Chapter 4 – Requirements Engineering

DR.A.SHANTHINI
ASSOCIATE PROFESSOR
DEPARTMENT OF DSBS
SRM IST

Topics covered

Functional and non-functional requirements

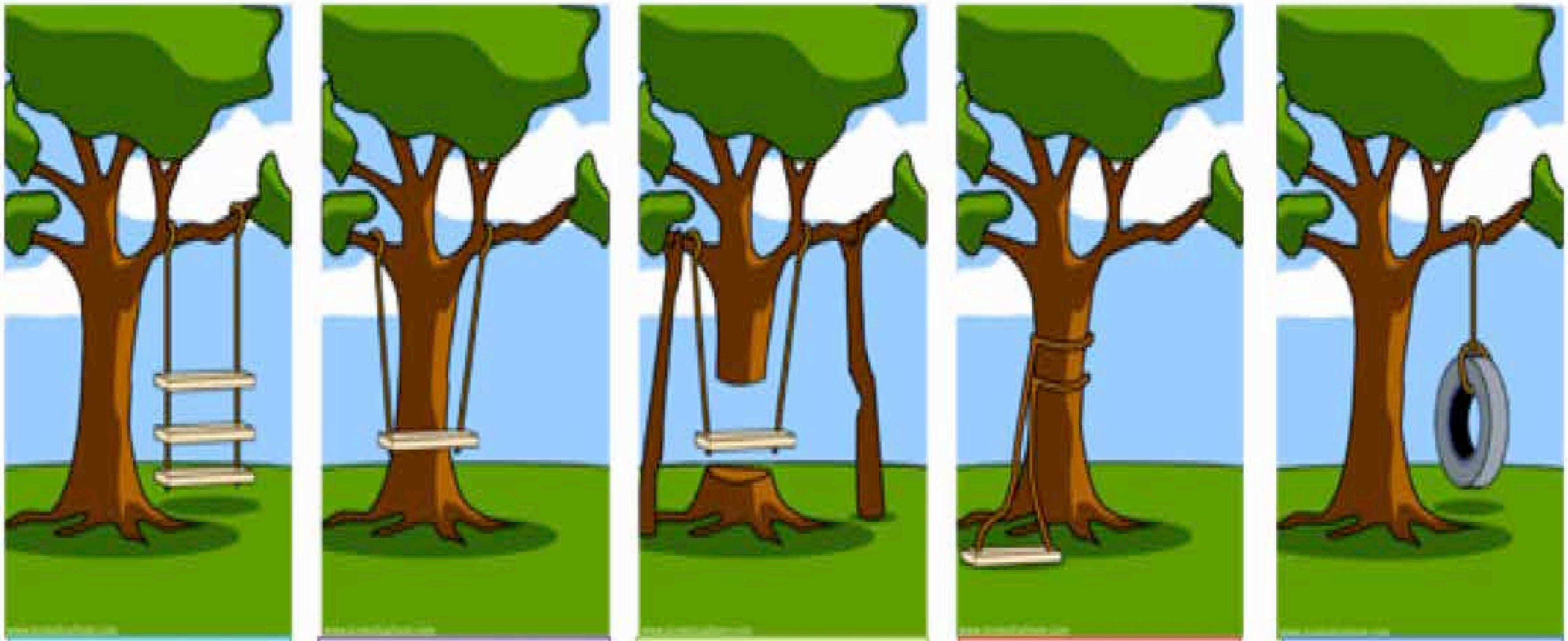
Requirements engineering processes

Requirements elicitation

Requirements specification

Requirements validation

Requirements change / Management



How the customer explained it.

How the Project Manager Understood it.

How the Engineer Designed it.

How the Technician Built it.

How the Customer really wanted it.

Requirements engineering

The process of establishing the services that a customer requires from a system and the constraints under which it operates and is developed.

The system requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is a requirement?

It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

This is inevitable as requirements may serve a dual function

- May be the basis for a bid for a contract - therefore must be open to interpretation;
- May be the basis for the contract itself - therefore must be defined in detail;
- Both these statements may be called requirements.

Requirements abstraction (Davis)

"If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system."

Types of requirement

User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

System requirements

- A structured document setting out detailed descriptions of the system’s functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and system requirements

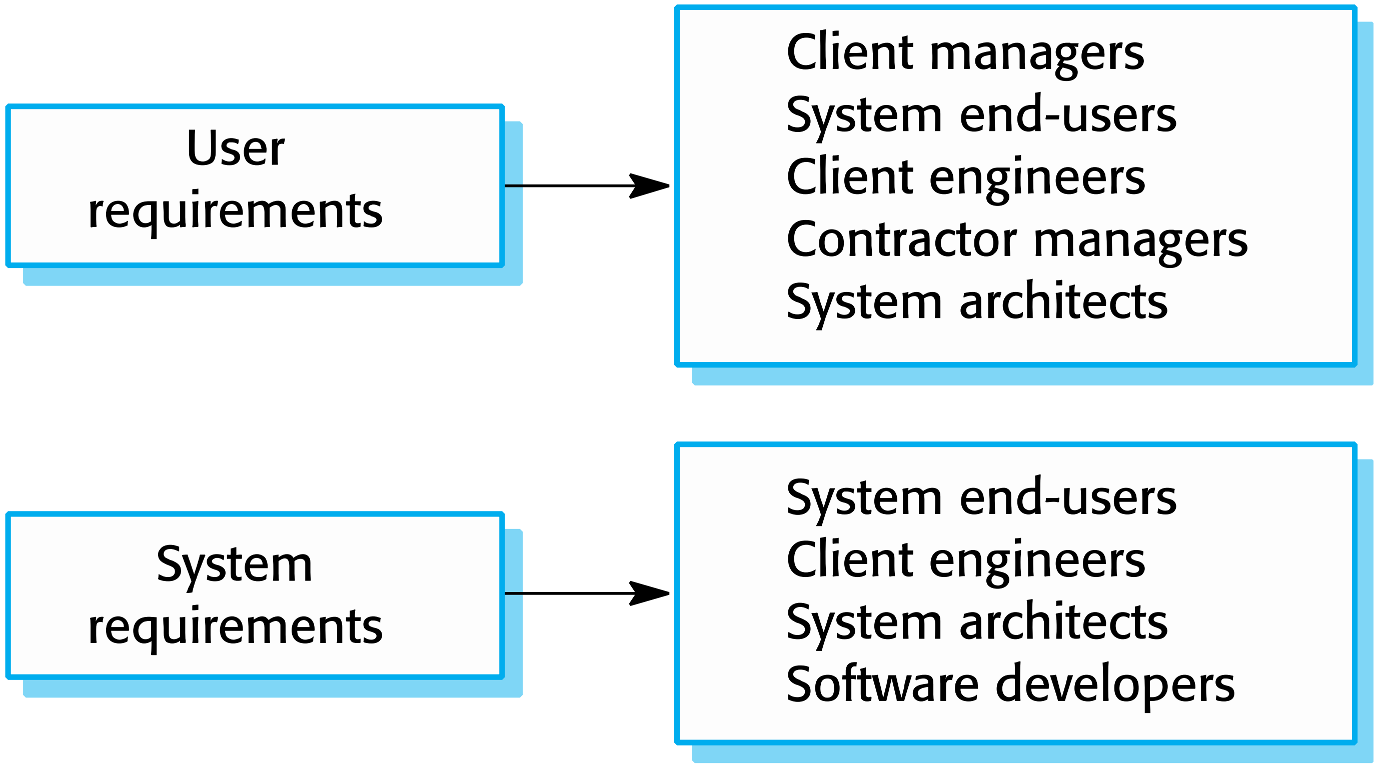
User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Readers of different types of requirements specification



Functional and non-functional requirements

Functional and non-functional requirements

Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- May state what the system should not do.

Non-functional requirements

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Often apply to the system as a whole rather than individual features or services.

Domain requirements

- Constraints on the system from the domain of operation

Functional requirements

Describe functionality or system services.

Depend on the type of software, expected users and the type of system where the software is used.

Functional user requirements may be high-level statements of what the system should do.

Functional system requirements should describe the system services in detail.

Mentcare system: functional requirements

A user shall be able to search the appointments lists for all clinics.

The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.

Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Requirements imprecision

Problems arise when functional requirements are not precisely stated.

Ambiguous requirements may be interpreted in different ways by developers and users.

Consider the term ‘search’ in requirement 1

- User intention – search for a patient name across all appointments in all clinics;
- Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

Requirements completeness and consistency

In principle, requirements should be both complete and consistent.

Complete

- They should include descriptions of all facilities required.

Consistent

- There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document.

