MODULE-1

FAQs about software engineering

This section is designed to answer some fundamental questions about software engineering and to give some impression about the discipline. The format that is used here is the 'FAQ (Frequently Asked Questions) list'. This approach is commonly used in Internet news groups to provide newcomers with answers to frequently asked questions. Figure 1.1 summarizes the answers to the questions in this section.

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What is software engineering?	Software engineering is an engineering discipline which is concerned with all aspects of software production.
What is the difference between Software Engineering and computer science?	Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between Software Engineering and System? Engineering ?	System engineering is concerned with all aspects of computer-based systems development, including hardware, software and process engineering. Software engineering is part of this process.
What is a software process?	set of activities whose goal is the development or evolution of software.
What is a software process model?	simplified representation of a software process, presented from a specific perspective.
What are the costs of software engineering?	Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.

What are software engineering methods?	Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
What is CASE (Computer-Aided Software Engineering)?	Software systems which are intended to provide automated support for software process activities. CASE systems *are* often used for method support.
What are the attributes of good Software?	The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What are the key challenges facing software engineering	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.

What is software?

Software is not just the programs but also all associated documentation and configuration data that is needed to make these programs operate correctly. A software system usually consists of a number of separate programs, configuration files, which are used to set up these programs, system documentation, which describes the structure of the system, and user documentation, which explains how to use the system and web sites for users to download recent product information.

Software engineers are concerned with developing software products, i.e., software which can be sold to a customer. There are two fundamental types of software product:

***1*. Generic products:** These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples of this type of product include software for PC's such as databases, word processors, drawing packages and project management tools.

2. Customized (or bespoke) products: These are systems which are commissioned by a particular customer. A software contractor develops the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process and air traffic control systems.

- An important difference between these types of software is that, in generic products, the organization that develops the software controls the software specification.
- For custom products, the specification is usually developed and controlled by the organization that is buying the software.
- More and more software companies are starting with a generic system and customizing it to the needs of a particular customer.
- **Example:** Enterprise Resource Planning (ERP) systems, such as the SAP system, are the best examples of this approach.

What is software engineering?

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

1. **Engineering discipline:** Engineers make things work. They apply theories, methods and tools where these are appropriate

2. All aspects of software production: Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production.

What's the difference between software engineering and computer science?

- Essentially, computer science is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software.
- Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics IS essential for electrical engineers. Ideally, all of software engineering should be underpinned by theories of computer science, but in reality this IS not the case.
- Software engineers must often use *ad hoc* approaches to developing the software. Elegant theories of computer science cannot always be applied to real, complex problems that require a software solution.

What is the difference between software engineering and system engineering?

- System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is therefore concerned with hardware development, policy and process design and system deployment as well as software engineering.
- System engineers are involved in specifying the system, defining its overall architecture and then integrating the different parts to create the finished system. System engineering is an older discipline than software engineering.

What is a software process?

A software process is the set of activities and associated results that produce a software product. There are four fundamental process activities that are common to all software processes. These are:

1. Software specification where customers and engineers define the software to be produced and the constraints on its operation.

2. Software development where the software is designed and programmed.

3. Software validation where the software is checked to ensure that it is what the customer requires.

4. Software evolution where the software is modified to adapt it to changing customer and market requirements. Different types of systems need different development processes.

For example, real-time software in an aircraft has to be completely specified before development begins whereas, in e-commerce systems, the specification and the program are usually developed together.

What is a software process model?

- A software process model is a simplified description of a software process that presents one view of that process.
- Process models may include activities that are part of the software process, software products and the roles of people involved in software engineering.
- Some examples of the types of software process model that may be produced are:

  1. A workflow model: This shows the sequence of activities in the process along with their inputs, outputs and dependencies. The activities in this model represent human actions.

  2. A dataflow or activity model: This represents the process as a set of activities, each of which carries out some data transformation. It shows how the input to the process, such as a specification, is transformed to an output, such as a design.

3. A role/action model: This represents the roles of the people involved in the software process and the activities for which they are responsible.

- Most software process models are based on one of three general models or paradigms of software development:

1. The waterfall approach This takes the above activities and represents them as separate process phases such as requirements specification, software design, implementation, testing and so on. After each stage is defined it is 'signed-off, and development goes on to the following stage.

