1. Introduction

Objectives

The objectives of this chapter are to introduce software engineering and to provide a framework for understanding the rest of the book. When you have read this chapter you will:

1. understand what software engineering is and why it is important;
2. understand that the development of different types of software systems may require different software engineering techniques;
3. understand some ethical and professional issues that are important for software engineers

Contents

1.1 FAQs about software engineering

This section is designed to answer some fundamental questions about software engineering.

Fig 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?	A set of activities whose goal is the development or evolution of software.
What is a software process model?	A 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.

1.1.1 What is software?

software is a computer programs. It 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

1. number of separate programs
2. configuration files- which are used to set up these programs
3. system documentation-which describes the structure of the system
4. user documentation- which explains how to use the system and web sites for users to download information.

There are two fundamental types of software product:

1. *Generic products:* These are stand-alone systems that are produced by a development organisation and sold on the open market to any customer who is able To buy them.

Examples of this type of product include software for PCs such as

databases,

word processors

drawing packages

project management tools.

2. *Customised (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.

- custom products

The specification is usually developed and controlled by the organization that is buying the software. The software developers must work to that specification.

1.1.2 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.

In this definition, there are two key phrases:

1. *Engineering discipline* Engineers make things work. They apply theories, methods and tools where these are appropriate, try to discover solutions to problems even when there are no applicable theories

Engineers also recognise that they must work to organisational and financial constraints.

2. *All aspects of software production* Software engineering is concerned with the technical processes of software development also with activities such as software project management and with the development of tools, methods and theories to support software production.

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

computer science is concerned with the theories and methods that underlie computers and software systems.

software engineering is concerned with the practical problems of producing software. Some knowledge of computer science is essential for software engineers.

1.1.4 What is the difference between software engineering and system engineering?

System engineering is concerned with all aspects of the development and evolution of systems where software plays a major role. System engineering is concerned with hardware development, policy and process design and system deployment as well as software engineering.

Software engineering is part of the engineering process concerned with developing the software, control, applications and databases in the system.

(System engineering=Software engineering+hardware+deployment+principles)

1.1.5 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/ generic activities ,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.

1.1.6 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 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 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. The activities here may represent transformations carried out by people or by computers.

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.

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*** 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. The system development process focuses on integrating these parts rather than developing them from scratch.

1.1.7 What are the costs of software engineering?

1. Distribution of costs across the different activities in the software process depends on the process used and the type of software that is being developed.

2. If you assume that the total cost of developing a complex software system is 100 cost units then Figure 1.2 illustrates how these are spent on different process activities.

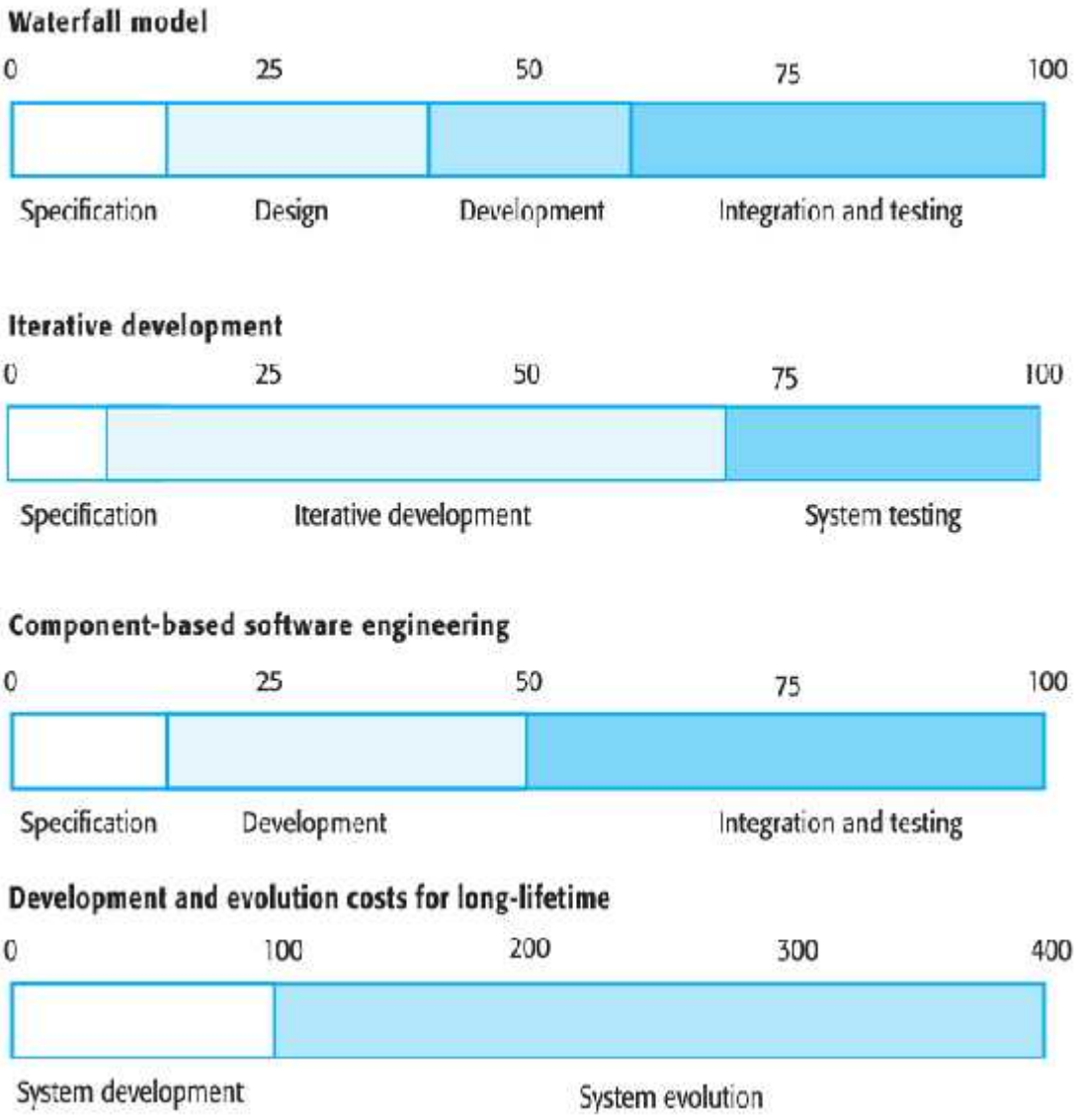


Figure 1.2 Software engineering activity cost distribution

a) **In the waterfall approach**, the costs of specification, design, implementation and integration are measured separately. System integration and testing is the most expensive development activity. Normally, this is about 40% of the total development costs but for some critical systems it is likely to be at least 50% of the system development costs.