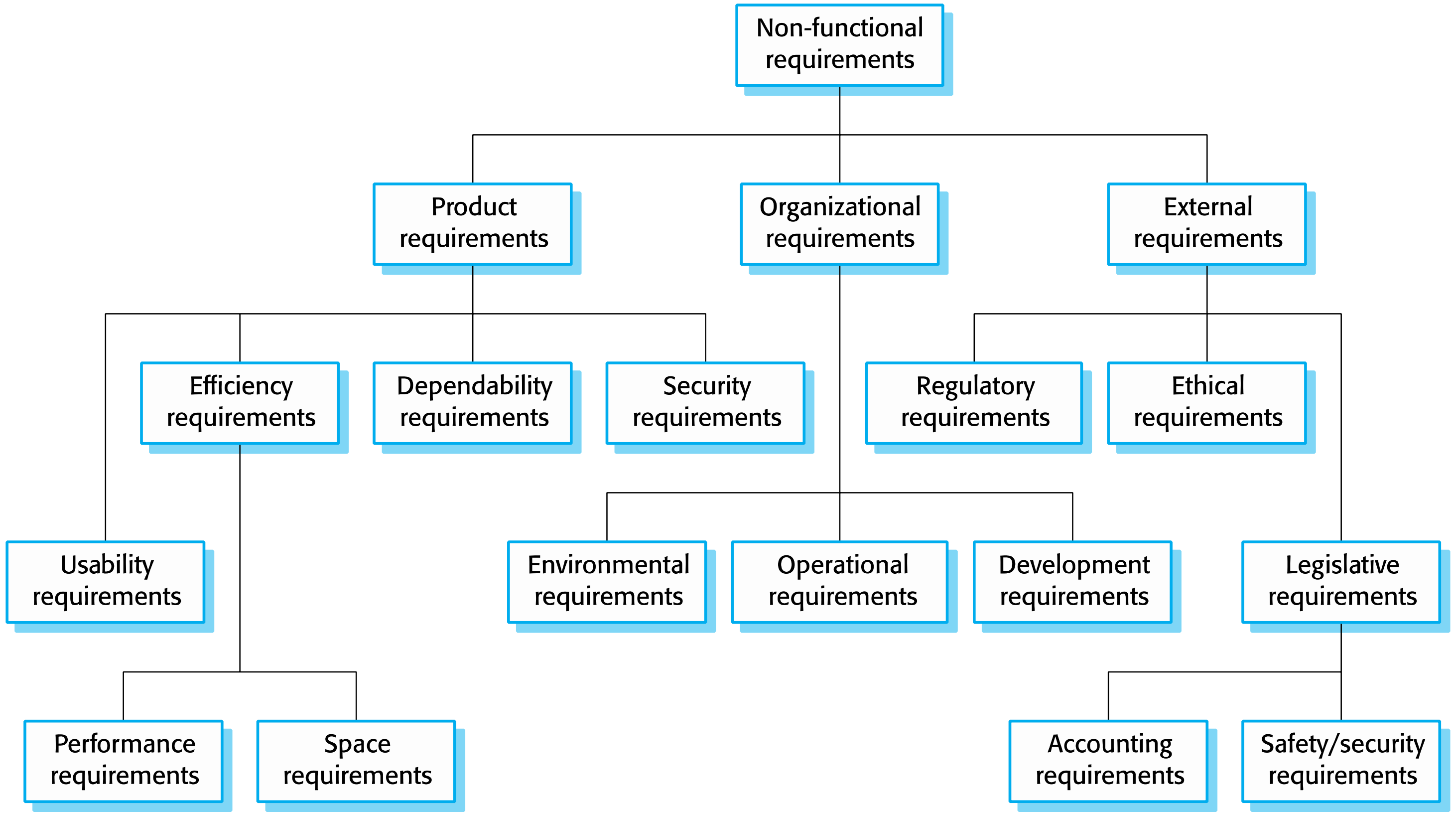
Non-functional requirements

These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

Process requirements may also be specified mandating a particular IDE, programming language or development method.

Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of nonfunctional requirement



Non-functional requirements implementation

Non-functional requirements may affect the overall architecture of a system rather than the individual components.

- For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.

A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.

- It may also generate requirements that restrict existing requirements.

Non-functional classifications

Product requirements

- Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

Organisational requirements

- Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.

External requirements

- Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Examples of nonfunctional requirements in the Mentcare system

Product requirement
The Mentcare system shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement
Users of the Mentcare system shall authenticate themselves using their health authority identity card.

External requirement
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Goals and requirements

Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.

Goal

- A general intention of the user such as ease of use.

Verifiable non-functional requirement

- A statement using some measure that can be objectively tested.

Goals are helpful to developers as they convey the intentions of the system users.

Usability requirements

The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)

Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)

Metrics for specifying nonfunctional requirements

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

Software Requirement Document

(OR)

Software Requirement specification (SRS)

Requirements specification

Requirements specification

The process of writing down the user and system requirements in a requirements document.

User requirements have to be understandable by end-users and customers who do not have a technical background.

System requirements are more detailed requirements and may include more technical information.

The requirements may be part of a contract for the system development

- It is therefore important that these are as complete as possible.

Ways of writing a system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don’t understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Requirements and design

In principle, requirements should state what the system should do and the design should describe how it does this.

In practice, requirements and design are inseparable

- A system architecture may be designed to structure the requirements;
- The system may inter-operate with other systems that generate design requirements;
- The use of a specific architecture to satisfy non-functional requirements may be a domain requirement.
- This may be the consequence of a regulatory requirement.

Natural language specification

Requirements are written as natural language sentences supplemented by diagrams and tables.

Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

Guidelines for writing requirements

Invent a standard format and use it for all requirements.

Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.

Use text highlighting to identify key parts of the requirement.

Avoid the use of computer jargon.

Include an explanation (rationale) of why a requirement is necessary.

Problems with natural language

Lack of clarity

- Precision is difficult without making the document difficult to read.

Requirements confusion

- Functional and non-functional requirements tend to be mixed-up.

Requirements amalgamation

- Several different requirements may be expressed together.

Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured specifications

An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.

This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

Form-based specifications

Definition of the function or entity.

Description of inputs and where they come from.

Description of outputs and where they go to.

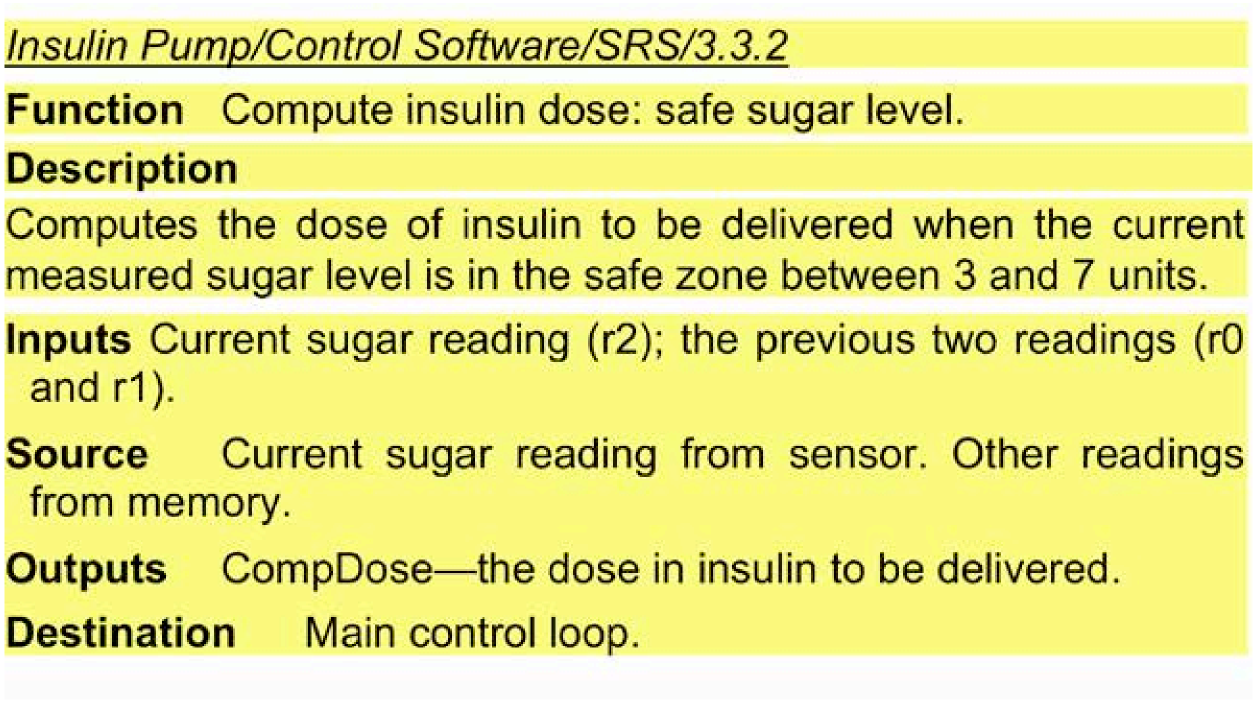
Information about the information needed for the computation and other entities used.

Description of the action to be taken.

Pre and post conditions (if appropriate).

The side effects (if any) of the function.

