
- Relationships for example:
  - Which object performs what activities
  - Which objects are associated with other objects
- System Boundary
  - Who generates input and who receives output
  - Which objects/activities are part of the system and which are not
  - Nested systems, related systems, interrelated systems

References

1. Shari Lawrence PFleeger and Joanne M. Atlee, Software Engineering Theory and Practice, Fourth Edition
2. Roger Pressman, Software Engineering: A Practitioner's Approach
3. Ghezzi et al., Fundamentals of Software Engineering
4. Book slides from UCF

Acknowledgement

- A few slides have been reused from UCF slides for the SE course