2. Iterative development This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from very abstract specifications. This is then refined with customer input to produce a system that satisfies the customer s needs.

3. Component-based software engineering (CBSE) This technique assumes that parts of the system already exist.

What are the attributes of good software?

As well as the services that it provides, software products have a number of other associated attributes that reflect the quality of that software. These attributes are not directly concerned with what the software does. Rather, they reflect its behavior while it is executing and the structure and organization of the source program and associated documentation. Examples of these attributes (sometimes called nonfunctional attributes) are the software s response time t9 a user query and the understandability of the program code. The specific set of attributes that you might expect from a software system obviously depends on its application. Therefore, a banking system must be secure, an interactive game must be responsive, a telephone switching system must be reliable, and so on. These can be generalized into the set of attributes shown in Figure 1.5, which, I believe, are the essential characteristics of a well-designed software system.

Product Characteristics	**escription**
Maintainability	Software should be written in such a way that it may evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable consequence of a changing business environment.
Dependability	Software dependability has a range of characteristics, including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of

	system failure.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Usability	Software must be usable, without undue effort, by the type of User for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

What are the key Challenges facing software engineering?

Software engineering in the 21st century faces three key challenges:

1. The heterogeneity challenge: Increasingly, systems are required to operate as distributed systems across networks that include different types of computers and with different kinds of support systems. It is often necessary to integrate new software with older legacy systems written in different programming languages. The heterogeneity challenge is the challenge of developing techniques for building dependable software that is flexible enough to cope with this heterogeneity.

2. The delivery challenge: Many traditional software engineering techniques are time-consuming. The time they take is required to achieve software quality. However, businesses today must be responsive and change very rapidly.

3. The trust challenge: As software is intertwined with all aspects of our lives, it is essential that we can trust mat software. This is especially true for remote software systems accessed through a web page or web service interface.

Professional and ethical responsibility

- Like other engineering disciplines, software engineering is carried out within a legal and social framework that limits the freedom of engineers.
- Software engineers must accept that their job involves wider responsibilities than simply the application of technical skills. They must also behave in an ethical and morally responsible way

However, there are areas where standards of acceptable behavior are not bounded by laws but by the more tenuous notion of professional responsibility. Some of these are:

1. **Confidentiality** You should normally respect the confidentiality of your employers or clients irrespective of whether a formal confidentiality agreement has been signed.

2. ***Competence*** You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. ***Intellectual property rights*** You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. ***Computer misuse*** You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer s machine, say) to extremely serious (dissemination of viruses). Professional societies and institutions have an important role to play in setting ethical standards. Organisations such as the ACM, the IEEE (Institute of Electrical and Electronic Engineers) and the British computer Society publish a code of professional conduct or code of ethics. Members of these organizations undertake to follow that code when they sign up for membership. These codes of conduct are generally concerned with fundamental ethical behaviour.

   The ACM and the IEEE have cooperated to produce a joint code of ethics and professional practice. This code exists in both a short form, shown in Figure 1.6 and a longer form (Gotterbam, et al., 1999) that adds detail and substance to the

PREAMBLE:

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code. Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC - Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.

5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.

8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Socio-technical systems

A system is a purposeful collection of interrelated components that work together to achieve some objective.

Systems that include software fall into two categories:

- **Technical computer-based systems** are systems that include hardware and software components but not procedures and processes. **Examples** of technical systems include televisions, mobile phones and most personal computer software.
- **Socio-technical systems** include one or more technical systems but, crucially, also include knowledge of how the system should be used to achieve some broader objective. This means that these systems have defined operational processes, include people (the operators) as inherent parts of the system, are governed by organizational policies and rules and may be affected by external constraints such as national laws and regulatory policies. **For example**, this book was created through a socio-technical publishing system that includes various processes and technical systems. Essential characteristics of socio-technical systems

Essential characteristics of socio-technical systems are as follows:

- They have **emergent properties** that are properties of the system as a whole rather than associated with individual parts of the system. Emergent properties depend on both the system components and the relationships between them. As this is so complex, the emergent properties can only be evaluated once the system has been assembled.
- They are often **nondeterministic**. This means that, when presented with a specific input, they may not always produce the same output. The system's behavior depends on the human operators, and people do not always react in the same way. Furthermore, use of the system may create new relationships between the system components and hence change its emergent behavior.

