UNIT-II

21CSC303J

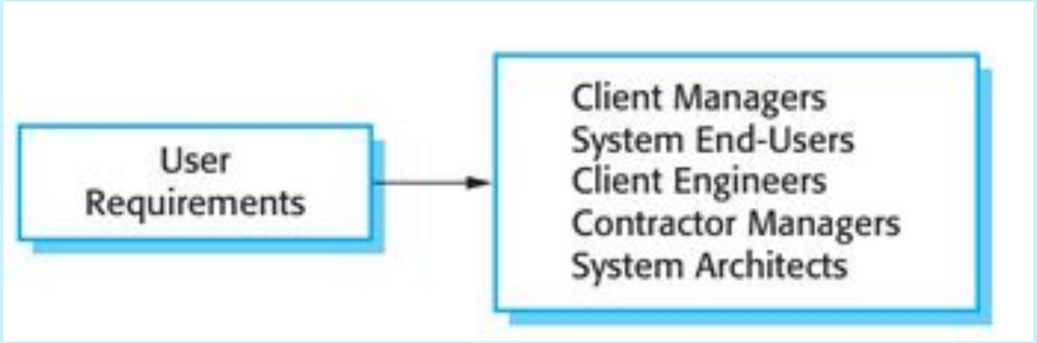
SOFTWARE ENGINEERING AND PROJECT MANAGEMENT

Unit II- Syllabus

- ➔ Software Requirements: Functional and non-functional requirements,
- ➔ User requirements,
- ➔ system requirements,
- ➔ The software requirements document.
- ➔ Requirements engineering process: Feasibility studies,
- ➔ Requirements elicitation and analysis,
- ➔ Requirements validation,
- ➔ Requirements management,
- ➔ Software project effort and cost estimation –
  - ◆ Cocomo model I,
  - ◆ Cocomo Model II,
  - ◆ LOC,
  - ◆ Function point metrics

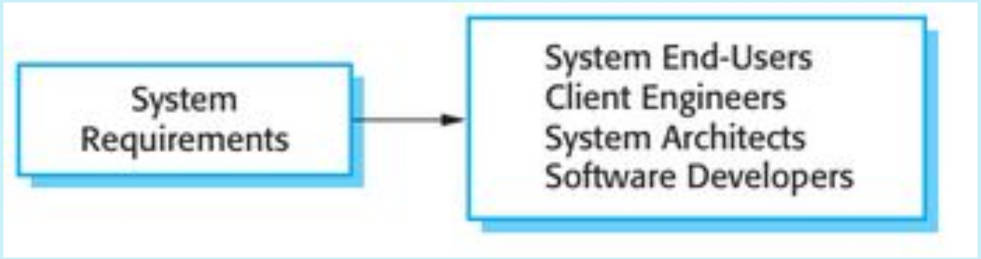
User requirements

- User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.



System requirements

- System requirements are more detailed descriptions of the software system’s functions, services, and operational constraints.
- The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented.
- It may be part of the contract between the system buyer and the software developers.
- The system requirements provide more specific information about the services and functions of the system that is to be implemented.



Functional and Non-functional requirements

Software system requirements are often classified as functional requirements or non-functional requirements:

- **1. Functional requirements** These are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

- **2. Non-functional requirements** These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, and constraints imposed by standards. Non-functional requirements often apply to the system as a whole, rather than individual system features or services.

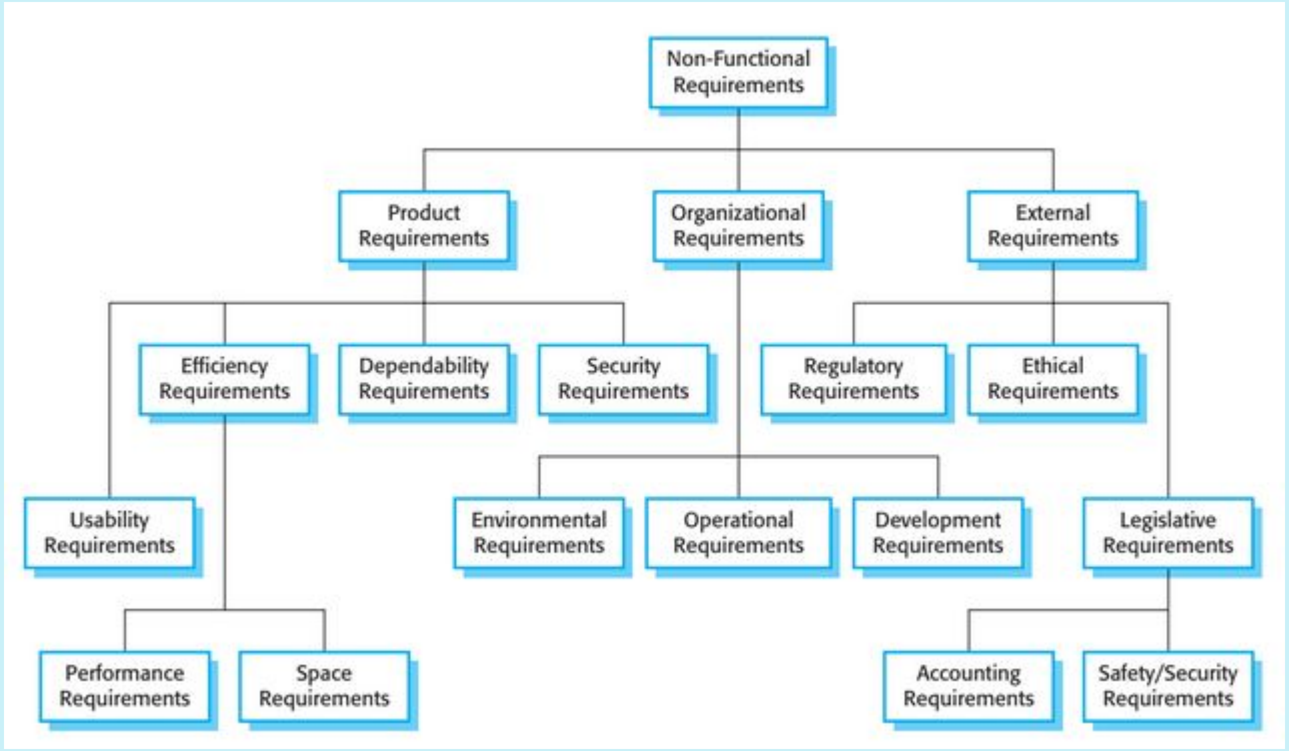
Functional requirements

- The functional requirements for a system describe what the system should do.
- These requirements depend on the type of software being developed, the expected users of the software, and the general approach taken by the organization when writing requirements.
- When expressed as user requirements, functional requirements are usually described in an abstract way that can be understood by system users.
- However, more specific functional system requirements describe the system functions, its inputs and outputs, exceptions, etc., in detail.
- Functional system requirements vary from general requirements covering what the system should do to very specific requirements reflecting local ways of working or an organization’s existing systems.

Non-functional requirements

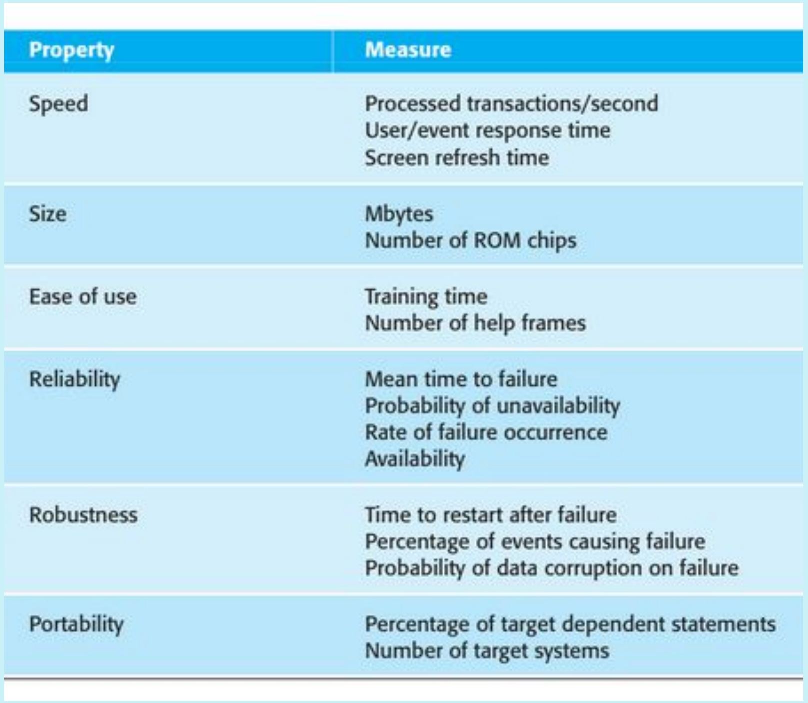
- Non-functional requirements, as the name suggests, are requirements that are not directly concerned with the specific services delivered by the system to its users.
- They may relate to emergent system properties such as reliability, response time, and store occupancy.
- Alternatively, they may define constraints on the system implementation such as the capabilities of I/O devices or the data representations used in interfaces with other systems.
- Non-functional requirements, such as performance, security, or availability, usually specify or constrain characteristics of the system as a whole.
- Non-functional requirements are often more critical than individual functional requirements. System users can usually find ways to work around a system function that doesn’t really meet their needs.
- However, failing to meet a non-functional requirement can mean that the whole system is unusable.
- **Product requirement** These requirements specify or constrain the behavior of the software.
- **Organizational requirements** These requirements derived from policies and procedures in the customer’s and developer’s organization.
- **External requirements** This covers all requirements that are derived from factors external to the system and its development process.