A structured specification of a requirement for an insulin pump



Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

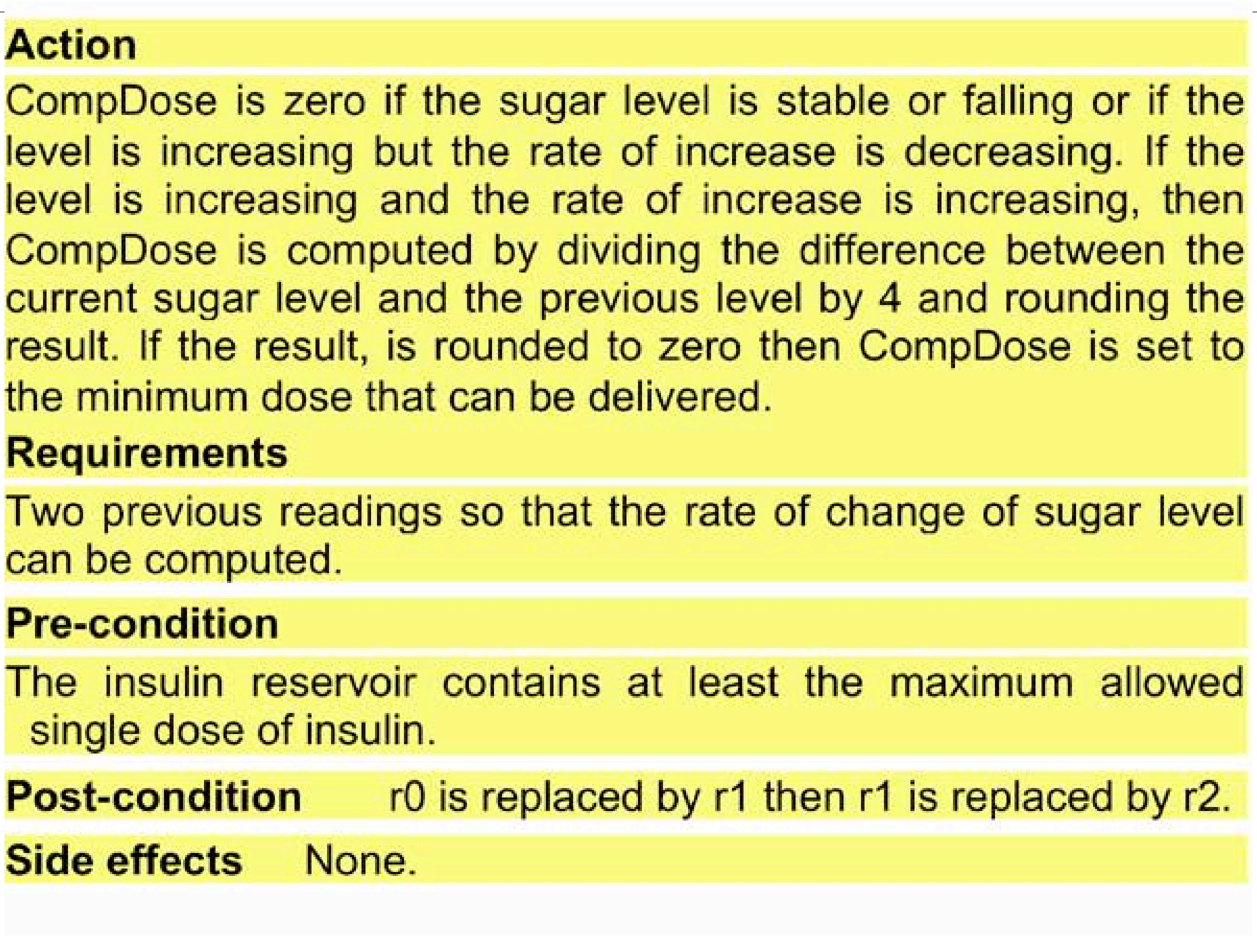
Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

A structured specification of a requirement for an insulin pump



Action

CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements

Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition

The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition r0 is replaced by r1 then r1 is replaced by r2.

Side effects None.

Tabular specification

Used to supplement natural language.

Particularly useful when you have to define a number of possible alternative courses of action.

For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Use cases

Use-cases are a kind of scenario that are included in the UML.

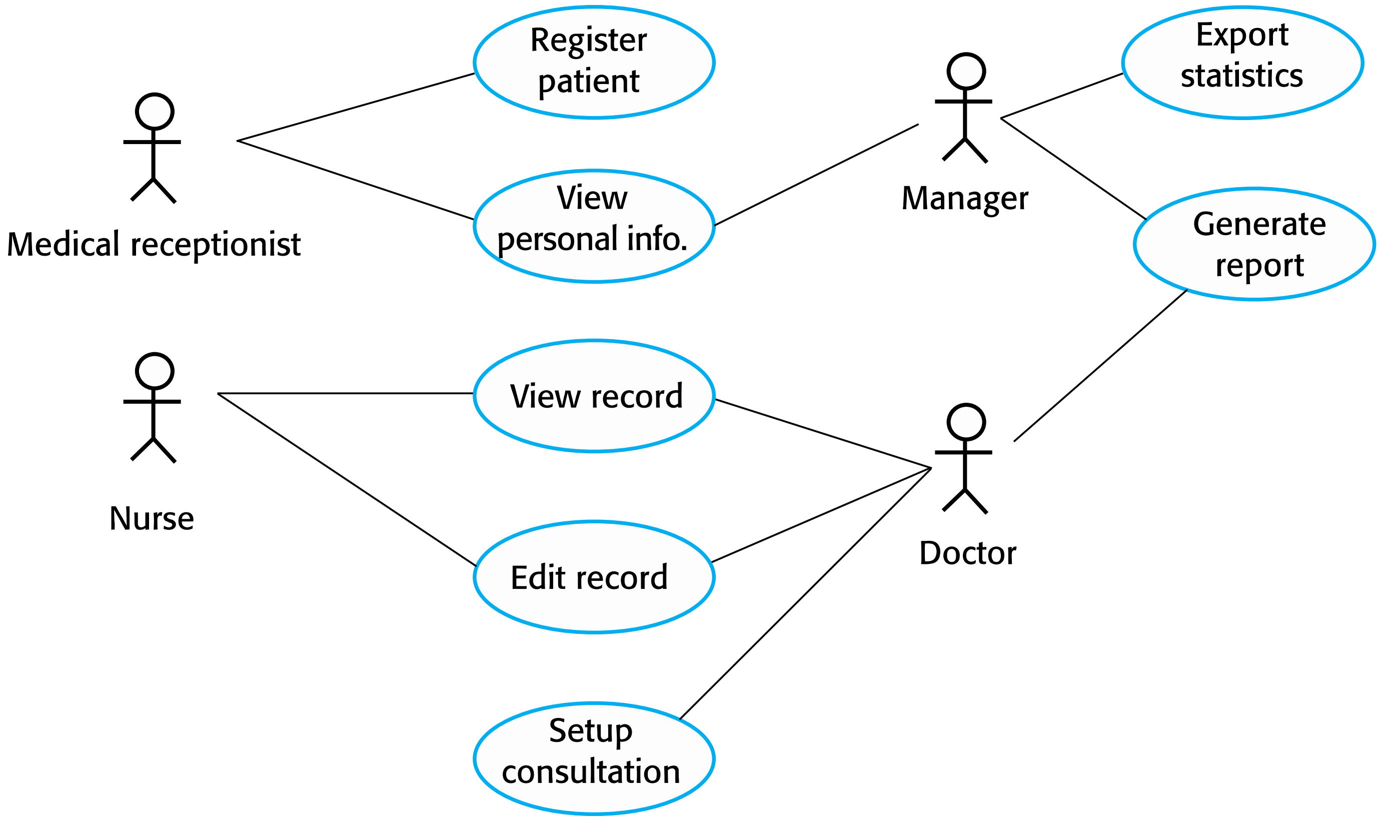
Use cases identify the actors in an interaction and which describe the interaction itself.

A set of use cases should describe all possible interactions with the system.

High-level graphical model supplemented by more detailed tabular description (see Chapter 5).

UML sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

Use cases for the Mentcare system



The software requirements document

The software requirements document is the official statement of what is required of the system developers.

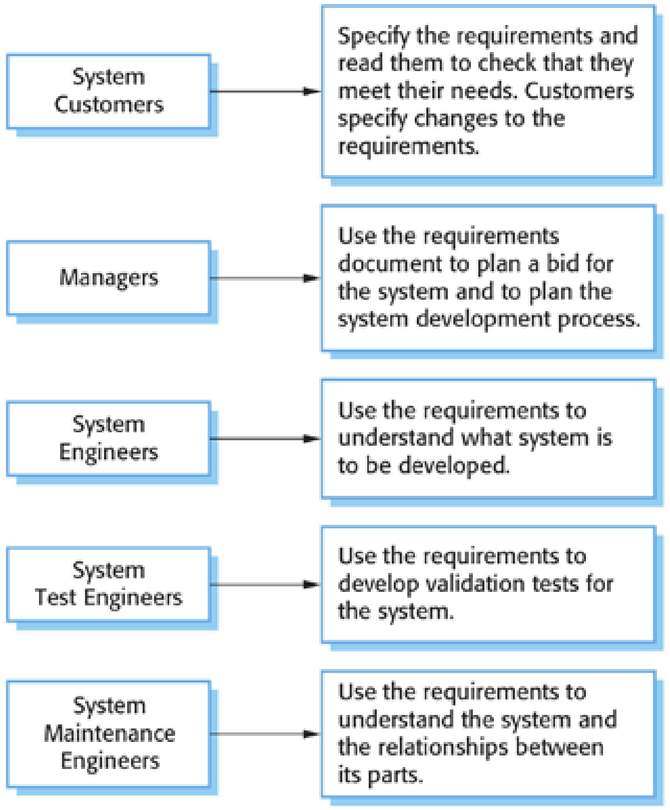
Should include both a definition of user requirements and a specification of the system requirements.