b) **In iterative approach**, there is no hard line between specification, design and development. Specification costs are reduced because only a high-level specification is produced before development in this approach.

Specification, design, implementation, integration and testing are carried out in parallel within a development activity.

However, you still need an independent system testing activity once the initial implementation is complete.

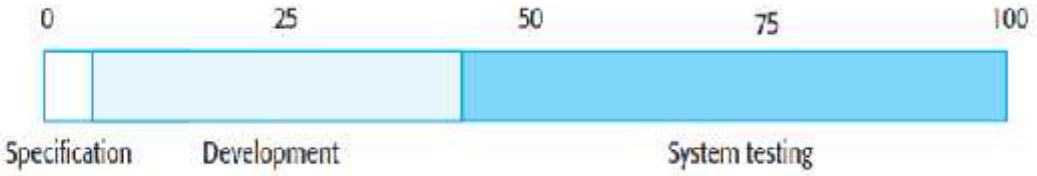
c) **In Component-based software engineering** has only been widely used for a short Time development process.

Integration and testing costs are increased because you have to ensure that the components that you use actually meet their specification and work as expected with other components.

d) **Development and evolution costs for long lifetime system**-On top of development costs, costs are also incurred in changing the software after it has gone into use. The costs of evolution vary dramatically depending on the type of system.

For long-lifetime software systems, such as control systems that may be used for 10 years or more, these costs are likely to exceed the development costs by a factor of 3 or 4, as illustrated in the bottom bar in Figure 1.3

Figure 1.3 Product development costs



3. Here products are usually developed from an outline specification, Specification costs are relatively low. However, because they are intended for use on a range of different configurations, they must be extensively tested.

4. Smaller business systems have a much shorter lifetime and correspondingly reduced evolution costs. These cost distributions hold for customised software that is specified by a customer and developed by a contractor.

1.1.8 What are software engineering methods?

A software engineering method is a structured approach to software development whose aim is to facilitate the production of high-quality software in a cost-effective way.

Methods include a number of different components (Figure 1.4).

Component	Description	Example
System model descriptions	Descriptions of the system models which should be developed and the notation used to define these models.	Object models, data-flow models, state machine models, etc.
Rules	Constraints which always apply to system models.	Every entity in a system model must have a unique name.
Recommendations	Heuristics which characterise good design practice in this method. Following these recommendations should lead to a well-organised system model.	No object should have more than seven sub-objects associated with it.
Process guidance	Descriptions of the activities which may be followed to develop the system models and the organisation of these activities.	Object attributes should be documented before defining the operations associated with an object.

Figure 1.4 Method components

1.1.9 What is CASE?

1. The acronym CASE stands for Computer-Aided Software Engineering. It covers a wide range of different types of programs that are used to support software process activities such as requirements analysis, system modelling, debugging and testing.

2. All methods associated with CASE technology include

EDITORS- for the notations used in the method

ANALYSIS MODULES- which check the system model according to the method rules

REPORT GENERATORS- to help create system documentation.

CODE GENERATOR- that automatically generates source code from the system model and some process guidance for software engineers.

1.1.10 What are the attributes of good software?

1. Attributes reflect the quality of the software,its behavior while it is executing and the structure and organisation of the source program and associated documentation. Fig 1.5

Figure 1.5 Essential attributes of good software

Product characteristic	Description
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 utilisation, 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.

1.1.11 What are the key challenges facing software engineering?

Software engineering faces three key challenges:

1. *The heterogeneity challenge* 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* The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.

3. *The trust challenge*:The trust challenge is to develop techniques that demonstrate that software can be trusted by its users.

1.2 Professional and ethical responsibility

1. Software engineers must behave in an ethical and morally responsible way if they are to be respected as professionals. They should not use their skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession.

Some of notion of professional responsibility are:

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

b). *Competence* You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.

c). *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.

d). *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).

2. ACM/IEEE code of ethics

a) .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.

b) .Members of these organisations undertake to follow that code when they sign up for membership. Fig 1.6

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

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.

Figure 1.6 ACM/IEEE Code of Ethics (©IEEE/ACM 1999)

2

Socio-technical systems

Contents

What is a system?

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

2. A system may include software, mechanical, electrical and electronic hardware and be operated by people.

3. Systems that include software fall into two categories:

a) 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.

Individuals and organisations use technical systems for some purpose but knowledge of this purpose is not part of the system.

Example, the word processor

I am using is not aware that is it being used to write a book.

b) 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.

These systems have defined operational processes,include people (the operators) as inherent parts of the system, are governed by organisational policies and rules and may be affected by external constraints such as national laws and regulatory policies.

Essential characteristics of socio-technical systems are.

A). They have EMERGENT PROPERTIES :are properties of the system as a whole associated with individual parts of the system.

Emergent properties depend on both the system components and the relationships between them.

B). 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, who do not always react in the same way.

C). The extent to which the system supports ORGANISATIONAL OBJECTIVES does not just depend on the system itself.

It also depends on the stability of these objectives, the relationships and conflicts between organisational objectives and how people in the organisation interpret these objectives.

2.1 Emergent system properties

1. properties of the system as a whole rather than properties that can be derived from the properties of components of a system.