Non-functional requirements



Non-functional requirements

- Metrics used to specify non-functional system properties.



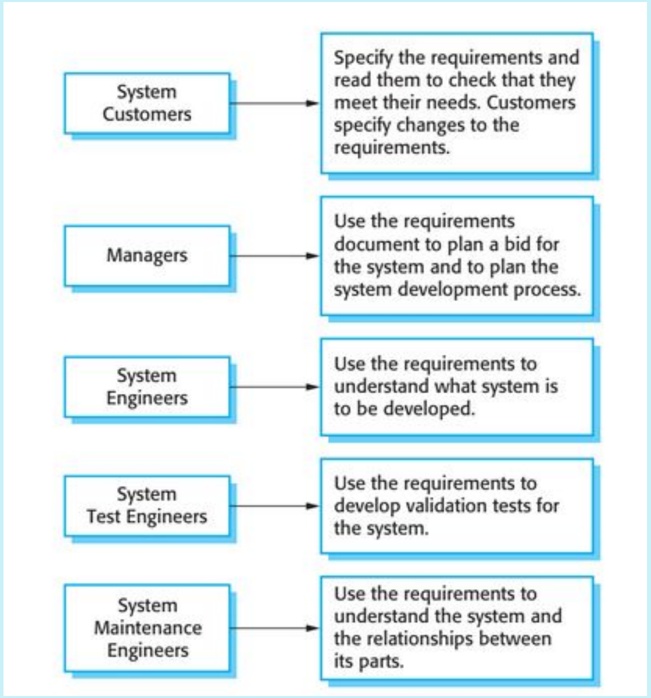
Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

The software requirements document.

- The software requirements document (sometimes called the software requirements specification or SRS) is an official statement of what the system developers should implement.
- It should include both the user requirements for a system and a detailed specification of the system requirements.
- Sometimes, the user and system requirements are integrated into a single description.
- In other cases, the user requirements are defined in an introduction to the system requirements specification.
- If there are a large number of requirements, the detailed system requirements may be presented in a separate document.
- Requirements documents are essential when an outside contractor is developing the software system.
- However, agile development methods argue that requirements change so rapidly that a requirements document is out of date as soon as it is written, so the effort is largely wasted.
- Rather than a formal document, approaches such as Extreme Programming (Beck, 1999) collect user requirements incrementally and write these on cards as user stories.
- The user then prioritizes requirements for implementation in the next increment of the system.

The software requirements document.

- The requirements document has a diverse set of users, ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software.


System Customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System Engineers
Use the requirements to understand what system is to be developed.
System Test Engineers
Use the requirements to develop validation tests for the system.
System Maintenance Engineers
Use the requirements to understand the system and the relationships between its parts.

The software requirements document.

- The diversity of possible users means that the requirements document has to be a compromise between communicating the requirements to customers, defining the requirements in precise detail for developers and testers, and including information about possible system evolution.
- Information on anticipated changes can help system designers avoid restrictive design decisions and help system maintenance engineers who have to adapt the system to new requirements.
- The level of detail that you should include in a requirements document depends on the type of system that is being developed and the development process used.
- Critical systems need to have detailed requirements because safety and security have to be analyzed in detail.
- When the system is to be developed by a separate company, the system specifications need to be detailed and precise.
- If an inhouse, iterative development process is used, the requirements document can be much less detailed and any ambiguities can be resolved during development of the system.

Requirements engineering processes

- Requirements engineering processes may include four high-level activities.
- These focus on
  - Assessing if the system is useful to the business (feasibility study),
  - Discovering requirements (elicitation and analysis),
  - Converting these requirements into some standard form (specification), and
  - Checking that the requirements actually define the system that the customer wants (validation).
- Requirements engineering is an iterative process in which the activities are interleaved.
- The activities are organized as an iterative process around a spiral, with the output being a system requirements document.
- The amount of time and effort devoted to each activity in each iteration depends on the stage of the overall process and the type of system being developed.
- Early in the process, most effort will be spent on understanding high-level business and non-functional requirements, and the user requirements for the system.
- Later in the process, in the outer rings of the spiral, more effort will be devoted to eliciting and understanding the detailed system requirements.

Requirements engineering processes

- This spiral model accommodates approaches to development where the requirements are developed to different levels of detail.
- The number of iterations around the spiral can vary so the spiral can be exited after some or all of the user requirements have been elicited.
- Agile development can be used instead of prototyping so that the requirements and the system implementation are developed together.
- Some people consider requirements engineering to be the process of applying a structured analysis method, such as object-oriented analysis (Larman, 2002).
- This involves analyzing the system and developing a set of graphical system models, such as use case models, which then serve as a system specification.
- The set of models describes the behavior of the system and is annotated with additional information describing, for example, the system’s required performance or reliability.
- Although structured methods have a role to play in the requirements engineering process, there is much more to requirements engineering than is covered by these methods.
- Requirements elicitation, in particular, is a human-centered activity and people dislike the constraints imposed on it by rigid system models.

Requirements engineering processes

- In virtually all systems, requirements change.
- The people involved develop a better understanding of what they want the software to do; the organization buying the system changes; modifications are made to the system's hardware, software, and organizational environment.

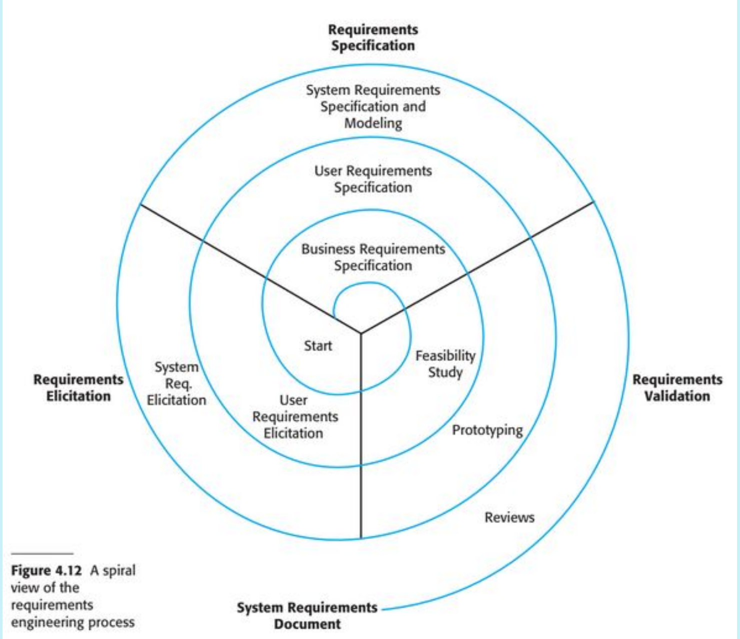

Requirements Specification
System Requirements Specification and Modeling
User Requirements Specification
Business Requirements Specification
Start
Feasibility Study
Prototyping
Reviews
Requirements Elicitation
System Req. Elicitation
User Requirements Elicitation
Requirements Validation
System Requirements Document

Figure 4.12 A spiral view of the requirements engineering process

Requirements elicitation and analysis

- After an initial feasibility study, the next stage of the requirements engineering process is requirements elicitation and analysis.
- In this activity, software engineers work with customers and system end-users to find out about the application domain,what services the system should provide, the required performance of the system, hardware constraints, and so on.
- Requirements elicitation and analysis may involve a variety of different kinds of people in an organization.
- A system stakeholder is anyone who should have some direct or indirect influence on the system requirements.
- Stakeholders include end users who will interact with the system and anyone else in an organization who will be affected by it.
- Other system stakeholders might be engineers who are developing or maintaining other related systems, business managers, domain experts, and trade union representatives.
- Each organization will have its own version or instantiation of this general model depending on local factors such as the expertise of the staff, the type of system being developed, the standards used, etc.