It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

SRS – Users of requirement doc

- 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.

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.

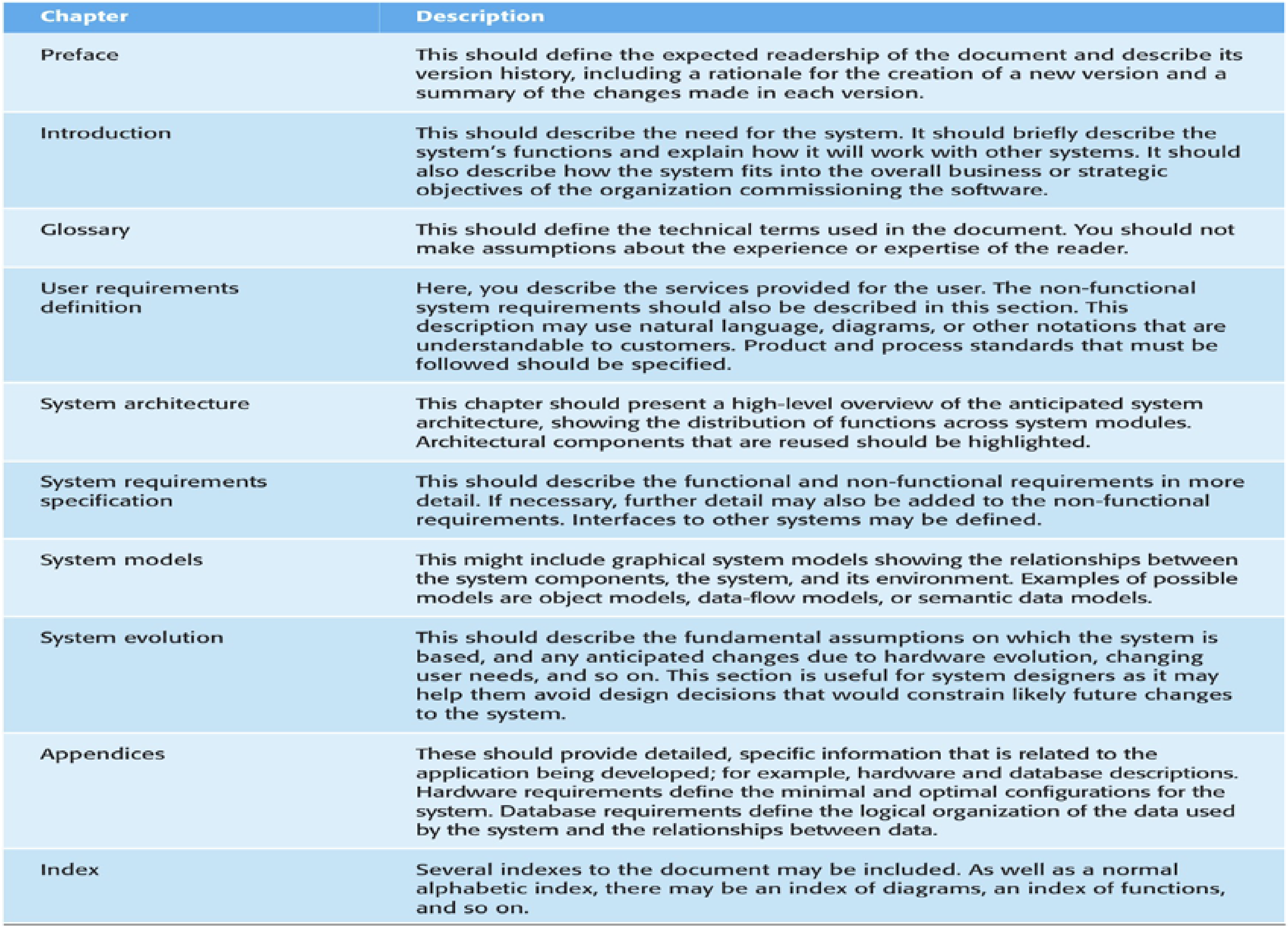
Requirements document variability

Information in requirements document depends on type of system and the approach to development used.

Systems developed incrementally will, typically, have less detail in the requirements document.

Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

Structure of requirement doc



Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components, the system, and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Requirements engineering processes

Requirements engineering processes

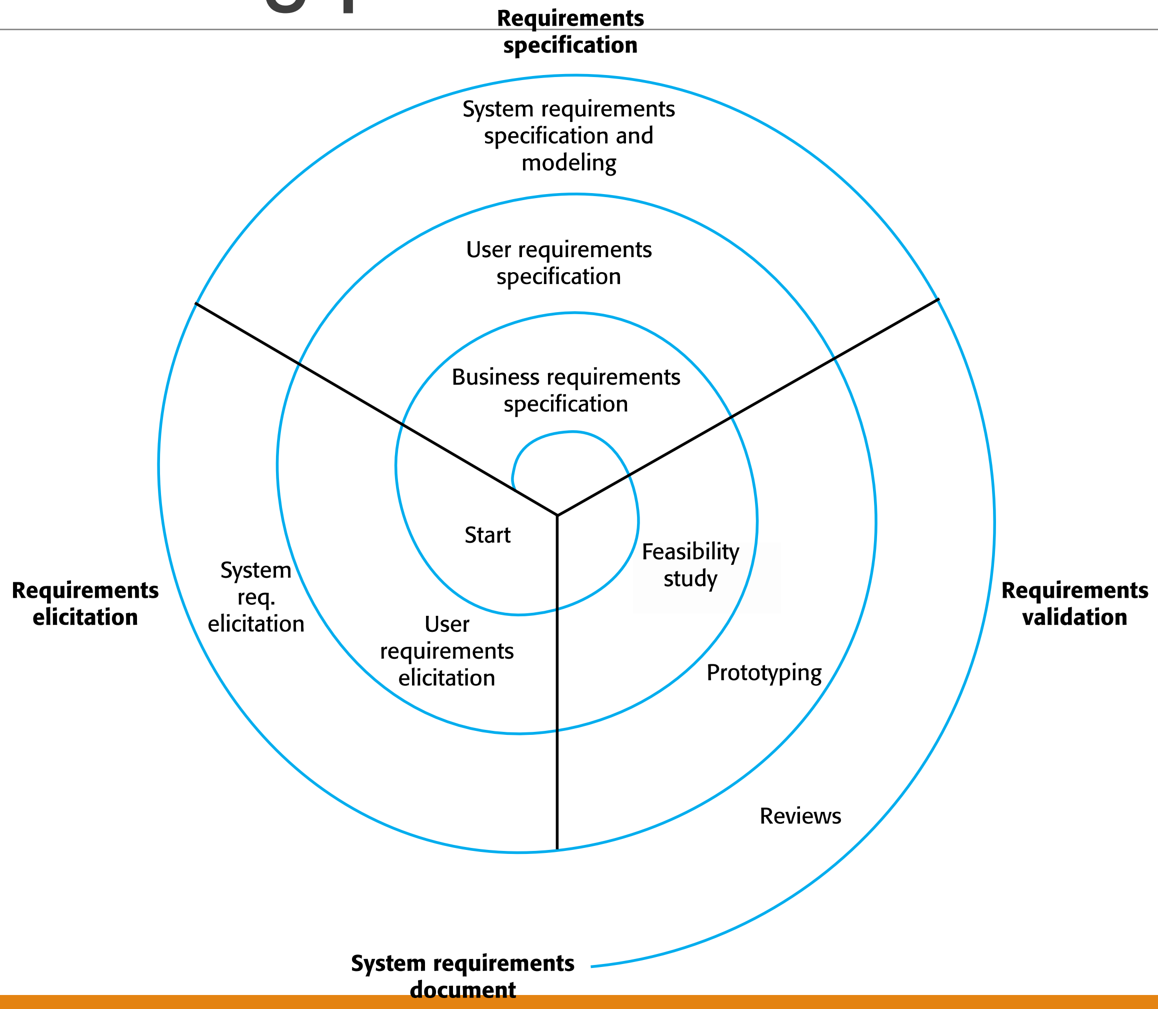
The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.

However, there are a number of generic activities common to all processes

1. Requirements elicitation;
2. Requirements analysis;
3. Requirements validation;
4. Requirements management.

In practice, RE is an iterative activity in which these processes are interleaved.

A spiral view of the requirements engineering process


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

1. Requirements elicitation

2.Requirements elicitation and analysis

Sometimes called requirements elicitation or requirements discovery.

Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system’s operational constraints.

May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

Requirements elicitation

Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.

Stages include:

- Requirements discovery,
- Requirements classification and organization,
- Requirements prioritization and negotiation,
- Requirements specification.

Problems of requirements elicitation

Stakeholders don’t know what they really want.