2. Emergent properties are a consequence of relationships between the system components.

3. These *emergent properties* cannot be attributed to any specific part of the system. Rather, they emerge only once the system components have been integrated.

4. Examples of some emergent properties are shown in Figure 2.1.

5. There are two types of emergent properties:

a). *Functional emergent properties* 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.

b). *Non-functional emergent properties* relate to the behaviour of the system in its operational environment.

Examples of non-functional properties are reliability, performance, safety and security. These are often critical for computer-based systems, as failure to

achieve some minimal defined level in these properties may make the system unusable.

Figure 2.1 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.

6. To illustrate the complexity of emergent properties, consider the property of system

RELIABILITY. Reliability is always be considered at the system level rather than at the individual component level.

a)The components in a system are interdependent, so failures in one component can be propagated through the system and affect the operation of other components.

b) 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?

-All of these are closely linked. Hardware failure can generate spurious signals that are outside the range of inputs expected by software.

-The software can then behave unpredictably.

-when system failures are occurring. These operator errors may further stress the hardware, causing more failures, and so on.

-Thus, the initial, recoverable failure can rapidly develop into a serious problem requiring a complete system shutdown.

7. however some properties are properties that the system should not exhibit

Safety-the system should not behave in an unsafe way

Security-the system should not permit un-authorised use

-measuring or assessing these properties is very hard.

2.2 Systems engineering

1. Systems engineering is the activity of specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.

2 .It is an activity concerned with the services that the system provides, the constraints under which the system must be built and operated and the ways in which the system is used to fulfill its purpose.

3. The phases of the systems engineering process are shown in Figure 2.2.

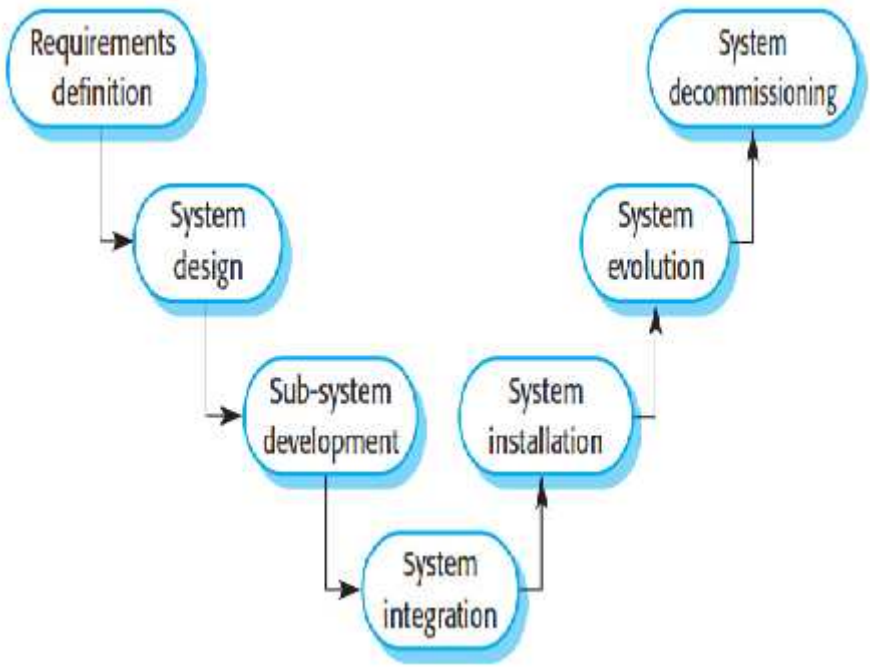


Figure 2.2 The systems engineering process

4. There are important distinctions between the system engineering process and the software development process:

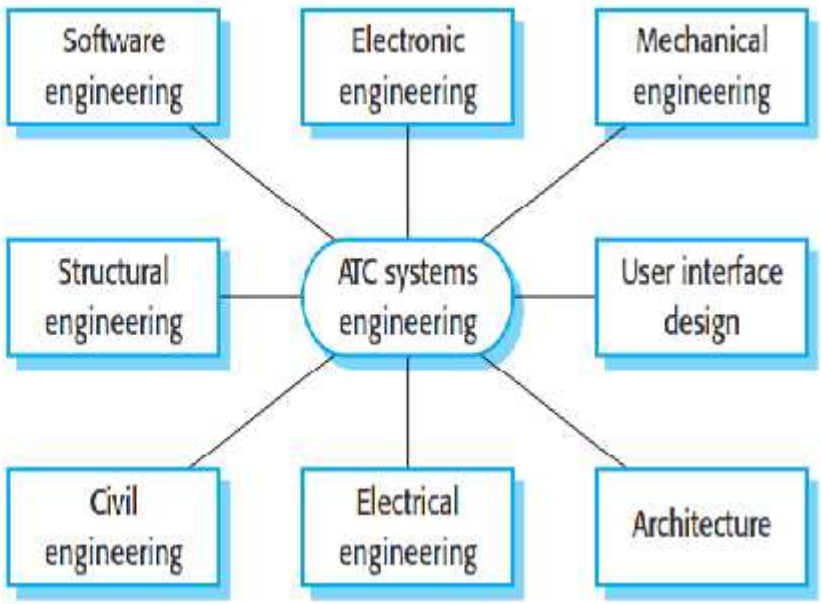
a).Limited scope for rework *during system development* Once some system engineering decisions, such as the siting of base stations in a mobile phone system, have been made, they are very expensive to change.

-One reason software has become so important in systems is that it allows changes to be made during system development, in response to new requirements.

b). *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.

5. Figure 2.3 shows some of the disciplines that may be involve in the system engineering team for an air traffic control (ATC) system that use radars and other sensors to determine aircraft position.

Figure 2.3 Disciplines involved in systems engineering



2.2.1 System requirements definition

1. System requirements definitions specify what the system should do (its functions) and its essential and desirable system properties. And involves consultations with system customers and end-users.