Requirements elicitation and analysis

The process activities are:

Requirements discovery

- This is the process of interacting with stakeholders of the system to discover their requirements.
- Domain requirements from stakeholders and documentation are also discovered during this activity.

Requirements classification and organization

- This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters.
- The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associate requirements with each sub-system.
- Requirements engineering and architectural design cannot be completely separate activities.

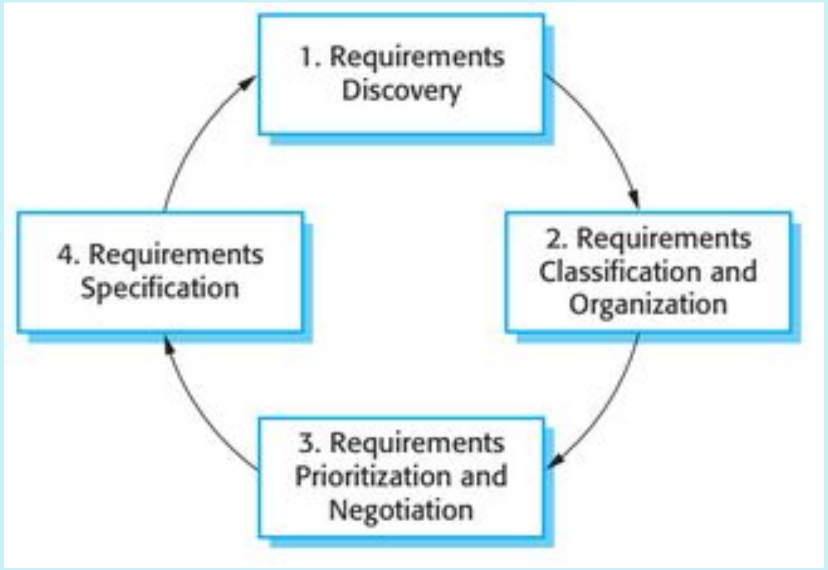
Requirements prioritization and negotiation

- Inevitably, when multiple stakeholders are involved, requirements will conflict.
- This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.
- Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.

Requirements elicitation and analysis

Requirements specification

- The requirements are documented and input into the next round of the spiral.


1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

Requirements elicitation and analysis

- The process cycle starts with requirements discovery and ends with the requirements documentation.
- The analyst’s understanding of the requirements improves with each round of the cycle.
- The cycle ends when the requirements document is complete.
- Eliciting and understanding requirements from system stakeholders is a difficult process for several reasons:
    - Stakeholders often don’t know what they want from a computer system except in the most general terms; they may find it difficult to articulate what they want the system to do; they may make unrealistic demands because they don’t know what is and isn’t feasible.
    - Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer’s domain, may not understand these requirements.
    - Different stakeholders have different requirements and they may express these in different ways. Requirements engineers have to discover all potential sources of requirements and discover commonalities and conflict.
    -

Requirements elicitation and analysis

  - Political factors may influence the requirements of a system. Managers may demand specific system requirements because these will allow them to increase their influence in the organization.
  - The economic and business environment in which the analysis takes place is dynamic. It inevitably changes during the analysis process. The importance of particular requirements may change. New requirements may emerge from stakeholders who were not originally consulted.
- Inevitably, different stakeholders have different views on the importance and priority of requirements and, sometimes, these views are conflicting.
- During the process, organize regular stakeholder negotiations so that compromises can be reached.
- It is impossible to completely satisfy every stakeholder but if some stakeholders feel that their views have not been properly considered then they may deliberately attempt to undermine the RE process.
- At the requirements specification stage, the requirements that have been elicited so far are documented in such a way that they can be used to help with requirements discovery.
- At this stage, an early version of the system requirements document may be produced with missing sections and incomplete requirements.
- Alternatively, the requirements may be documented in a completely different way.
- Writing requirements on cards can be very effective as these are easy for stakeholders to handle, change, and organize.

Requirements validation,

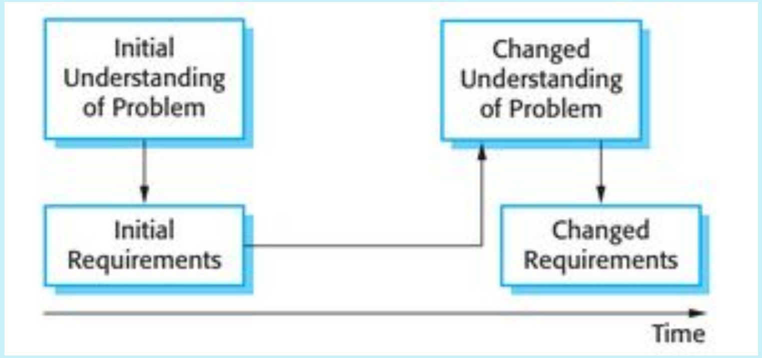
- Requirements validation is checking that requirements actually define the system that the customer really wants.
- It overlaps with analysis as it is concerned with finding problems with the requirements.
- Requirements validation is important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service.
- The cost of fixing a requirements problem by making a system change is usually much greater than repairing design or coding errors.
- The reason for this is that a change to the requirements usually means that the system design and implementation must also be changed. Furthermore the system must then be re-tested.
- During the requirements validation process, different types of checks should be carried out on the requirements in the requirements document.
- These checks include:

Validity checks

- A user may think that a system is needed to perform certain functions. However, further thought and analysis may identify additional or different functions that are required. Systems have diverse stakeholders with different needs and any set of requirements is inevitably a compromise across the stakeholder community.

Requirements validation,

Consistency checks : Requirements in the document should not conflict. That is, there should not be contradictory constraints or different descriptions of the same system function.
Completeness checks : The requirements document should include requirements that define all functions and the constraints intended by the system user.
Realism checks : Using knowledge of existing technology, the requirements should be checked to ensure that they can actually be implemented. These checks should also take account of the budget and schedule for the system development.
Verifiability : To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable. This means that you should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement.


Initial Understanding of Problem
Changed Understanding of Problem
Initial Requirements
Changed Requirements
Time

Requirements validation,

- Requirements validation techniques that can be used individually or in conjunction with one another:
    - **Requirements reviews** The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.
    - **Prototyping** In this approach to validation, an executable model of the system in question is demonstrated to end-users and customers. They can experiment with this model to see if it meets their real needs.
    - **Test-case generation** Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered. Developing tests from the user requirements before any code is written is an integral part of extreme programming.
- Ultimately, it is difficult to show that a set of requirements does in fact meet a user's needs.
- Users need to picture the system in operation and imagine how that system would fit into their work.
- It is hard even for skilled professionals to perform this type of abstract analysis and harder still for system users.
- As a result, rarely find all requirements problems during the requirements validation process.
- It is inevitable that there will be further requirements changes to correct omissions and misunderstandings after the requirements document has been agreed upon.

Requirements management

- The requirements for large software systems are always changing.
- One reason for this is that these systems are usually developed to address ‘wicked’ problems—problems that cannot be completely defined.
- Because the problem cannot be fully defined, the software requirements are bound to be incomplete.
- During the software process, the stakeholders’ understanding of the problem is constantly changing.
- The system requirements must then also evolve to reflect this changed problem view.
- Once a system has been installed and is regularly used, new requirements inevitably emerge.
- It is hard for users and system customers to anticipate what effects the new system will have on their business processes and the way that work is done.
- Once end- users have experience of a system, they will discover new needs and priorities.
- There are several reasons why change is inevitable:
    - 1. The business and technical environment of the system always changes after installation. New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.

Requirements management

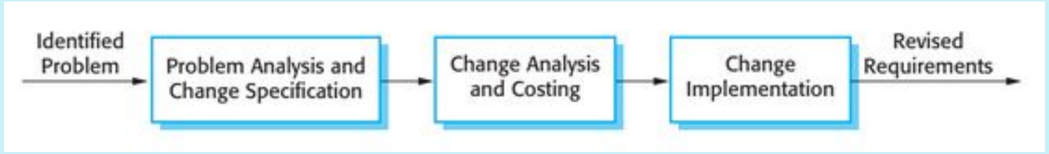
- ○ 2. The people who pay for a system and the users of that system are rarely the same people. System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.
- ○ 3. Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory. The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

- Requirements management is the process of understanding and controlling changes to system requirements.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes.
- You need to establish a formal process for making change proposals and linking these to system requirements.
- The formal process of requirements management should start as soon as a draft version of the requirements document is available.
- However, you should start planning how to manage changing requirements during the requirements elicitation process.