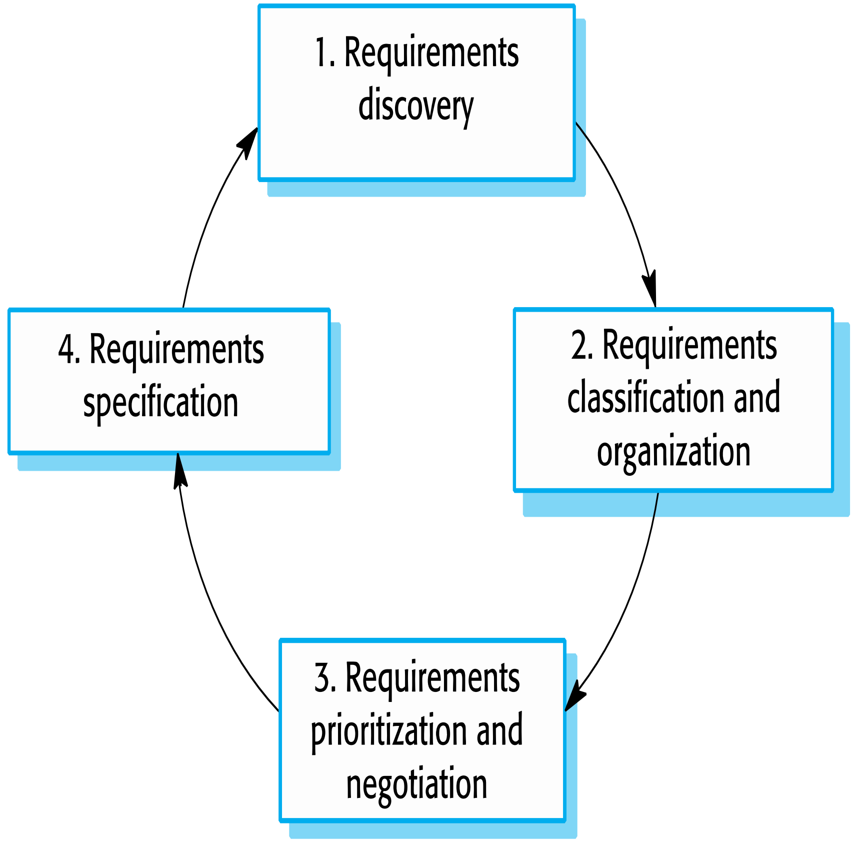
Stakeholders express requirements in their own terms.

Different stakeholders may have conflicting requirements.

Organisational and political factors may influence the system requirements.

The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

The requirements elicitation and analysis process


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

1) Requirements discovery: This is the process of interacting with stakeholders of the system to discover their requirements.
2) This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters.
3) when multiple stake holders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.
4) The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced

Process activities

Requirements discovery

- Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.

Requirements classification and organisation

- Groups related requirements and organises them into coherent clusters.

Prioritisation and negotiation

- Prioritising requirements and resolving requirements conflicts.

Requirements specification

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

Requirements discovery

The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.

Interaction is with system stakeholders from managers to external regulators.

Systems normally have a range of stakeholders.

Interviewing

Formal or informal interviews with stakeholders are part of most RE processes.

Types of interview

- Closed interviews based on pre-determined list of questions
- Open interviews where various issues are explored with stakeholders.

Effective interviewing

- Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
- Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Interviews in practice

Normally a mix of closed and open-ended interviewing.

Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.

Interviewers need to be open-minded without pre-conceived ideas of what the system should do

You need to prompt the use to talk about the system by suggesting requirements rather than simply asking them what they want.

Problems with interviews

Application specialists may use language to describe their work that isn’t easy for the requirements engineer to understand.

Interviews are not good for understanding domain requirements

- Requirements engineers cannot understand specific domain terminology;
- Some domain knowledge is so familiar that people find it hard to articulate or think that it isn’t worth articulating.

Ethnography

A social scientist spends a considerable time observing and analysing how people actually work.

People do not have to explain or articulate their work.

Social and organisational factors of importance may be observed.

Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

Scope of ethnography

Requirements that are derived from the way that people actually work rather than the way in which process definitions suggest that they ought to work.

Requirements that are derived from cooperation and awareness of other people's activities.

- Awareness of what other people are doing leads to changes in the ways in which we do things.

Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

Focused ethnography

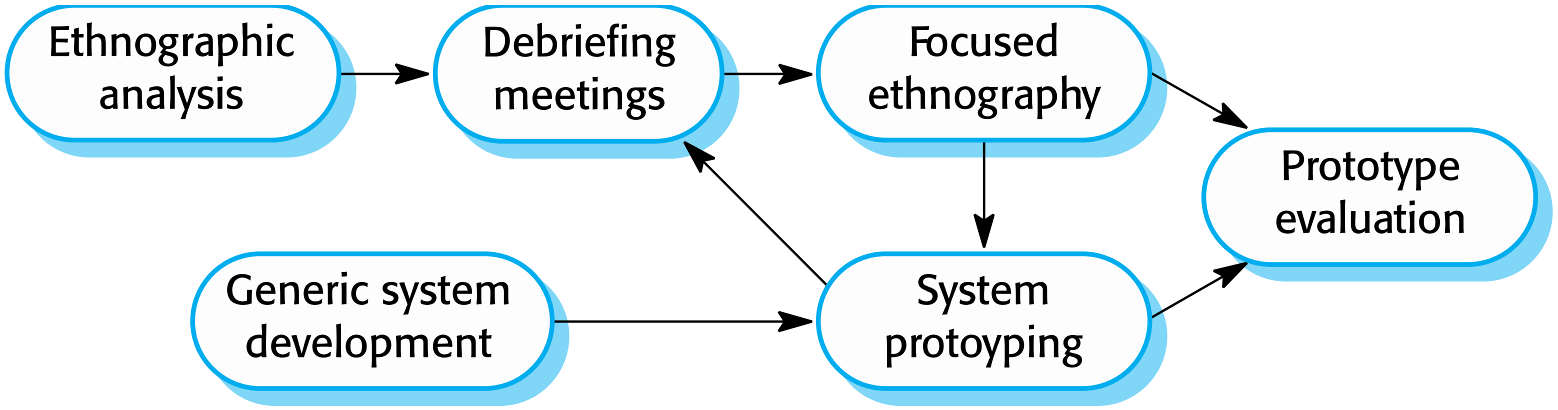
Developed in a project studying the air traffic control process

Combines ethnography with prototyping

Prototype development results in unanswered questions which focus the ethnographic analysis.

The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.

Ethnography and prototyping for requirements analysis



Stories and scenarios

Scenarios and user stories are real-life examples of how a system can be used.

Stories and scenarios are a description of how a system may be used for a particular task.

Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

Photo sharing in the classroom (iLearn)

Jack is a primary school teacher in Ullapool (a village in northern Scotland). He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact of fishing. As part of this, pupils are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history resources site) to access newspaper archives and photographs. However, Jack also needs a photo sharing site as he wants pupils to take and comment on each others' photos and to upload scans of old photographs that they may have in their families.

Jack sends an email to a primary school teachers group, which he is a member of to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account. He uses the iLearn setup service to add KidsTakePics to the services seen by the pupils in his class so that when they log in, they can immediately use the system to upload photos from their mobile devices and class computers.

Scenarios

A structured form of user story

Scenarios should include

- A description of the starting situation;
- A description of the normal flow of events;
- A description of what can go wrong;
- Information about other concurrent activities;
- A description of the state when the scenario finishes.

Uploading photos iLearn)

Initial assumption: A user or a group of users have one or more digital photographs to be uploaded to the picture sharing site. These are saved on either a tablet or laptop computer. They have successfully logged on to KidsTakePics.

Normal: The user chooses upload photos and they are prompted to select the photos to be uploaded on their computer and to select the project name under which the photos will be stored. They should also be given the option of inputting keywords that should be associated with each uploaded photo. Uploaded photos are named by creating a conjunction of the user name with the filename of the photo on the local computer.

On completion of the upload, the system automatically sends an email to the project moderator asking them to check new content and generates an on-screen message to the user that this has been done.

Uploading photos

What can go wrong:

No moderator is associated with the selected project. An email is automatically generated to the school administrator asking them to nominate a project moderator. Users should be informed that there could be a delay in making their photos visible.

Photos with the same name have already been uploaded by the same user. The user should be asked if they wish to re-upload the photos with the same name, rename the photos or cancel the upload. If they chose to re-upload the photos, the originals are overwritten. If they chose to rename the photos, a new name is automatically generated by adding a number to the existing file name.

Other activities: The moderator may be logged on to the system and may approve photos as they are uploaded.

System state on completion: User is logged on. The selected photos have been uploaded and assigned a status ‘awaiting moderation’. Photos are visible to the moderator and to the user who uploaded them.

3.Requirements validation

Requirements validation

Concerned with demonstrating that the requirements define the system that the customer really wants.

Requirements error costs are high so validation is very important

- Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements checking

Validity. Does the system provide the functions which best support the customer’s needs?

Consistency. Are there any requirements conflicts?

Completeness. Are all functions required by the customer included?

Realism. Can the requirements be implemented given available budget and technology

Verifiability. Can the requirements be checked?

Requirements validation techniques

Requirements reviews

- Systematic manual analysis of the requirements.

Prototyping

- Using an executable model of the system to check requirements. Covered in Chapter 2.

Test-case generation

- Developing tests for requirements to check testability.