2. This requirements definition phase usually concentrates on deriving three types of requirement:

a) *Abstract functional requirements* The basic functions that the system must provide are defined at an abstract level.

-More detailed functional requirements specification takes place at the sub-system level.

For example, in an air traffic control system,

> An abstract functional requirement would specify that a flight-plan database should be used to store the flight plans of all aircraft entering the controlled airspace.

b) *System properties* These are non-functional emergent system properties such as availability, performance and safety. These nonfunctional system properties affect the requirements for all sub-systems.

c). *Characteristics that the system must not exhibit* It is sometimes as important to specify what the system must *not* do as it is to specify what the system should do.

For example, if you are specifying an air traffic control system, you might specify that the system should not present the controller with too much information.

3. System objective- it is an important part of the requirements definition phase. It is to establish a set of overall objectives that the system should meet.

a)-Here we define why the system is being procured for a particular environment.

To illustrate what this means, say you are specifying a system for an office building to provide for fire protection and for intruder detection.

Functional objective*:To provide a fire and intruder alarm system for the building that will provide internal and external warning of fire or unauthorised intrusion.*

Organisational objectives :*To ensure that the normal functioning of the work carried out in the building is not seriously disrupted by events such as fire and unauthorised intrusion*

4. System requirement problems:

a) complex systems are usually developed to address wicked problems(A 'wicked problem' is a problem that is so complex and where there are so many related entities that there is no definitive problem specification.)

-here the problems are not fully understood

-keep on changing as the system is being specified.

b) Hard to define non-functional requirements without knowing the component structure of system.

(An extreme example of a 'wicked problem' is earthquake planning. No one can accurately predict where the epicentre of an earthquake will be, what time it will occur or what effect it will have on the local environment. We cannot therefore

completely specify how to deal with a major earthquake. The problem can only be tackled after it has happened.)

2.2.2 System design

1. System design (Figure 2.4) is concerned with how the system functionality is to be provided by the components of the system.

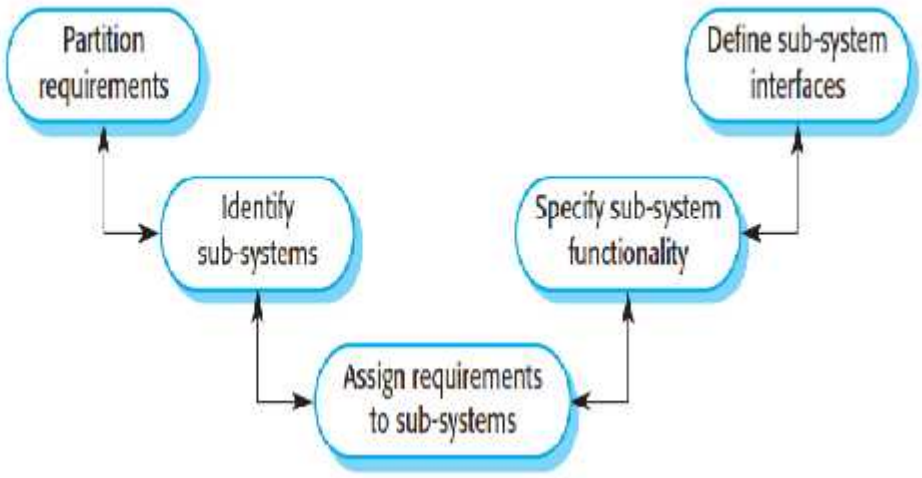


Figure 2.4 The system design process

2. The activities involved in this process are:

a) *Partition requirements:* Here the requirements are analysed and organized them into related groups.

b) *Identify sub-systems* :identify a set of sub-systems that can individually or collectively meet the requirements. Groups of requirements are usually related to sub-systems. sub-system identification may also be influenced by other organizational or environmental factors.

c). *Assign requirements to sub-systems* :assign the requirements to subsystems.

d). *Specify sub-system functionality* -specify the specific functions provided by each sub-system.

e).Define sub-system interfaces: You define the interfaces that are provided and required by each sub-system. Once these interfaces have been agreed upon, it becomes possible to develop these sub-systems in parallel.

As the double-ended arrows in Figure 2.4 imply, there is a lot of feedback and iteration from one stage to another in this design process.

Spiral Model

1-As the design process continues, problems with existing requirements and new requirements may emerge.