Requirements management,

Requirements management planning

- Planning is an essential first stage in the requirements management process.
- The planning stage establishes the level of requirements management detail that is required.
- During the requirements management stage, you have to decide on:
  - **Requirements identification** Each requirement must be uniquely identified so that it can be cross-referenced with other requirements and used in traceability assessments.
  - **A change management process** This is the set of activities that assess the impact and cost of changes.
  - **Traceability policies** These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.The traceability policy should also define how these records should be maintained.
  - **Tool support** Requirements management involves the processing of large amounts of information about the requirements. Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.


Identified Problem
Problem Analysis and Change Specification
Change Analysis and Costing
Change Implementation
Revised Requirements

Requirements management,

- Requirements management needs automated support and the software tools for this should be chosen during the planning phase. You need tool support for:
    - 1. Requirements storage The requirements should be maintained in a secure, managed data store that is accessible to everyone involved in the requirements engineering process.
    - 2. Change management The process of change management is simplified if active tool support is available.
    - 3. Traceability management As discussed above, tool support for traceability allows related requirements to be discovered. Some tools are available which use natural language processing techniques to help discover possible relationships between requirements.
- For small systems, it may not be necessary to use specialized requirements management tools.
- The requirements management process may be supported using the facilities available in word processors, spreadsheets, and PC databases.
- However, for larger systems, more specialized tool support is required.

Requirements management,

Requirements change management

- Requirements change management (Figure 4.18) should be applied to all proposed changes to a system's requirements after the requirements document has been approved.
- Change management is essential because you need to decide if the benefits of implementing new requirements are justified by the costs of implementation.
- The advantage of using a formal process for change management is that all change proposals are treated consistently and changes to the requirements document are made in a controlled way.
- There are three principal stages to a change management process:
  - **1.Problem analysis and change specification** The process starts with an identified requirements problem or, sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
  - **2. Change analysis and costing** The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated both in terms of modifications to the requirements document and, if appropriate, to the system design and implementation. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.

Requirements management,

Requirements change management

- ○ **3. Change implementation** The requirements document and, where necessary, the system design and implementation, are modified. You should organize the requirements document so that you can make changes to it without extensive rewriting or reorganization. As with programs, changeability in documents is achieved by minimizing external references and making the document sections as modular as possible. Thus, individual sections can be changed and replaced without affecting other parts of the document.

- If a new requirement has to be urgently implemented, there is always a temptation to change the system and then retrospectively modify the requirements document.
- Once system changes have been made, it is easy to forget to include these changes in the requirements document or to add information to the requirements document that is inconsistent with the implementation.
- Agile development processes (XP), have been designed to cope with requirements that change during the development process.
- In these processes, when a user proposes a requirements change, this change does not go through a formal change management process.
- Rather, the user has to prioritize that change and, if it is high priority, decide what system features that were planned for the next iteration should be dropped.

Lines of Code (LOC)

- LOC is possibly the simplest among all metrics available to measure project size.
- Consequently, this metric is extremely popular.
- This metric measures the size of a project by counting the number of source instructions in the developed program.
- Obviously, while counting the number of source instructions, comment lines, and header lines are ignored.
- Determining the LOC count at the end of a project is very simple.
- However, accurate estimation of LOC count at the beginning of a project is a very difficult task.
- One can possibly estimate the LOC count at the starting of a project, only by using some form of systematic guess work.
- The project manager divides the problem into modules, and each module into sub-modules and so on, until the LOC of the leaf-level modules are small enough to be predicted.
- To be able to predict the LOC count for the various leaf-level modules sufficiently accurately, past experience in developing similar modules is very helpful.
- By adding the estimates for all leaf level modules together, project managers arrive at the total size estimation.
- In spite of its conceptual simplicity, LOC metric has several shortcomings when used to measure problem size.

Lines of Code (LOC)

LOC is a measure of coding activity alone.

- A good problem size measure should consider the total effort needed to carry out various life cycle activities and not just the coding effort. LOC, however, focuses on the coding activity alone—it merely computes the number of source lines in the final program. The presumption that the total effort needed to develop a project is proportional to the coding effort is easily countered by noting the fact that even when the design or testing issues are very complex, the code size might be small and vice versa. Thus, the design and testing efforts can be grossly disproportionate to the coding effort. Code size, therefore, is obviously an improper indicator of the problem size.

LOC count depends on the choice of specific instructions:

- LOC gives a numerical value of problem size that can vary widely with coding styles of individual programmers. By coding style, we mean the choice of code layout, the choice of the instructions in writing the program, and the specific algorithms used. Different programmers may lay out their code in very different ways. Unless this issue is handled satisfactorily, there is a possibility of arriving at very different size measures for essentially identical programs. This problem can, to a large extent, be overcome by counting the language tokens in a program rather than the lines of code. However, a more intricate problem arises due to the specific choices of instructions made in writing the program. For example, one programmer may use a switch statement in writing a C program and another may use a sequence of if … then ... else ... statements. Therefore, the following can easily be concluded.

Lines of Code (LOC)

LOC measure correlates poorly with the quality and efficiency of the code:

- Larger code size does not necessarily imply better quality of code or higher efficiency. Some programmers produce lengthy and complicated code as they do not make effective use of the available instruction set or use improper algorithms. In fact, it is true that a piece of poor a n d sloppily written piece of code can have larger number of source instructions than a piece t h a t is efficient and has been thoughtfully written. Calculating productivity as LOC generated per man-month may encourage programmers to write lots of poor quality code rather than fewer lines of high quality code achieve the same functionality.

LOC metric penalises use of higher-level programming languages and code reuse:

- A paradox is that if a programmer consciously uses several library routines, then the LOC count will be lower. This would show up as smaller program size, and in turn, would indicate lower effort! Thus, if managers use the LOC count to measure the effort put in by different developers (that is, their productivity), they would be discouraging code reuse by developers. Modern programming methods such as object-oriented programming and reuse of components makes the relationships between LOC and other project attributes even less precise.

Lines of Code (LOC)

LOC metric measures the lexical complexity of a program and does not address the more important issues of logical and structural complexities:

- Between two programs with equal LOC counts, a program incorporating complex logic would require much more effort to develop than a program with very simple logic. To realise why this is so, imagine the effort that would be required to develop a program having multiple nested loops and decision constructs and compare that with another program having only sequential control flow.

It is very difficult to accurately estimate LOC of the final program from problem specification:

- As already discussed, at the project initiation time, it is a very difficult task to accurately estimate the number of lines of code (LOC) that the program would have after development. The LOC count
- can accurately be computed only after the code has fully been developed. Since project planning is carried out even before any development activity starts, the LOC metric is of little use to the project managers during project planning.

Software Project Effort and Cost Estimation

- Effort estimation for any software project as well as outsourced projects it is even more crucial.
- Effort estimate along with the schedule indicate to the customer what the cost impact will be and when the software can be realized.
- The management in customer organizations typically expects a lot from software projects.
- Software projects are seen as strategic tools to compete in the market.
- Therefore, a successful software implementation is regarded as a market edge and can influence the fortunes of that organization.
- Software projects are costly as software professionals are expensive to hire.
- The optimal usage of time of these high-salaried people requires careful project planning to minimize wastage of time of these high-cost resources.
- At the same time, the service provider should be able to bill its customer for the actual effort put forth in delivering the project so that neither the customer nor the service provider is at a loss for wrong billing in the costs involved.
- Therefore, an accurate effort and cost estimate is of paramount importance for software projects.

Software Project Effort and Cost Estimation

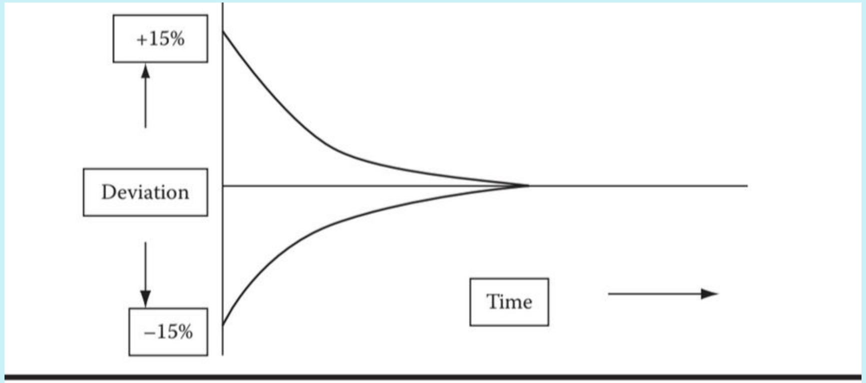
- With regard to effort for a software project, there are two aspects.
- One is to provide a good effort estimate and present it to the customer.
- The other aspect is to use it to form the project team based on the skills required for the project and the kind of budget that will be available for the project so that the right kind of people can be staffed for the project within the specified budget.
- Tight budgets and tight schedules are the general norm for most projects today and this makes good and reliable effort, schedule and cost estimates for projects even more important.