Requirements reviews

Regular reviews should be held while the requirements definition is being formulated.

Both client and contractor staff should be involved in reviews.

Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Review checks

Verifiability
- Is the requirement realistically testable?

Comprehensibility
- Is the requirement properly understood?

Traceability
- Is the origin of the requirement clearly stated?

Adaptability
- Can the requirement be changed without a large impact on other requirements?

4.Requirements change & Management

Changing requirements

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.

The people who pay for a system and the users of that system are rarely the same people.

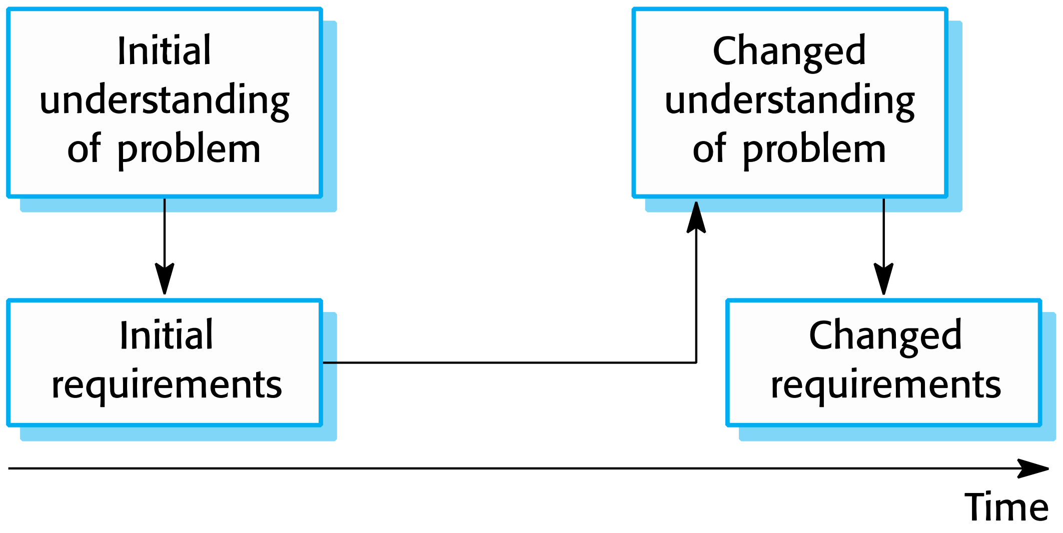
- System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing requirements

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 evolution


Initial understanding of problem
Changed understanding of problem
Initial requirements
Changed requirements
Time

Requirements management

Requirements management is the process of managing changing requirements during the requirements engineering process and system development.

New requirements emerge as a system is being developed and after it has gone into use.

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.

Requirements management planning

Establishes the level of requirements management detail that is required.

Requirements management decisions:

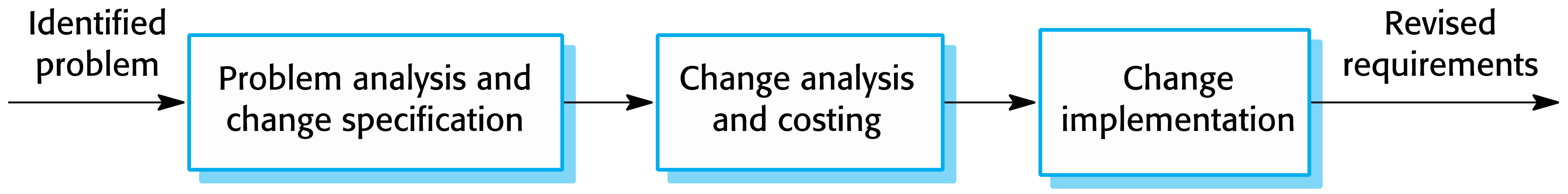
- *Requirements identification* Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
- *A change management process* This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
- *Traceability policies* These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
- *Tool support* Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements change management

Deciding if a requirements change should be accepted

- *Problem analysis and change specification*
  - 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.
- *Change analysis and costing*
  - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
- Change implementation
  - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

Requirements change management


Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements

Key points

Requirements for a software system set out what the system should do and define constraints on its operation and implementation.

Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.

Non-functional requirements often constrain the system being developed and the development process being used.

They often relate to the emergent properties of the system and therefore apply to the system as a whole.

Key points

The requirements engineering process is an iterative process that includes requirements elicitation, specification and validation.

Requirements elicitation is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.

You can use a range of techniques for requirements elicitation including interviews and ethnography. User stories and scenarios may be used to facilitate discussions.

Key points

Requirements specification is the process of formally documenting the user and system requirements and creating a software requirements document.

The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.

Key points

Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.

Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.

Chapter 26

- **Estimation for Software Projects**

Slide Set to accompany

Software Engineering: A Practitioner’s Approach, 7/e

by Roger S. Pressman

Software Project Planning

The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?

So the end result gets done on time, with quality!

Project Planning Task Set-I

- Establish project scope
- Determine feasibility
- Analyze risks
  - Risk analysis is considered in detail in Chapter 25.
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources

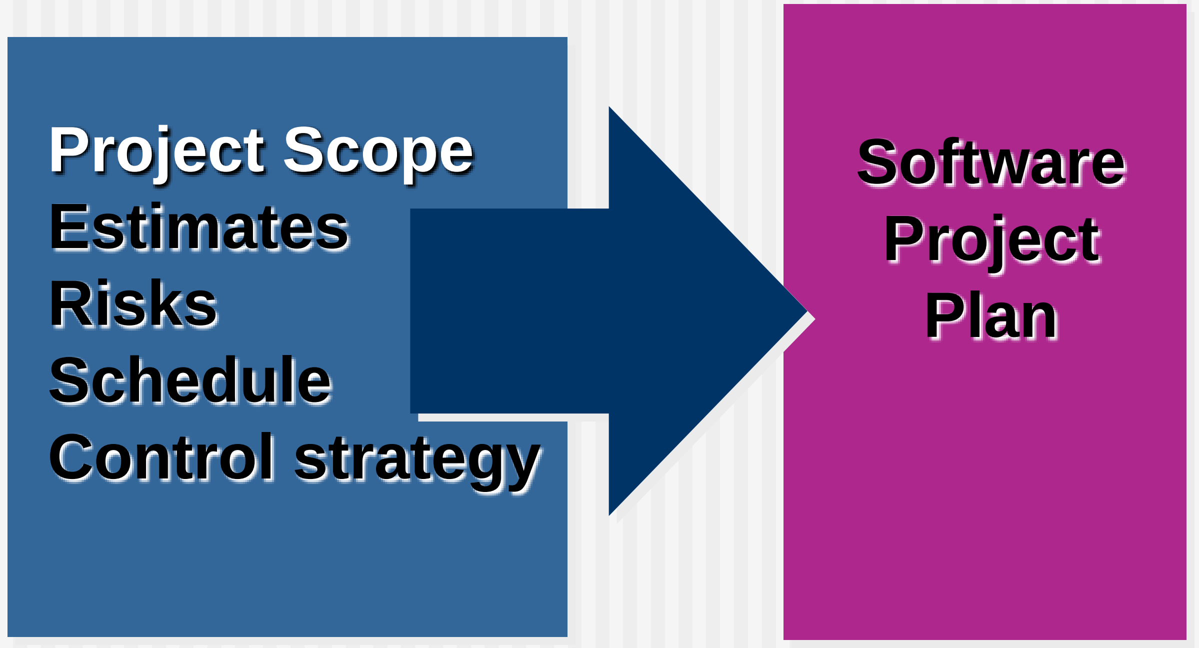
Project Planning Task Set-II

- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use-cases
  - Reconcile the estimates
- Develop a project schedule
  - Scheduling is considered in detail in Chapter 27.
    - Establish a meaningful task set
    - Define a task network
    - Use scheduling tools to develop a timeline chart
    - Define schedule tracking mechanisms

Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires
  - experience
  - access to good historical information (metrics)
  - the courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty

Write it Down!



To Understand Scope ...