- The extent to which the system supports organizational objectives does not just depend on the system itself. It also depends on the stability of these objectives, the relationships and conflicts between organizational objectives and how people in the organization interpret these objectives. New management may reinterpret the organizational objective that a system is designed to support, and a successful' system may then become a 'failure'.

Emergent system properties:

- Properties of the system as a whole rather than properties that can be derived from the properties of components of a system
- Emergent properties are a consequence of the relationships between system components
- They can therefore only be assessed and measured once the components have been integrated into a system

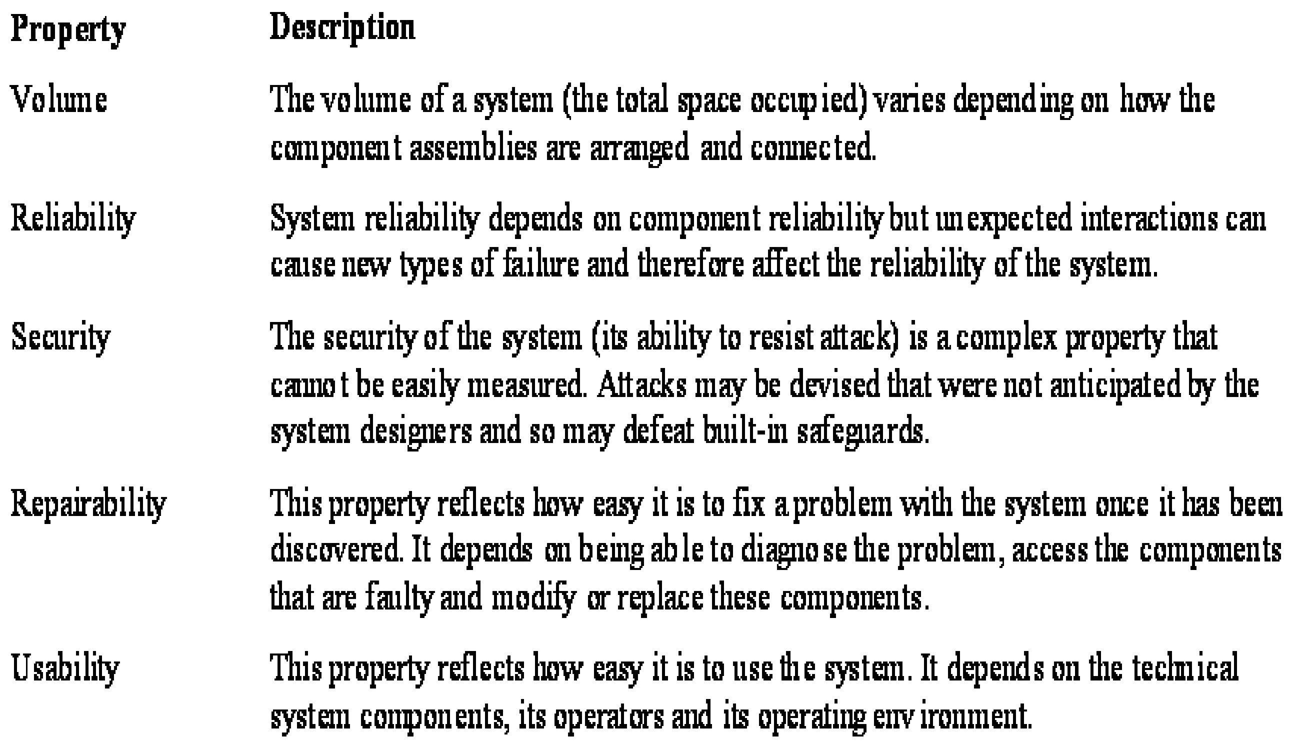
Examples of emergent properties:

- The overall weight of the system:
  - ➢ This is an example of an emergent property that can be computed from individual component properties.
- The reliability of the system:
  - ➢ This depends on the reliability of system components and the relationships between the components.
- The usability of a system
  - ➢ This is a complex property which is not simply dependent on the system hardware and software but also depends on the system operators and the environment where it is used.