2-These linked processes is shown as a spiral, as shown in Figure 2.5

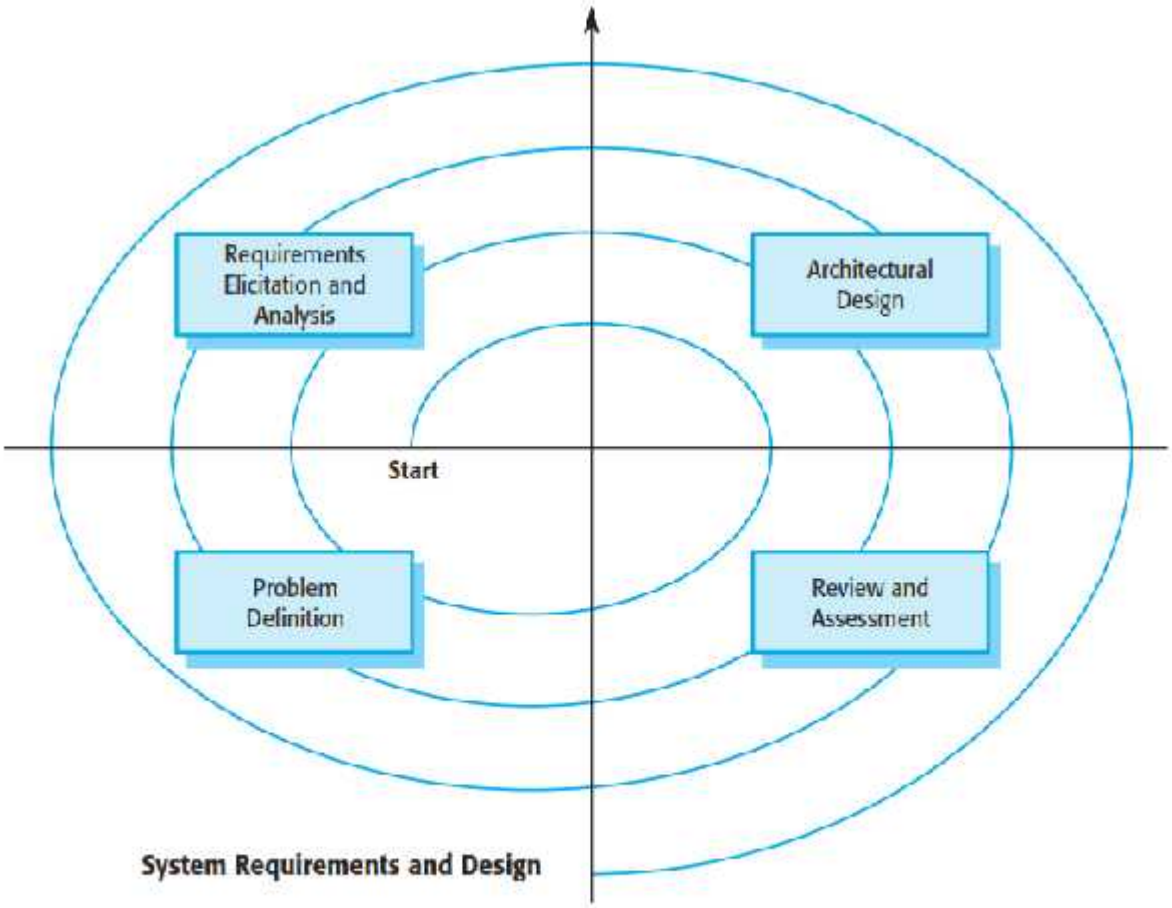


Figure 2.5 A spiral model of requirements and design

3-Starting in the centre, each round of the spiral may add detail to the requirements and the design. And focus on new knowledge collected during the requirements and design process .

2.2.3 System modelling

1. During the system requirements and design activity, systems may be modelled as a set of components and relationships between these components.

2. These are normally illustrated graphically in a system architecture model that gives overview of the system organisation.

3. The system architecture may be presented as a block diagram showing the major sub-systems and the interconnections between these sub-systems.

4. For example, Figure 2.6 shows the decomposition of an intruder alarm system into its principal components. The block diagram should be supplemented by brief descriptions of each sub-system, as shown in Figure 2.7.

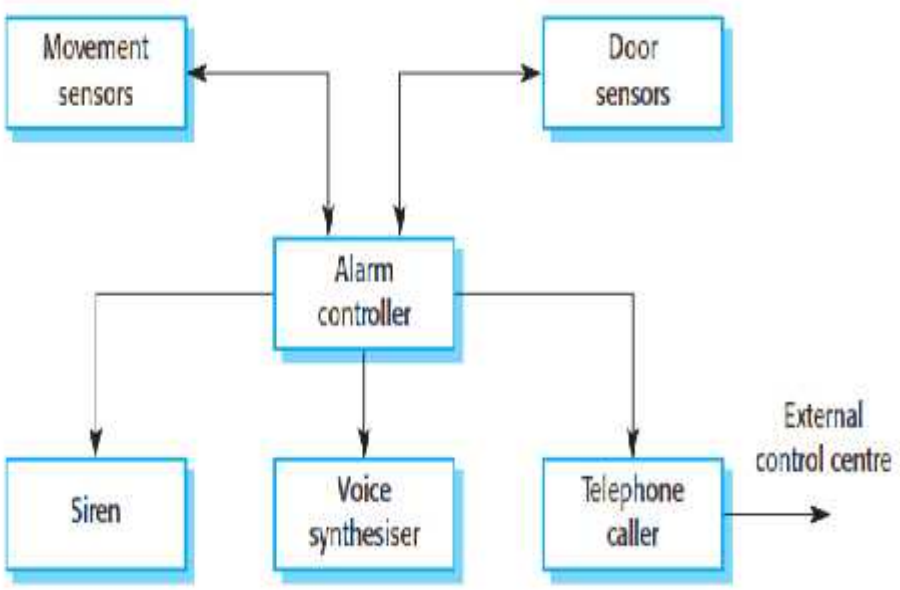


Figure 2.6 A simple burglar alarm system

Figure 2.7 Sub-system descriptions in the burglar alarm system

Sub-system	Description
Movement sensors	Detects movement in the rooms monitored by the system
Door sensors	Detects door opening in the external doors of the building
Alarm controller	Controls the operation of the system
Siren	Emits an audible warning when an intruder is suspected
Voice synthesiser	Synthesises a voice message giving the location of the suspected intruder
Telephone caller	Makes external calls to notify security, the police, etc.

5. Here system has been decomposed into functional components. Functional components are components that, when viewed from the perspective of the sub-system

Figure 2.8 shows the architecture of a much larger system for air traffic control. Several major sub-systems shown are themselves large systems. The arrowed lines that link these systems show information flow between these sub-systems.