Effort Estimation Techniques

- Effort estimation is an evolutionary phenomenon.
- The beginning of any project sees an initial effort estimate which is rough and mostly inaccurate at best.
- The more the information available about the project, the more accurate will be the estimate.
- As more and more information becomes available for any project as it progresses, it makes sense to revise project estimate regularly to make the estimate more accurate.

Software Project Effort and Cost Estimation

- Statistical effort estimate techniques are extremely useful for effort estimation.
- Actual effort data from past projects provide good guidance as to what the effort required for the given project could be.
- Comparing data available for current project with past executed projects should provide this valuable estimated effort information.
- Thus, historical projects data come in handy for effort estimation.



Effort estimate deviation with elapse of time.

Function Point (FP) Metric

- Function point metric was proposed by Albrecht in 1983.
- This metric overcomes many of the shortcomings of the LOC metric.
- Since its inception in late 1970s, function point metric has steadily gained popularity.
- Function point metric has several advantages over LOC metric.
- One of the important advantages of the function point metric over the LOC metric is that it can easily be computed from the problem specification itself.
- Using the LOC metric, on the other hand, the size can accurately be determined only after the product has fully been developed.
- The conceptual idea behind the function point metric is the following.
- The size of a software product is directly dependent on the number of different high-level functions or features it supports.
- This assumption is reasonable, since each feature would take additional effort to implement.
- Though each feature takes some effort to develop, different features may take very different amounts of efforts to develop.
- The implicit assumption made is that the more the number of data items that a function reads from the user and outputs and the more the number of files accessed, the higher is the complexity of the function.

Function Point (FP) Metric

- Albrecht postulated that in addition to the number of basic functions that a software performs, size also depends on the number of files and the number of interfaces that are associated with the software.
- Here, interfaces refer to the different mechanisms for data transfer with external systems including the interfaces with the user, interfaces with external computers, etc.

Function point (FP) metric computation

- The size of a software product (in units of function points or FPs) is computed using different characteristics of the product identified in its requirements specification.
- It is computed using the following three steps:
    - Step 1: Compute the unadjusted function point (UFP) using a heuristic expression.
    - Step 2: Refine UFP to reflect the actual complexities of the different parameters used in UFP computation.
    - Step 3: Compute FP by further refining UFP to account for the specific characteristics of the project that can influence the entire development effort.

Function Point (FP) Metric

Step 1: UFP computation

- The unadjusted function points (UFP) is computed as the weighted sum of five characteristics of a product as shown in the following expression.
- The weights associated with the five characteristics were determined empirically by Albrecht through data gathered from many projects.

UFP=(No. of I/P)*4 + (No. of O/P)*5 + (No. of inquiries)*4 + (No. of files)*10 + (No. of interfaces)*10

1. Number of inputs: Each data item input by the user is counted. However, it should be noted that data inputs are considered different from user inquiries. Inquiries are user commands such as print-account-balance that require no data values to be input by the user. Inquiries are counted separately (see the third point below). It needs to be further noted that individual data items input by the user are not simply added up to compute the number of inputs, but related inputs are grouped and considered as a single input. For example, while entering the data concerning an employee to an employee payroll software; the data items name, age, sex, address, phone number, etc. are together considered as a single input. All these data items can be considered to be related, since they describe a single employee.

Function Point (FP) Metric

2. Number of outputs: The outputs considered include reports printed, screen outputs, error messages produced, etc. While computing the number of outputs, the individual data items within a report are not considered; but a set of related data items is counted as just a single
Output.

3. Number of inquiries: An inquiry is a user command (without any data input) and only requires some actions to be performed by the system. Thus, the total number of inquiries is essentially the number of distinct interactive queries (without data input) which can be made by the users. Examples of such inquiries are print account balance, print all student grades, display rank holders' names, etc.

4. Number of files: The files referred to here are logical files. A logical file represents a group of logically related data. Logical files include data structures as well as physical files.

5. Number of interfaces: Here the interfaces denote the different mechanisms that are used to exchange information with other systems. Examples of such interfaces are data files on tapes, disks, communication links with other systems, etc.

Function Point (FP) Metric

Step 2: Refine parameters

- UFP computed at the end of step 1 is a gross indicator of the problem size.
- This UFP needs to be refined.
- This is possible, since each parameter (input,output, etc.) has been implicitly assumed to be of average complexity.
- However , this is rarely true. For example, some input values may be extremely complex, some very simple, etc.
- In order to take this issue into account, UFP is refined by taking into account the complexities of the parameters of UFP computation.
- The complexity of each parameter is graded into three broad categories—simple, average, or complex.
- The weights for the different parameters are determined based on the numerical values shown in Table.
- Based on these weights of the parameters, the parameter values in the UFP are refined.
- For example, rather than each input being computed as four FPs, very simple inputs are computed as three FPs and very complex inputs as six FPs.

Function Point (FP) Metric

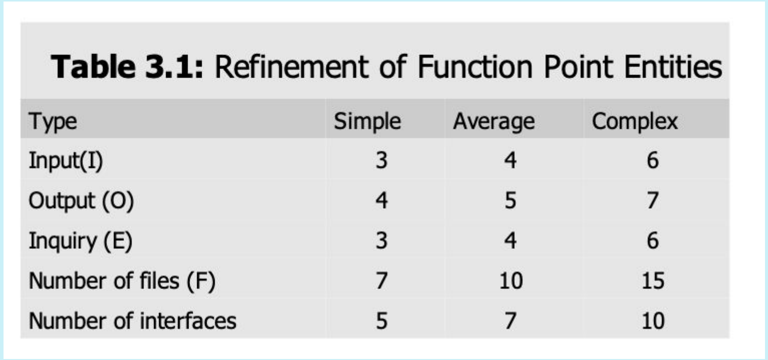


Table 3.1: Refinement of Function Point Entities

Type	Simple	Average	Complex
Input(I)	3	4	6
Output (O)	4	5	7
Inquiry (E)	3	4	6
Number of files (F)	7	10	15
Number of interfaces	5	7	10

Step 3: Refine UFP based on complexity of the overall project

- In the final step, several factors that can impact the overall project size are considered to refine the UFP computed in step 2.
- Examples of such project parameters that can influence the project sizes include high transaction rates, response time requirements, scope for reuse, etc.
- Albrecht identified 14 parameters that can influence the development effort.
- The list of these parameters have been shown in Table