Types of emergent property:

- Functional properties:
  - ➢ These appear when all the parts of a system work together to achieve some objective. For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.
- Non-functional emergent properties
  - ➢ Examples are reliability, performance, safety, and security. These relate to the behavior of the system in its operational environment. They are often critical for computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable.

Examples of emergent properties:



Property	Description
Volume	The volume of a system (the total space occupied) varies depending on how the component assemblies are arranged and connected.
Reliability	System reliability depends on component reliability but unexpected interactions can cause new types of failure and therefore affect the reliability of the system.
Security	The security of the system (its ability to resist attack) is a complex property that cannot be easily measured. Attacks may be devised that were not anticipated by the system designers and so may defeat built-in safeguards.
Repairability	This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty and modify or replace these components.
Usability	This property reflects how easy it is to use the system. It depends on the technical system components, its operators and its operating environment.

Table: Examples of emergent properties

There are three related influences on the overall reliability of a system:

- **Hardware reliability**
  - ➢ What is the probability of a hardware component failing and how long does it take to repair that component?
- **Software reliability**
  - ➢ How likely is it that a software component will produce an incorrect output. Software failure is usually distinct from hardware failure in that software does not wear out.
- **Operator reliability**
  - ➢ How likely is it that the operator of a system will make an error?

Reliability relationships:

- Hardware failure can generate spurious signals that are outside the range of inputs expected by the software.
- Software errors can cause alarms to be activated which cause operator stress and lead to operator errors.

- The environment in which a system is installed can affect its reliability.

<u>The system engineering process</u>

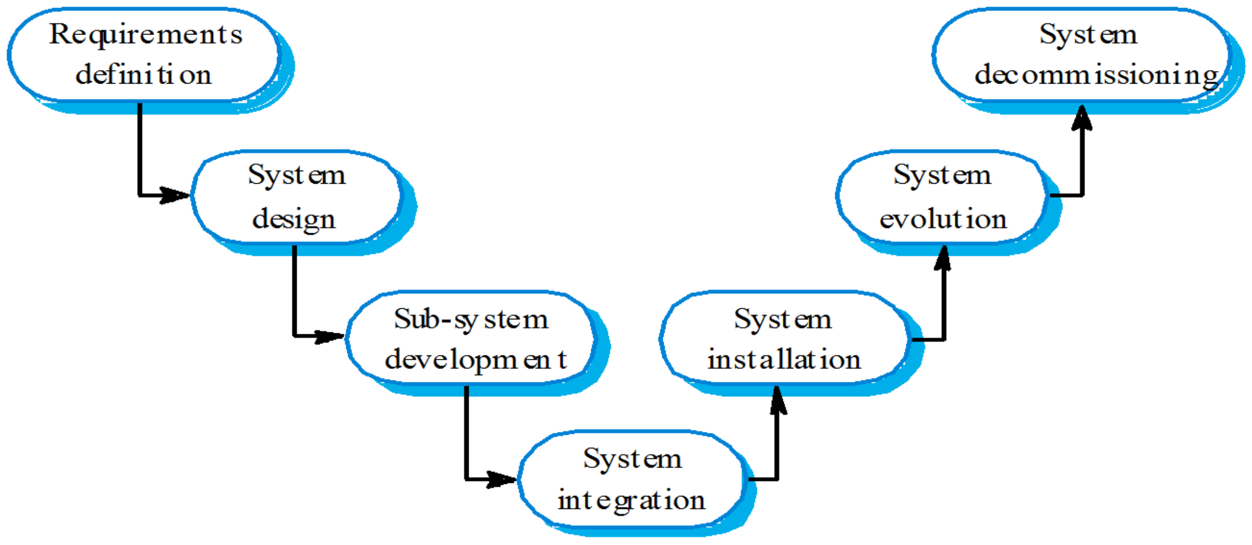


Figure: **Systems engineering process**

- Usually follows a 'waterfall' model because of the need for parallel development of different parts of the system.
  - ➢ Little scope for iteration between phases because hardware changes are very expensive. Software may have to compensate for hardware problems.
- Inevitably involves engineers from different disciplines who must work together
  - ➢ Much scope for misunderstanding here. Different disciplines use a different vocabulary and much negotiation is required. Engineers may have personal agendas to fulfil.
- There are important distinctions between the system engineering process and the software development process:
  - ➢ Limited scope for rework during system development once some system engineering decisions, such as the sitting of base stations in a mobile phone system, have been made, they are very expensive to change. Reworking the system design to solve these problems is rarely possible.
  - ➢ Interdisciplinary involvement many engineering disciplines may be involved in system engineering. There is a lot of scope for misunderstanding because different engineers use different terminology and conventions.