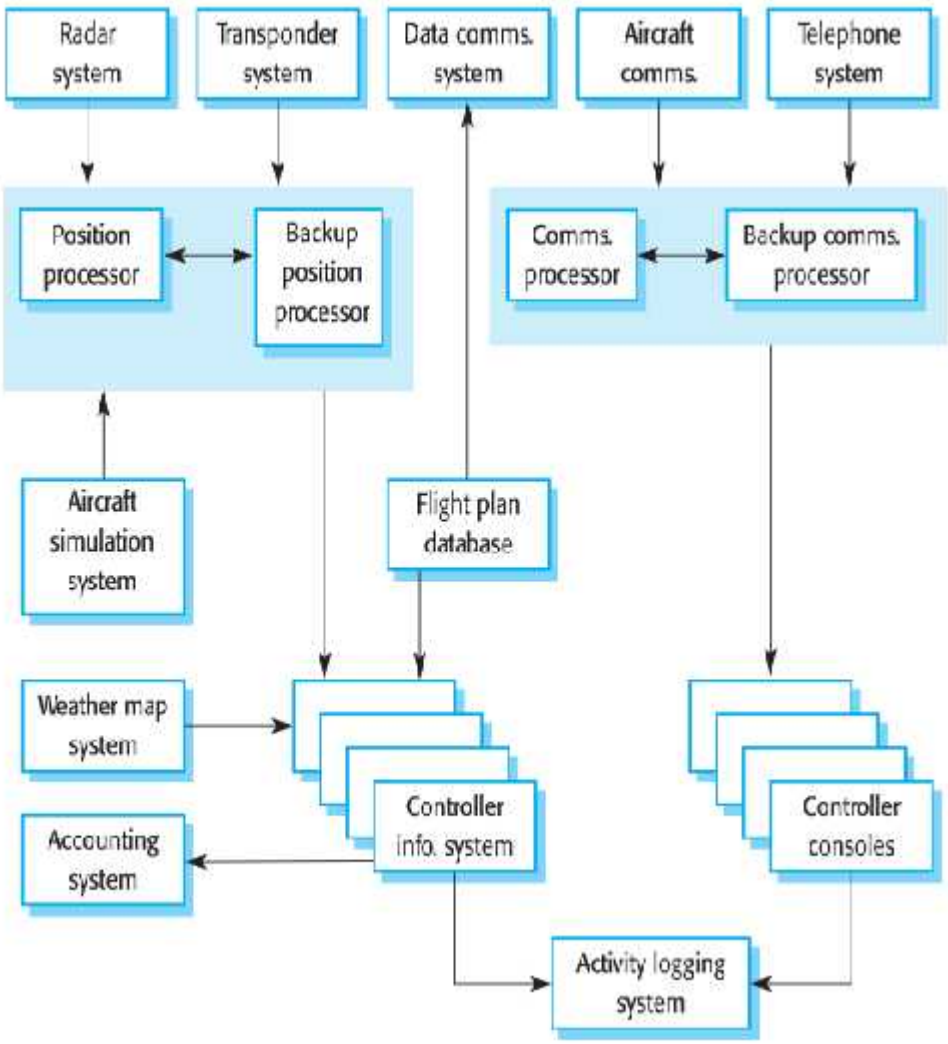


Figure 2.8 An architectural model of an air traffic control system

2.2.4 Sub-system development

1. During sub-system development, the sub-systems identified during system design are implemented.

2. This involve starting another system engineering process for individual sub-systems or, if the sub-system is software, a software process involving requirements, design, implementation and testing.

3. sub-systems can be developed from scratch during the development process. Normally, COTS sub-systems (commercial, off-the-shelf (COTS) systems) are bought for integration into the system. It is cheaper to buy existing products than to develop special-purpose components.

4. At this stage, the design activity phase is re-entered to accommodate a bought in

Component(COTS component).

5. Because COTS systems may not meet the requirements exactly .therefore slight modifications are made for design phase.

6. Sub-systems are usually developed in parallel.

Problems involved

1.Lack of communication across implementation teams.

2.Any system changes proposal leads to extension of development because of need for rework.

2.2.5 Systems integration

During the systems integration process, you take the independently developed subsystems and put them together to make up a complete system.

1. Integration can be done using a 'big bang' approach,

where all the sub-systems are integrated at the same time.

2. Integration can be done using' incremental integration' approach,

Where sub-systems are integrated one at a time.

It is the best approach, for two reasons:

a). It is usually impossible to schedule the development of all the sub-systems so that they are all finished at the same time.

b). Incremental integration reduces the cost of error location. If many sub-systems are simultaneously integrated, an error that arises during testing may be in any of these sub-systems.

When a single sub-system is integrated with an already working system, errors that occur are probably in the newly integrated sub-system or In the interactions between the existing subsystems and the new sub-system.

3. Once the components have been integrated, an extensive programme of system testing takes place. This testing aimed at testing the interfaces between components and the behaviour of the system as a whole.

2.2.6 System evolution

1. Large, complex systems have a very long lifetime. During their life, they are changed to correct errors in the original system requirements and to implement new requirements that have emerged.

2. System evolution, like software evolution is inherently costly for several reasons:

a) Proposed changes have to be analysed very carefully from a business and a technical perspective. Changes have to contribute to the goals of the system and should not simply be technically motivated.

b). Because sub-systems are never completely independent, changes to one subsystem may adversely affect the performance or behaviour of other subsystems. Consequent changes to these sub-systems may therefore be needed.

c). As systems age, their structure typically becomes corrupted by change so the costs of making further changes increases.

2.2.7 System decommissioning

1. System decommissioning means taking the system out of service after the end of its useful operational lifetime.

For hardware systems this may involve disassembling and recycling materials or dealing with toxic substances.

2. Software has no physical decommissioning problems, but some software may be incorporated in a system to assist with the decommissioning process.

For example, software may be used to monitor the state of hardware components. When the system is decommissioned, components that are not worn can therefore be identified and reused in other systems.

2.3 Organisations, people and computer systems

1. Socio-technical systems are enterprise systems that are intended to help deliver some organisational or business goal.

2. This might be to increase sales, reduce material used in manufacturing, collect taxes, maintain a safe airspace, etc influenced by the organisation's policies and procedures and by its working culture.

3. The users of the system are people who are influenced by the way the organisation is managed and by their interactions with other people inside and outside of the organisation.

4. Human and organisational factors from the system's environment that affect the system design include:

a) ***Process changes*** Does the system require changes to the work processes in the environment? If so, training will certainly be required. If changes are significant, or if they involve people losing their jobs, there is a danger that the users will resist the introduction of the system.

b). ***Job changes*** Does the system de-skill the users in an environment or cause them to change the way they work? If so, they may actively resist the introduction of the system into the organisation. Designs that involve managers having to change their way of working to fit the computer system are often resented. The managers may feel that their status in the organisation is being reduced by the system.

c). ***Organisational changes*** Does the system change the political power structure in an organisation? For example, if an organisation is dependent on a complex system, those who know how to operate the system have a great deal of political power.