Function Point (FP) Metric

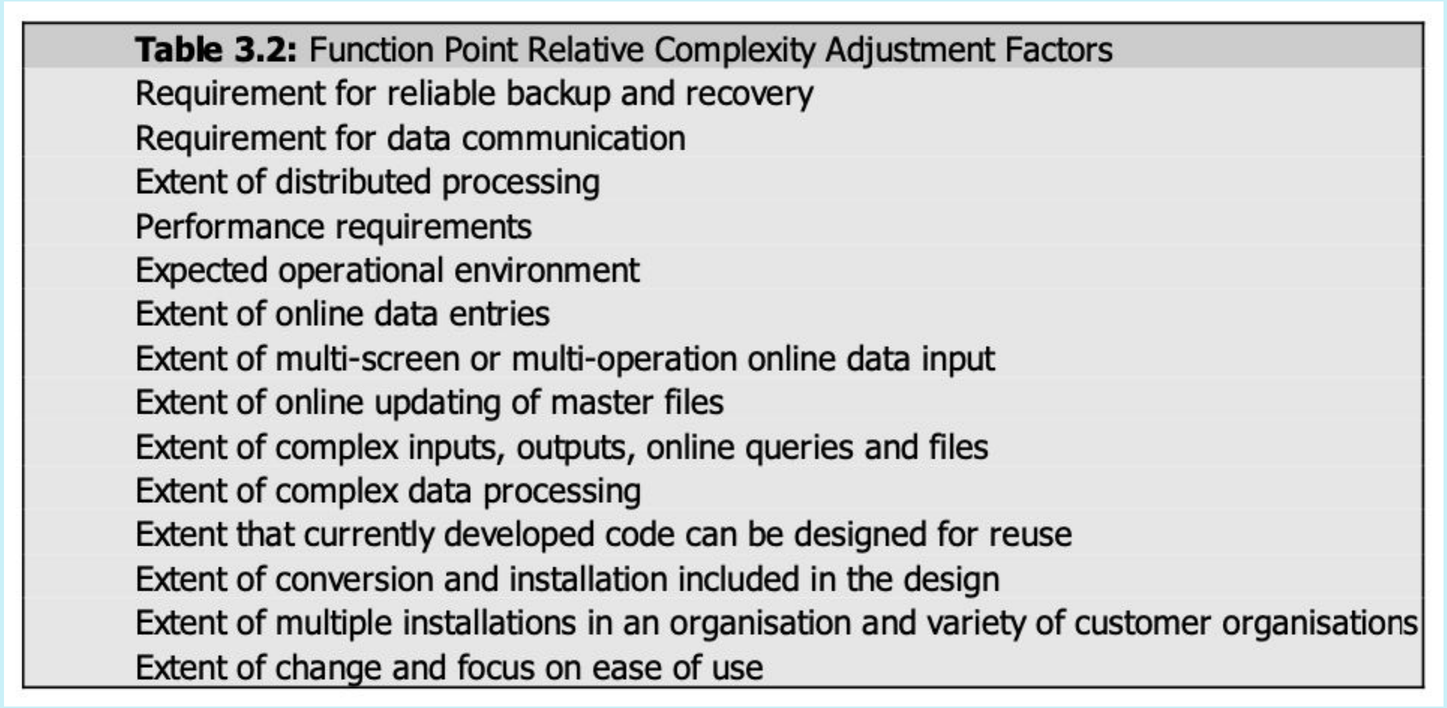
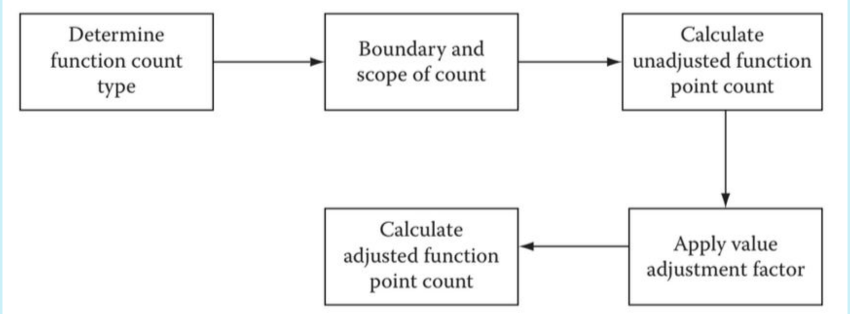


Table 3.2: Function Point Relative Complexity Adjustment Factors

Requirement for reliable backup and recovery
Requirement for data communication
Extent of distributed processing
Performance requirements
Expected operational environment
Extent of online data entries
Extent of multi-screen or multi-operation online data input
Extent of online updating of master files
Extent of complex inputs, outputs, online queries and files
Extent of complex data processing
Extent that currently developed code can be designed for reuse
Extent of conversion and installation included in the design
Extent of multiple installations in an organisation and variety of customer organisations
Extent of change and focus on ease of use

Function Point (FP) Metric

- Each of these 14 parameters is assigned a value from 0 (not present or no influence) to 6 (strong influence).The resulting numbers are summed, yielding the total degree of influence (DI).
- A technical complexity factor (TCF) for the project is computed and the TCF is multiplied with UFP to yield FP.
- TCF expresses the overall impact of corresponding project parameters on the development effort.
- TCF is computed as (0.65+0.01*DI). As DI can vary from 0 to 84, TCF can vary from 0.65 to 1.49.
- Finally, FP is given as the product of UFP and TCF .
- That is,FP=UFP*TCF.


Determine function count type
Boundary and scope of count
Calculate unadjusted function point count
Apply value adjustment factor
Calculate adjusted function point count

Function point count process steps.

Delphi Cost Estimation

- Delphi cost estimation technique is carried out by a team comprising a group of experts and a co-ordinator .
- In this approach, the co-ordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate.
- Estimators complete their individual estimates anonymously and submit them to the co-ordinator .
- In their estimates, the estimators mention any unusual characteristic of the product which has influenced their estimations.
- The co-ordinator prepares the summary of the responses of all the estimators, and also includes any unusual rationale noted by any of the estimators.
- The prepared summary information is distributed to the estimators.
- Based on this summary, the estimators re-estimate.This process is iterated for several rounds.
- However , no discussions among the estimators is allowed during the entire estimation process.
- The purpose behind this restriction is that if any discussion is allowed among the estimators,then many estimators may easily get influenced by the rationale of an estimator who may be more experienced or senior.

Delphi Cost Estimation

- After the completion of several iterations of estimations, the co-ordinator takes the responsibility of compiling the results and preparing the final estimate.
- The Delphi estimation, though consumes more time and effort, overcomes an important shortcoming of the expert judgement technique in that the results can not unjustly be influenced by overly
- assertive and senior members.
- Effort estimate = (pessimistic estimate + likely estimate × 4 + optimistic estimate)/6
- Here pessimistic estimate is the one where a team member’s estimate is the highest (in terms of number of man months).
- The likely estimate is the average of the most common estimate figure.
- In most cases, the likely estimate is the estimate given by the person who has been assigned to the task for which the effort estimate is being made.
- The optimistic estimate is the one where a team member’s estimate is the lowest (in terms of number of man months).

COnstructive COst estimation MOdel (COCOMO)

- COnstructive COst estimation MOdel (COCOMO) was proposed by Boehm [1981].
- COCOMO prescribes a three stage process for project estimation.
- In the first stage, an initial estimate is arrived at.
- Over the next two stages, the initial estimate is refined to arrive at a more accurate estimate.
- COCOMO uses both single and multivariable estimation models at different stages of estimation.
- The three stages of COCOMO estimation technique are—
    - basic COCOMO,
    - intermediate COCOMO, and
    - complete COCOMO.
-

COnstructive COst estimation MOdel (COCOMO)

Basic COCOMO

- There are many ways COCOMO calculations can be made, as variations of the original COCOMO model have been improved upon or adapted to suit many environments.
- For a quick effort calculation, a variation of the COCOMO model is used which is known as basic COCOMO.
- The basic COCOMO calculation equation is as follows: Effort = 2.94 * EAF *(KLOC)^ E
  - EAF is the effort adjustment factor derived from cost drivers
  - E is the exponent derived from scale drivers
  - KLOC is the kilo lines of software code
  - Values for EAF range from 1.0 to 2.0. Values for E range from 1.0 to 1.5.
- Schedule duration is calculated as   Duration = × 3.67 (effort)^SE
  - where SE is the schedule equation derived from scale drivers.
- Staffing needs can be calculated by dividing effort with duration.
- In the basic COCOMO model, hardware constraints, use of modern tools and techniques,personal productivity, etc. are not taken into account.
- Basic COCOMO is most suitable for making estimates at early stage of any project.

COnstructive COst estimation MOdel (COCOMO)

Intermediate COCOMO

- In intermediate COCOMO, we make an effort estimate for the project with the product size along with the cost drivers.
- The cost driver set includes assessment of attributes for product, project, hardware, and the project team’s experience and skills.
- These attributes are categorized as product attributes, which include required reliability, application database size, and application complexity.
- Hardware attributes include run-time performance constraint, memory constraint, virtual machine environment volatility, turnabout time requirement.
- Project team attributes include analyst capability, software engineer capability, application experience, virtual machine experience, and programming language experience.
- Project attributes include software tool usage, software engineering methods usage, and development schedule requirement.
- How each of the cost drivers impacts the effort estimate is assessed by assigning appropriate weights to these attributes.

COnstructive COst estimation MOdel (COCOMO)

Intermediate COCOMO

- To assign these weights, first a six-point scale is created with scales of very low, low, nominal, high, very high, and extra high.
- The values for these scales vary from a low of 0.70 to a high of 1.60.
- For any project, each of the attributes is given relevant values based on this scale.
- These attribute values are industry standard but at what scale value any attribute falls is decided by the estimating person.
- The formula for intermediate COCOMO is given as E = a(KLOC)^(E).EAF
- where a and E are a coefficients whose values depend on the kind of software project (organic, semi-detached, or embedded) for which the estimation is being made .