- Systems engineering is an interdisciplinary activity involving teams drawn from various backgrounds.
- System engineering teams are needed because of the wide knowledge required to consider all the implications of system design decisions.
- As an illustration of this, Figure 2.3 shows some of the disciplines that may be involved in the system engineering team for an air traffic control (ATe) system that uses radars and other sensors to determine aircraft position.
- Different disciplines negotiate to decide how functionality should be provided. Often there is no correct' decision on how a system should be decomposed.

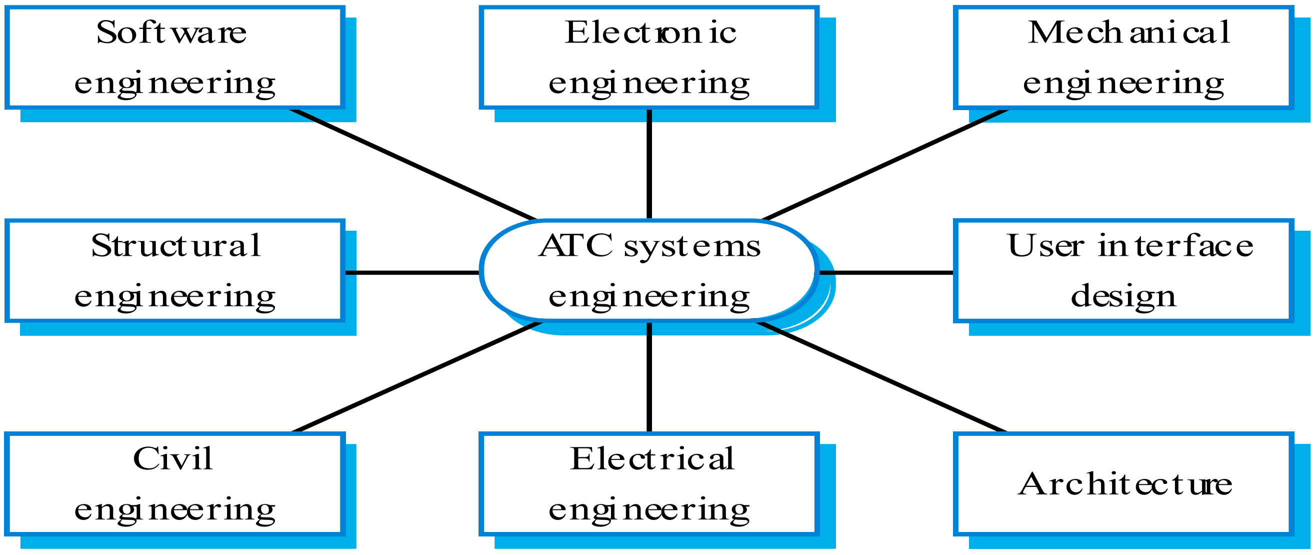


Figure: Disciplines involved in systems engineering

System requirements definition

- Three types of requirement defined at this stage:
  - ➢ Abstract functional requirements. System functions are defined in an abstract way.
  - ➢ System properties. Non-functional requirements for the system in general are defined.
  - ➢ Undesirable characteristics. Unacceptable system behavior is specified.
    - Should also define overall organizational objectives for the system.
    - A statement of objectives based on the system functionality might be:

System design

System design is concerned with how the system functionality is to be provided by the components of the system.