2.3.1 Organisational processes

However, the development process is not the only process involved in systems engineering. It interacts with the system procurement process and with the process of using and operating the system. This is illustrated in Figure 2.9.

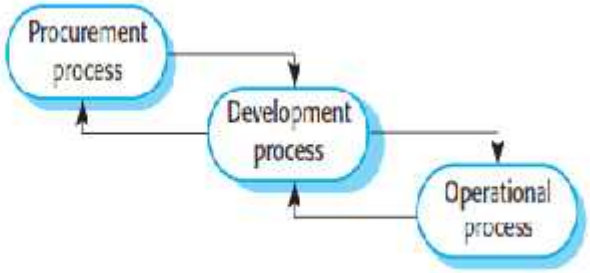


Figure 2.9 Procurement, development and operational processes

procurement process It is normally embedded within the organisation that will buy and use the system (the client organisation).

-The process of system procurement is concerned with making decisions about the best way for an organisation to acquire a system and deciding on the best suppliers of that system.

Figure 2.10 shows the procurement process for both existing systems and systems that have to be specially designed.

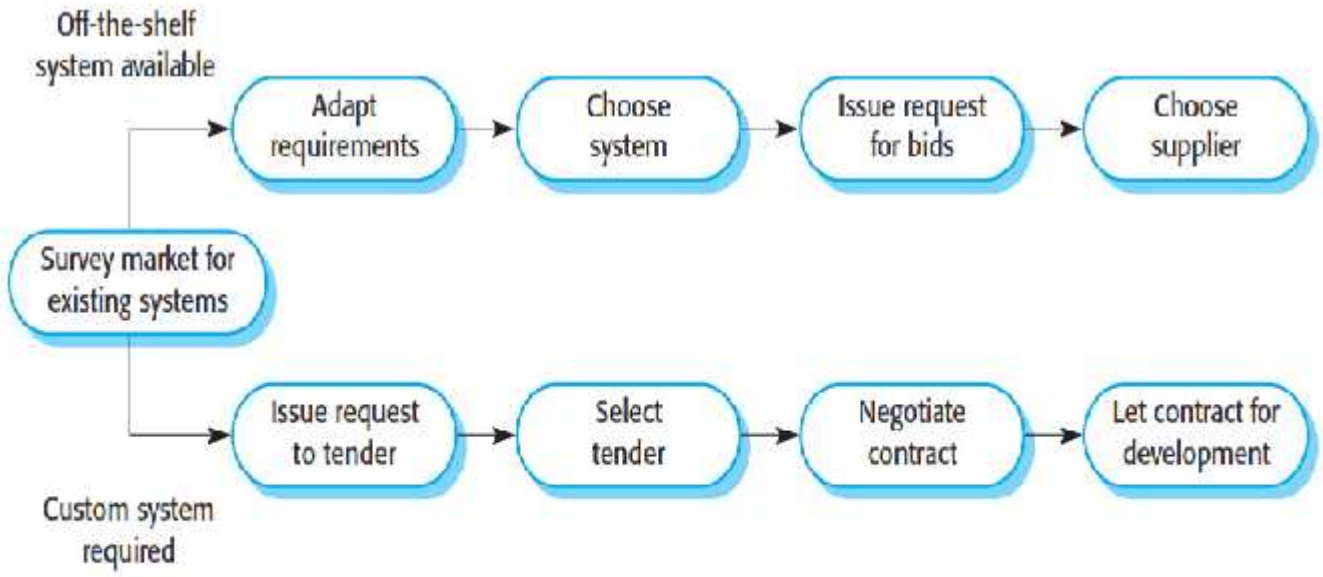


Figure 2.10 The system procurement process

Some important points about the process shown in this diagram are:

1. Off-the-shelf components do not usually match requirements exactly. Therefore, choosing a system means finding the closest match between the system requirements and the facilities offered by off-the-shelf systems, then modify the requirements and this can have knock-on effects on other sub-systems.

2. When a system is to be built specially, the specification of requirements acts as the basis of a contract for the system procurement. It is therefore a legal, as well as a technical, document.

3. After a contractor to build a system has been selected, there is a contract negotiation period where negotiation to further changes to the requirements and issues such as the cost of changes to the system are discussed.

Development process Complex systems are usually developed by a different organization (the supplier) from the organization that is procuring the system(organization buy the sub-system from different supplier organization. The reason for this is that the procurer's business is rarely system development so its employees do not have the skills needed to develop complex systems themselves).

1-This supplier, who is usually called the *principal contractor*, may contract out the development of different sub-systems to a number of sub-contractors.

2-For large systems, such as air traffic control systems, a group of suppliers may form a consortium to bid for the contract.

3- The consortium should include all of the capabilities required for this type of system, such as computer hardware suppliers, software developers, peripheral suppliers and suppliers of specialist equipment .

4-The procurer deals with the contractor rather than the sub-contractors so that there is a single procurer/supplier interface.

5-The sub-contractors design and build parts of the system to a specification that is produced by the principal contractor. Once completed, the principal contractor integrates these different components and delivers them to the customer buying the system.

6-Depending on the contract, the procurer may allow the principal contractor a free choice of sub-contractors or may require the principal contractor to choose sub-contractors from an approved list.

Operational processes are the processes that are involved in using the system for its defined purpose. For example, operators of an air traffic control system follow specific processes when aircraft enter and leave airspace so on

1-For new systems, these operational processes have to be defined and documented during the system development process.

2-Operators may have to be trained and other work processes adapted to make effective use of the new system.

3-The key benefit of having people in a system is that people have a unique capability of being able to respond effectively to unexpected situations even when they have never had direct experience of these situations.

4-Operators also use their local knowledge to adapt and improve processes.