COnstructive COst estimation MOdel (COCOMO)

Detailed COCOMO

- In basic and intermediate COCOMO, the effort estimate is a gross estimate at the project level.
- But a project is further divided into many phases.
- Each phase may need to have a separate effort estimate calculation.
- This is done in the detailed COCOMO model.
- In the initial stages of the project, when a rough estimate is needed for each project phase, the basic COCOMO model is used.
- In later stages in the project when all project details are clear and an effort estimate is needed for each project phase, the intermediate COCOMO is used to calculate the effort estimate for each phase.
- The same values that are used for calculation at the project level can be used for calculations at the phase level.
- The only difference will be that at this level, the effort estimate will take values for relevant cost driver attributes and not for the entire project.
- For instance, for the design phase,the effort estimate will take attribute values only for cost drivers that will influence the design phase.

COnstructive COst estimation MOdel (COCOMO)-II

- The COCOMO II project [Boehm et al., 1995; Horowitz, 1997] is an effort being performed by the USC Center for Software Engineering, with the financial and technical support of numerous industry affiliates.
- The objectives of this project are threefold:
  - To develop a software cost and schedule estimation model for the life-cycle practices of the 1990s and 2000s
  - To develop a software cost database and tool support for improvement of the cost model
  - To provide a quantitative analytic framework for evaluating software technologies and their economic impact.

- COCOMO II strategy is to preserve the openness of the original COCOMO model, tailor it to the marketplace just described, key the inputs and outputs to the level of information available, and enable the model to be tailored to various project process strategies.
- In particular, this generation of COCOMO provides range estimates rather than point estimates.

COnstructive COst estimation MOdel (COCOMO)-II

- These vary over the life cycle from early, coarse-grained inputs and wide-ranging estimates to later, fine-grained inputs and more-precise estimates.
- To support this strategy, COCOMO II defines three different models for cost estimation.
- The models correspond to the level of fidelity and uncertainty appropriate for the current phase of the life cycle.
- The post-architecture model corresponds closely to the traditional COCOMO model, where it was assumed that the project had stable requirements, plans, and candidate architecture at the outset.
- Projects then followed a waterfall process through delivery with little requirements volatility.
- The post-architecture model provides for fine-grained estimates of the project once it has a requirements baseline, an architecture baseline, and a plan for the construction phase.
- The early design model provides for coarser grained estimates in the elaboration phase of the life cycle, and the applications composition model allows for very rough order-of-magnitude estimates during the inception phase of a project.
- The applications composition model corresponds to exploratory work typically done during prototyping efforts and feasibility analyses.
- The estimating equation is a simple linear relationship of object points and domain complexity.

COnstructive COst estimation MOdel (COCOMO)-II

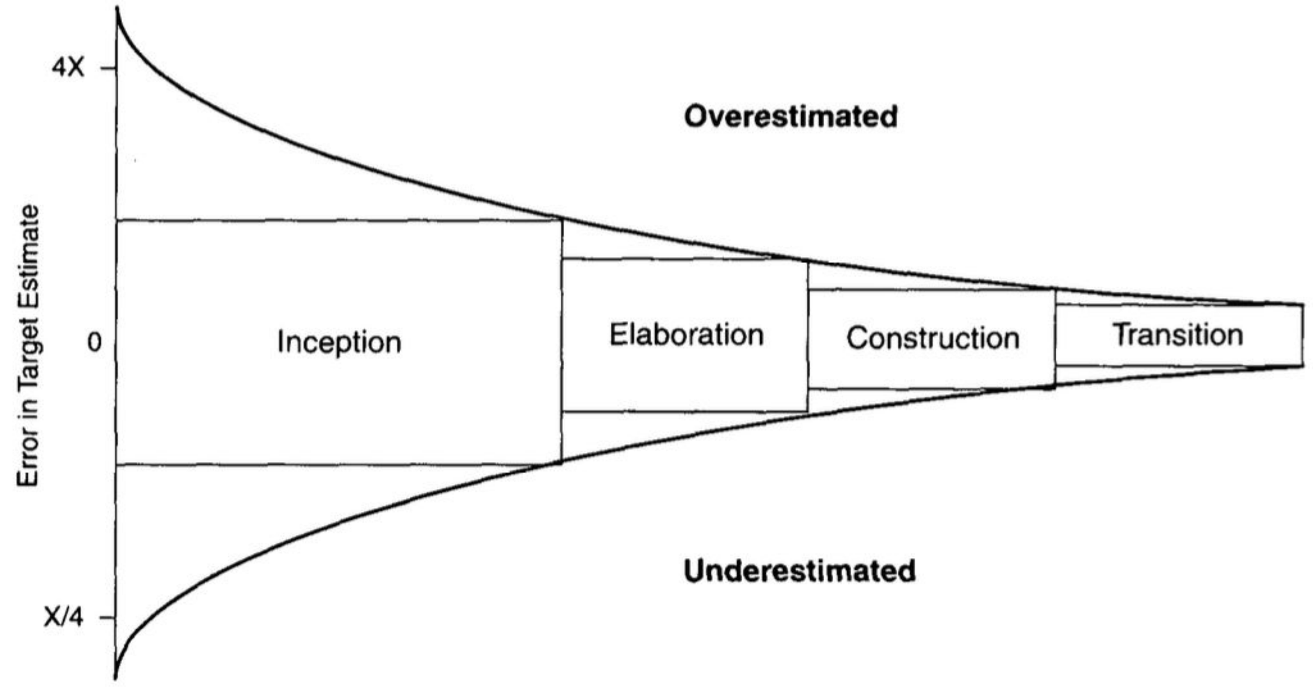
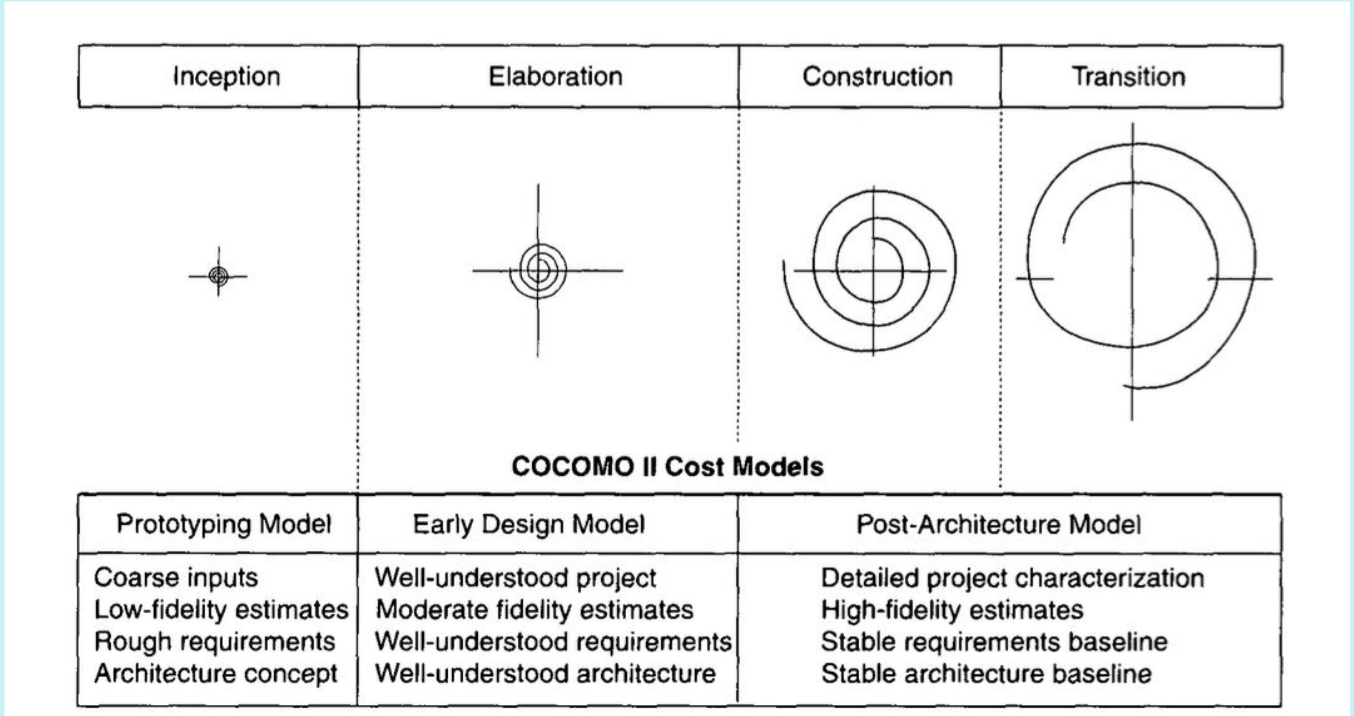