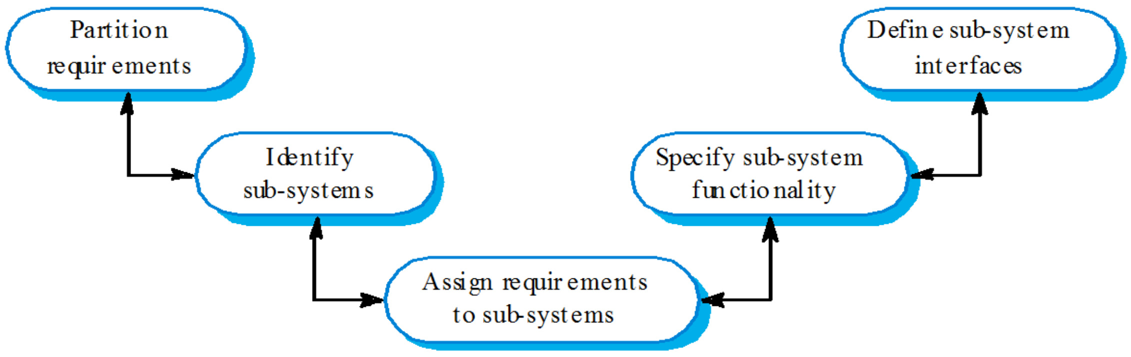

Partition requirements
Identify sub-systems
Assign requirements to sub-systems
Specify sub-system functionality
Define sub-system interfaces

Figure: System Design Process

The activities involved in this process are:

- **Partition requirements**
  - ➢ Organize requirements into related groups
- **Identify sub-systems**
  - ➢ Identify a set of sub-systems which collectively can meet the system requirements.
- **Assign requirements to sub-systems**
  - ➢ Causes particular problems when COTS are integrated.
- **Specify sub-system functionality**
- **Define sub-system interfaces**
  - ➢ Critical activity for parallel sub-system development

System design problems:

- Requirements partitioning to hardware, software and human components may involve a lot of negotiation.
- Difficult design problems are often assumed to be readily solved using software.
- Hardware platforms may be inappropriate for software requirements so software must compensate for this.

Requirements and design:

- Requirements engineering and system design are inextricably linked.
- Constraints posed by the system's environment and other systems limit design choices so the actual design to be used may be a requirement.
- Initial design may be necessary to structure the requirements.
- As you do design, you learn more about the requirements.

Spiral model of requirements/design:

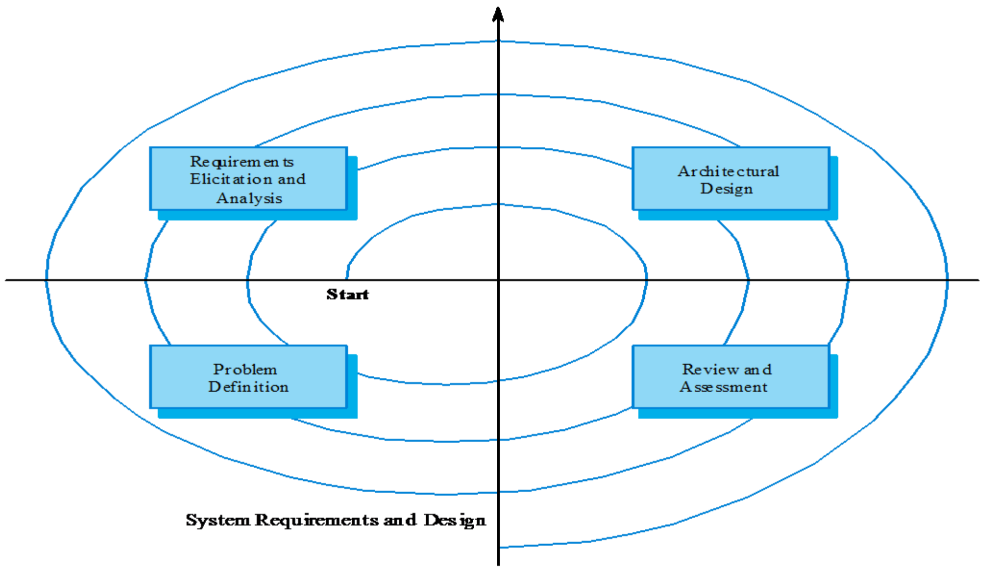

Requirements Elicitation and Analysis
Architectural Design
Start
Problem Definition
Review and Assessment
System Requirements and Design

Figure: A spiral model of requirements and design

Legacy systems:

- Socio-technical systems that have been developed using old or obsolete Technology.
- Crucial to the operation of a business and it is often too risky to discard these
- systems
- Bank customer accounting system;
- Aircraft maintenance system.
- Legacy systems constrain new business processes and consume a high proportion of company budgets.