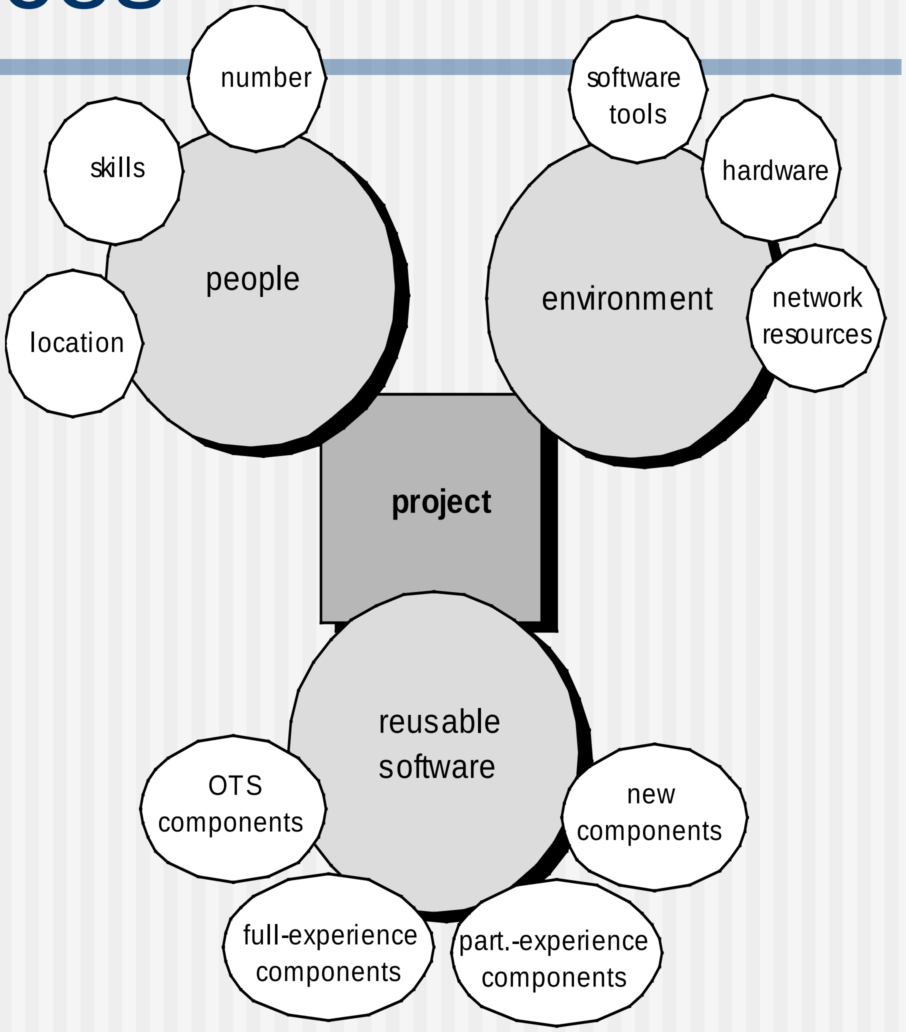
- Understand the customers needs
- understand the business context
- understand the project boundaries
- understand the customer's motivation
- understand the likely paths for change
- understand that ...

Even when you understand, nothing is guaranteed!

What is Scope?

- *Software scope* describes
  - the functions and features that are to be delivered to end-users
  - the data that are input and output
  - the “content” that is presented to users as a consequence of using the software
  - the performance, constraints, interfaces, and reliability that *bound* the system.
- Scope is defined using one of two techniques:
  - A narrative description of software scope is developed after communication with all stakeholders.
  - A set of use-cases is developed by end-users.

Resources

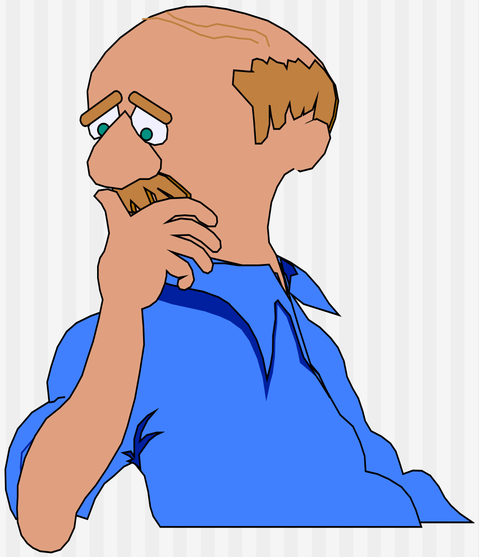

number
skills
people
location
software tools
hardware
environment
network resources
project
reusable software
OTS components
new components
full-experience components
part.-experience components

Project Estimation

- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process

Estimation Techniques

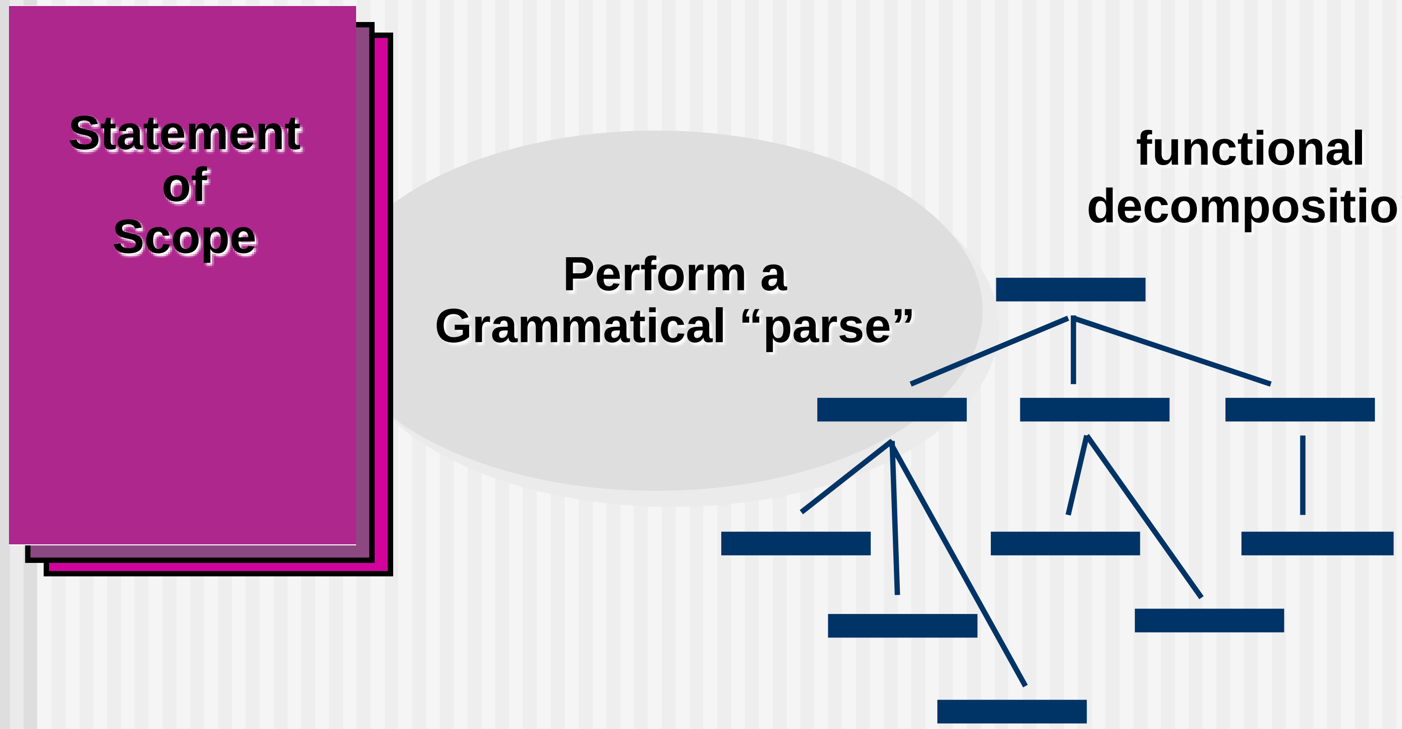
- Past (similar) project experience
- Conventional estimation techniques
  - task breakdown and effort estimates
  - size (e.g., FP) estimates
- Empirical models
- Automated tools



Estimation Accuracy

- Predicated on …
  - the degree to which the planner has properly estimated the size of the product to be built
  - the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects)
  - the degree to which the project plan reflects the abilities of the software team
  - the stability of product requirements and the environment that supports the software engineering effort.

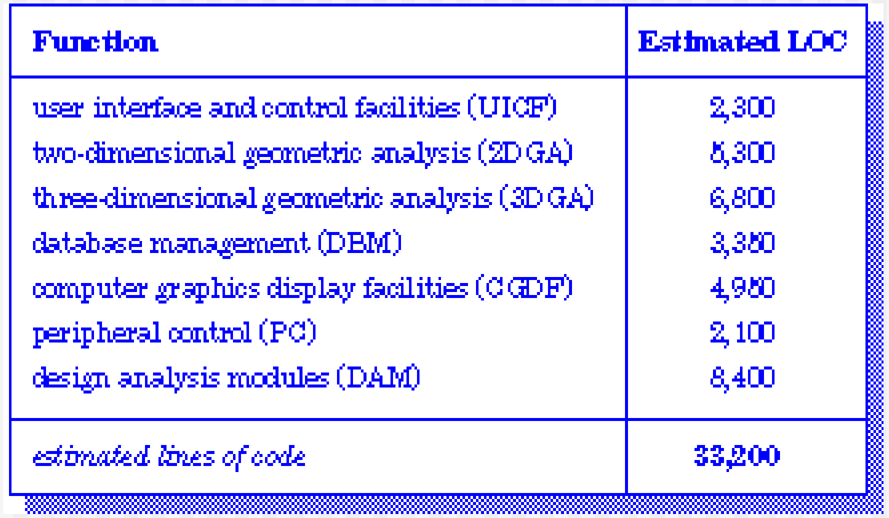
Functional Decomposition



Conventional Methods: LOC/FP Approach

- compute LOC/FP using estimates of information domain values
- use historical data to build estimates for the project