5-designers should design operational processes to be flexible and adaptable. The operational processes should not be too constraining, they should not require

operations to be done in a particular order. So that they can respond for operational errors effectively.

2.4 Legacy systems

Because of the time and effort required to develop a complex system, large computerbased systems usually have a long lifetime.

-For example, military systems are often designed for a 20-year lifetime.

It is sometimes too expensive and too risky to discard such business critical systems after a few years of use. Their development continues throughout their life with changes to accommodate new requirements, new operating platforms, and so forth.

Definition: Legacy systems are socio-technical computer-based systems that have been developed in the past, often using older or obsolete technology.

1. These systems include not only hardware and software but also legacy processes and procedures—old ways of doing things that are difficult to change because they rely on legacy software.

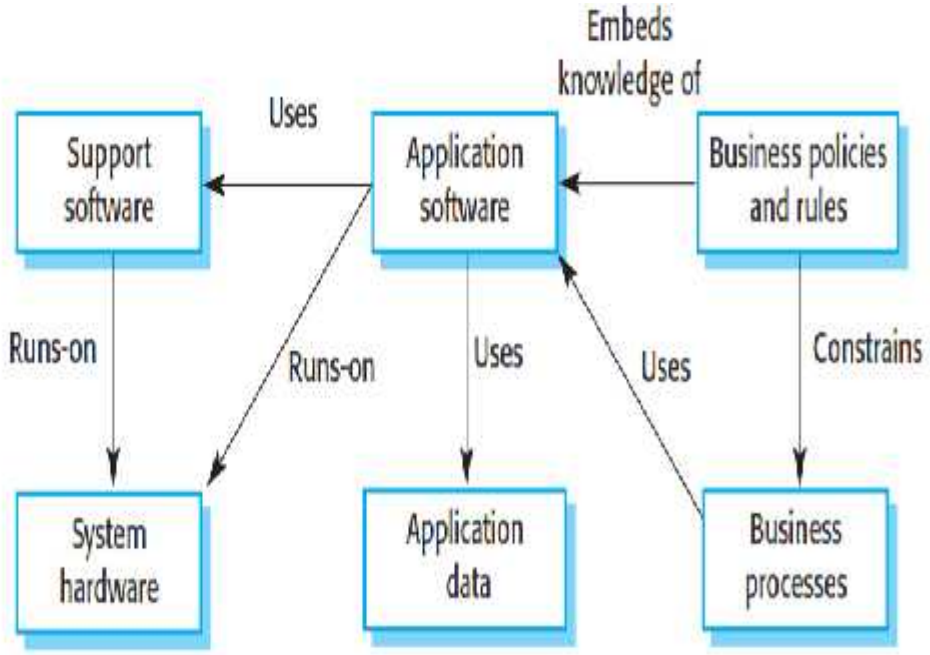
2. Changes to one part of the system inevitably involve changes to other components,

3. Legacy systems are often business-critical systems. They are maintained because it is too risky to replace them.

For example, for most banks the customer accounting system was one of their earliest systems.

4. Figure 2.11 illustrates the logical parts of a legacy system and their relationships:

Figure 2.11 Legacy system components



a). *System hardware* In many cases, legacy systems have been written for mainframe hardware that is no longer available, that is expensive to maintain and that may not be compatible with current organisational IT purchasing policies.

b). Support software The legacy system may rely on a range of support software from the operating system and utilities provided by the hardware manufacturer through to the compilers used for system development. Again, these may be obsolete and no longer supported by their original providers.

c). *Application software* The application system that provides the business services is usually composed of a number of separate programs that have been developed at different times. Sometimes the term *legacy system* means this application software system rather than the entire system.

d). *Application data* These are the data that are processed by the application system. This data may be inconsistent and may be duplicated in several files.

e). *Business processes* These are processes that are used in the business to achieve some business objective. Business processes may be designed around a legacy system and constrained by the functionality that it provides.

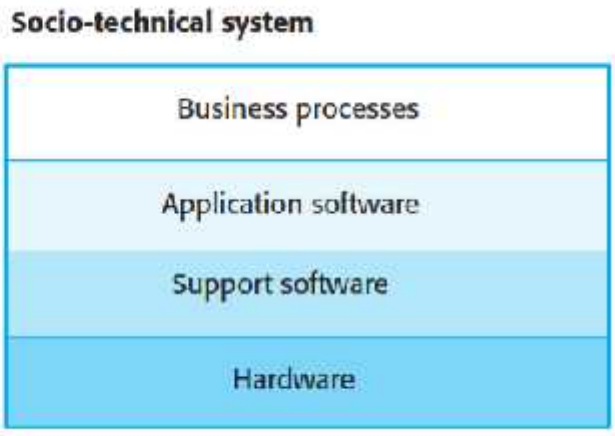
An example of a business process in an insurance company would be issuing an insurance policy.

f). *Business policies and rules* These are definitions of how the business should be carried out and constraints on the business. Use of the legacy applicationsystem may be embedded in these policies and rules.

5. An alternative way of looking at these components of a legacy system is as a series of layers, as shown in Figure 2.12.

a) Each layer depends on the layer immediately below it and interfaces with that layer. If interfaces are maintained, then you should be able to make changes within a layer without affecting either of the adjacent layers.

Figure 2.12 Layered model of a legacy system



b) Here changes to one layer of the system may require consequent changes to layers that are both above and below the changed level. The reasons for this are:

1. Changing one layer in the system may introduce new facilities, and higher layers in the system may then be changed to take advantage of these facilities.

For example, a new database introduced at the support software layer may include facilities to access the data through a web browser, and business processes may be modified to take advantage of this facility.

2.Changing the software may slow the system down so that new hardware is needed to improve the system performance. The increase in performance from the new hardware may then mean that further software changes which were previously impractical become possible.

3.It is often impossible to maintain hardware interfaces, especially if a radical change to a new type of hardware is proposed.

For example, if a company moves from mainframe hardware to client-server systems these usually have different operating systems. Major changes to the application software may therefore be required.