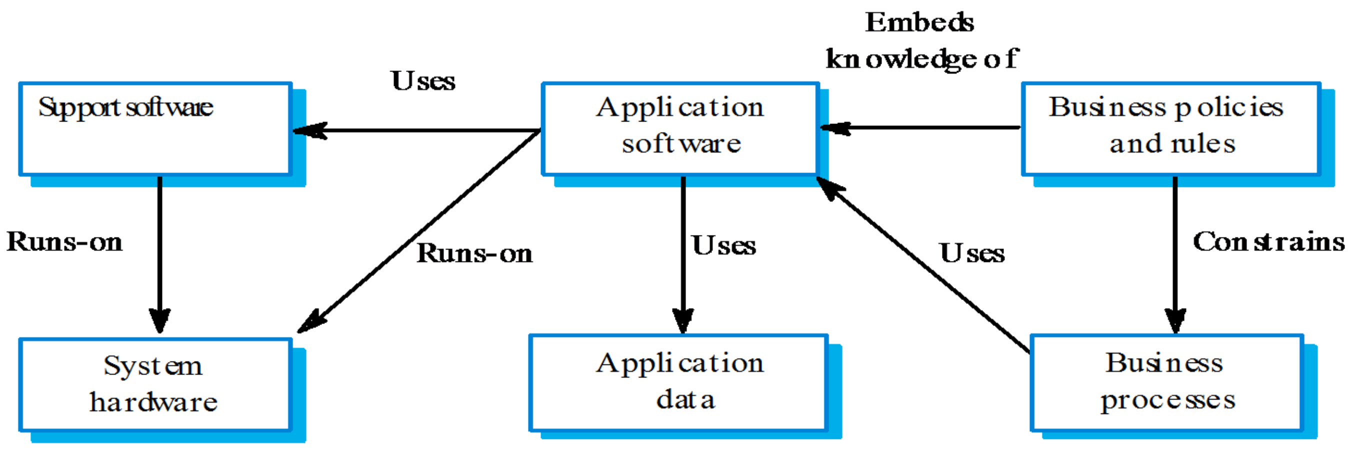


Figure: Legacy system components

<u>Legacy system components:</u>

- Hardware - may be obsolete mainframe hardware.
- Support software - may rely on support software from suppliers who are no longer in business.
- Application software - may be written in obsolete programming languages.
- Application data - often incomplete and inconsistent.
- Business processes - may be constrained by software structure and functionality.
- Business policies and rules - may be implicit and embedded in the system software.

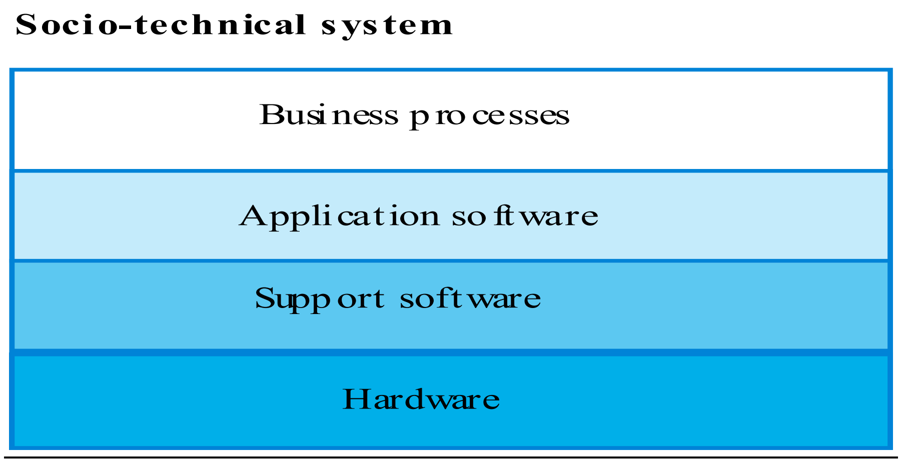


Figure: Layered model of a legacy system