FIGURE B-2. *Software estimation over a project life cycle*

COnstructive COst estimation MOdel (COCOMO)-II



Prototyping Model	Early Design Model	Post-Architecture Model
Coarse inputs	Well-understood project	Detailed project characterization
Low-fidelity estimates	Moderate fidelity estimates	High-fidelity estimates
Rough requirements	Well-understood requirements	Stable requirements baseline
Architecture concept	Well-understood architecture	Stable architecture baseline

FIGURE B-3. *COCOMO II estimation over a project life cycle*

COnstructive COst estimation MOdel (COCOMO)-II

- COCOMO 2 provides three models to arrive at increasingly accurate cost estimations.
- These can be used to estimate project costs at different phases of the software product.
- As the project progresses, these models can be applied at the different stages of the same project.
  - Application composition model: This model as the name suggests, can be used to estimate the cost for prototype development.
  - Early design model: This supports estimation of cost at the architectural design stage.
  - Post-architecture model: This provides cost estimation during detailed design and coding stages. The post-architectural model can be considered as an update of the original COCOMO.
- The other two models help consider the following two factors. Now a days every software is interactive and GUI-driven. GUI development constitutes a significant part of the overall development effort.
- The second factor concerns several issues that affect productivity such as the extent of reuse.

COnstructive COst estimation MOdel (COCOMO)-II

Application composition model

- The application composition model is based on counting the number of screens, reports, and modules (components).
- Each of these components is considered to be an object (this has nothing to do with the concept of objects in the object-oriented paradigm).
- These are used to compute the object points of the application.
- Effort is estimated in the application composition model as follows:
  - Estimate the number of screens, reports, and modules (components) from an analysis of the SRS document.
  - Determine the complexity level of each screen and report, and rate these as either simple, medium, or difficult. The complexity of a screen or a report is determined by the number of tables and views it contains.
  - Use the weight values in Table. The weights have been designed to correspond to the amount of effort required to implement an instance of an object at the assigned complexity class.
  - Add all the assigned complexity values for the object instances together to obtain the object points.

COnstructive COst estimation MOdel (COCOMO)-II

Application composition model

- Estimate percentage of reuse expected in the system. Note that reuse refers to the amount of pre-developed software that will be used within the system. Then, evaluate New Object-Point count (NOP) as follows,

$$NOP = \frac{(\text{Object-Points})(100 - \%\text{ of reuse})}{100}$$

- Determine the productivity using Table. The productivity depends on the experience of the developers as well as the maturity of the CASE environment used.
- Finally, the estimated effort in person-months is computed as E =NOP/PROD.

COnstructive COst estimation MOdel (COCOMO)-II

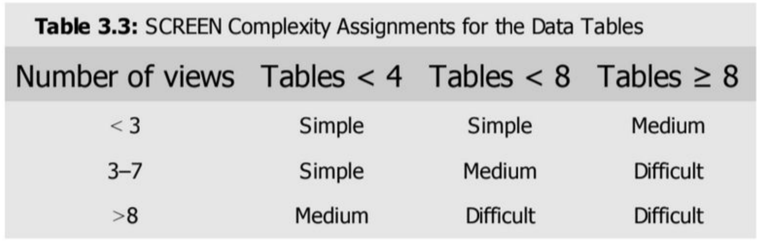


Table 3.3: SCREEN Complexity Assignments for the Data Tables

Number of views	Tables < 4	Tables < 8	Tables ≥ 8
< 3	Simple	Simple	Medium
3–7	Simple	Medium	Difficult
>8	Medium	Difficult	Difficult

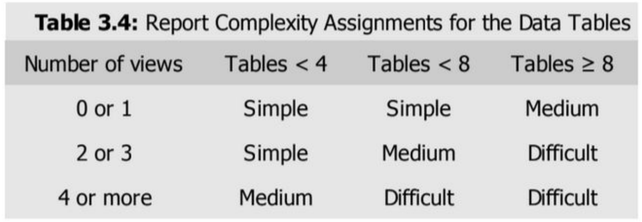


Table 3.4: Report Complexity Assignments for the Data Tables

Number of views	Tables < 4	Tables < 8	Tables ≥ 8
0 or 1	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
4 or more	Medium	Difficult	Difficult

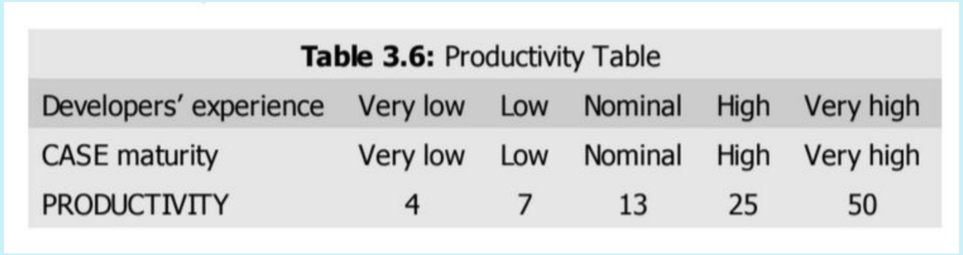


Table 3.6: Productivity Table

Developers' experience	Very low	Low	Nominal	High	Very high
CASE maturity	Very low	Low	Nominal	High	Very high
PRODUCTIVITY	4	7	13	25	50

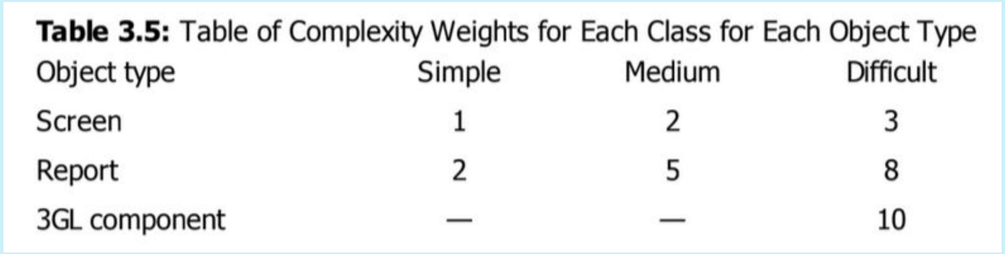


Table 3.5: Table of Complexity Weights for Each Class for Each Object Type

Object type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	—	—	10

COnstructive COst estimation MOdel (COCOMO)-II

- The early design model corresponds to the level of detail available in the engineering stage of a project, during which the architecture, requirements, and plans are being synthesized.
- The overall cost estimate equation is as follows:

  Effort = 2.45 EArch (Size)^p

  - Effort = number of staff-months
  - EArch = product of seven early design effort adjustment factors (Table B-5)
  - Size = number of function points (preferred) or KSLOC
  - P = process exponent
- The early design phase parameters are composites of the post-architecture parameters.
- They provide a simpler estimating method for the early life cycle when there are more unknowns.
- The post-architecture cost estimating equation is as follows:

  Effort = 2.45 EApp (Size)"

  - Effort = number of staff-months
  - EApp= product of 17 post-architecture effort adjustment factors (Table B-6)
  - Size = number of KSLOC (preferred) or function points
  - P = process exponent

COnstructive COst estimation MOdel (COCOMO)-II

- COCOMO II uses the same exponent for the early design and the post-architecture models.
- The process exponent can range from (1.01..1.26) and is defined as the combined effects of the following five parameters:
  - Application precedentedness: the degree of domain experience of the development organization
  - Process flexibility: the degree of contractual rigor, ceremony, and change freedom inherent in the project contract, life-cycle activities, and stakeholder communications
  - Architecture risk resolution: the degree of technical feasibility demonstrated before commitment to full-scale production
  - Team cohesion: the degree of cooperation and shared vision among stakeholders (buyers, developers, users, and maintainers, among others)
  - Process maturity: the maturity level of the development organization, as defined by the Software Engineering Institute's Capability Maturity Model
- The COCOMO II exponent parameterization is an evolutionary upgrade of the Ada COCOMO approach with a more solid basis.

COnstructive COst estimation MOdel (COCOMO)-II

- The actual exponent for COCOMO II is determined by summing the effects for each parameter. The combined impact of these process parameters can be very high.
- The COCOMO II team has yet to permit an actual economy of scale to be achieved (that is, the value of P is never less than 1.0).
- They believe that economy of scale is achievable through corresponding reductions in size resulting from use of commercial components, reusable components, CASE tools, and object-oriented technologies.