Example: LOC Approach



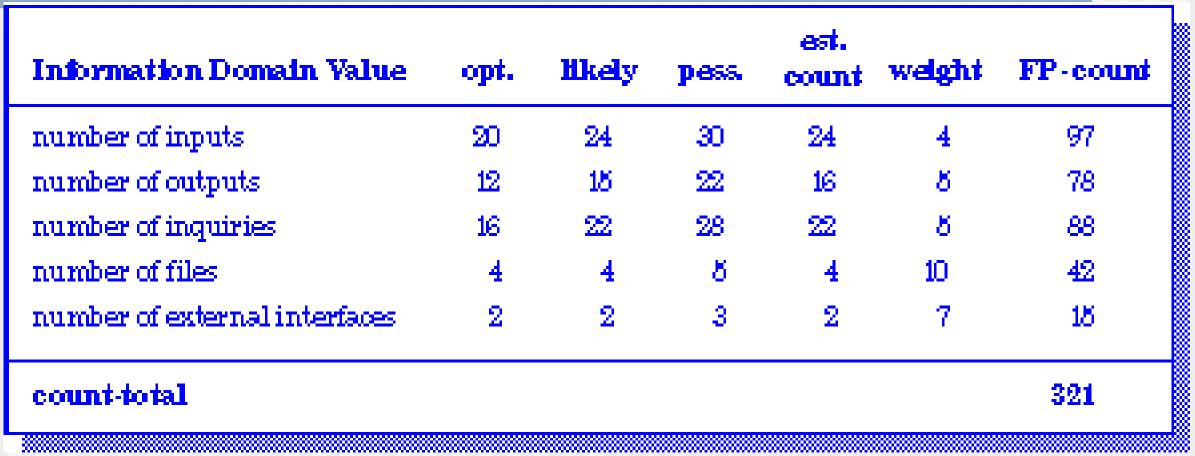
Function	Estimated LOC
user interface and control facilities (UICF)	2,300
two-dimensional geometric analysis (2DGA)	5,300
three-dimensional geometric analysis (3DGA)	6,800
database management (DBM)	3,350
computer graphics display facilities (CGDF)	4,950
peripheral control (PC)	2,100
design analysis modules (DAM)	8,400
estimated lines of code	**33,200**

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate =$8000 per month, the cost per line of code is approximately $13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is **$431,000 and the estimated effort is 54 person-months.**

Example: FP Approach



Information Domain Value	opt.	likely	pess.	est. count	weight	FP-count
number of inputs	20	24	30	24	4	97
number of outputs	12	15	22	16	5	78
number of inquiries	16	22	28	22	5	88
number of files	4	4	5	4	10	42
number of external interfaces	2	2	3	2	7	15
count-total						321

The estimated number of FP is derived:

$FP_{estimated}$ = count-total 3 [0.65 + 0.01 3 S (F_i)]

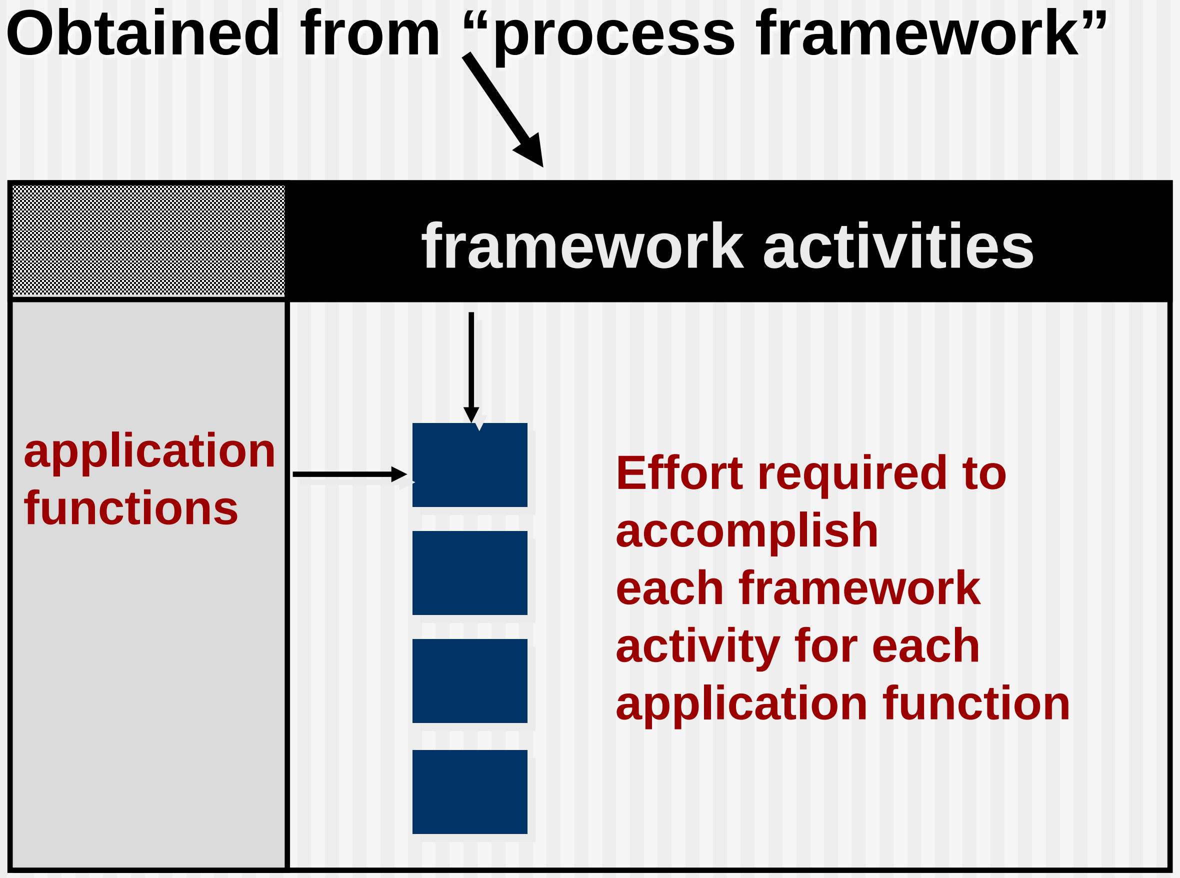
$FP_{estimated}$ = 375

organizational average productivity = 6.5 FP/pm.

burdened labor rate = $8000 per month, approximately $1230/FP.

Based on the FP estimate and the historical productivity data, **total estimated project cost is $461,000 and estimated effort is 58 person-months.**

Process-Based Estimation


Obtained from “process framework”
framework activities
application functions
Effort required to accomplish each framework activity for each application function

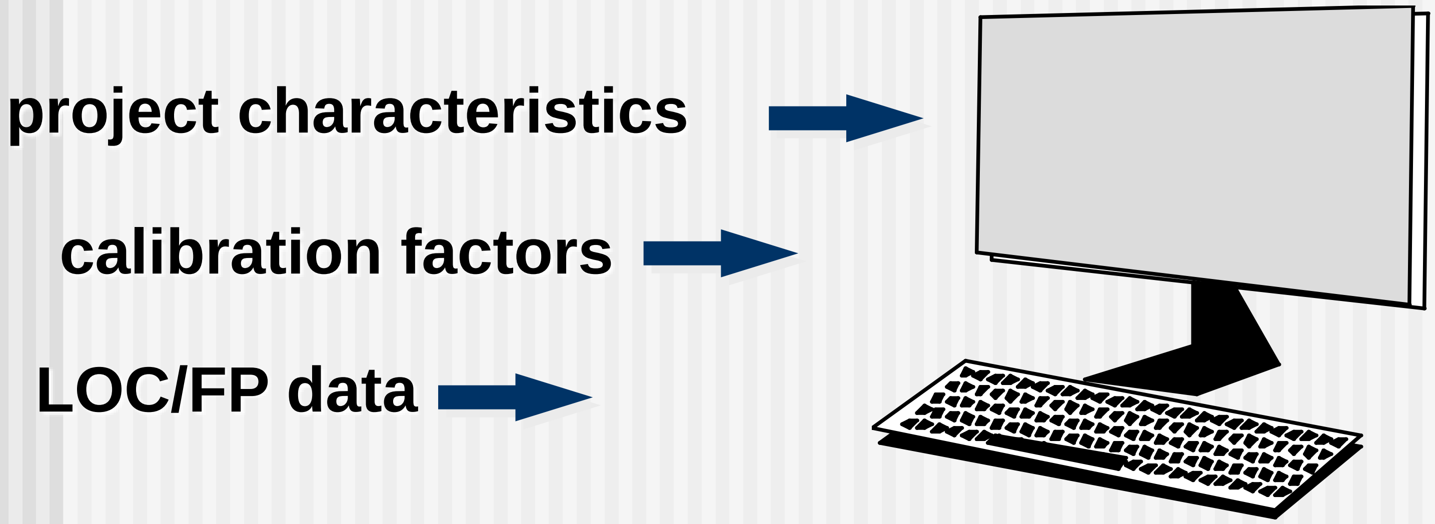
Process-Based Estimation Example

Activity →	CC	Planning	Risk Analysis	Engineering		Construction Release		CE	Totals
Task →				analysis	design	code	test		
Function									
↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DSM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

Based on an average burdened labor rate of $8,000 per month, **the total estimated project cost is $368,000 and the estimated effort is 46 person-months.**

Tool-Based Estimation



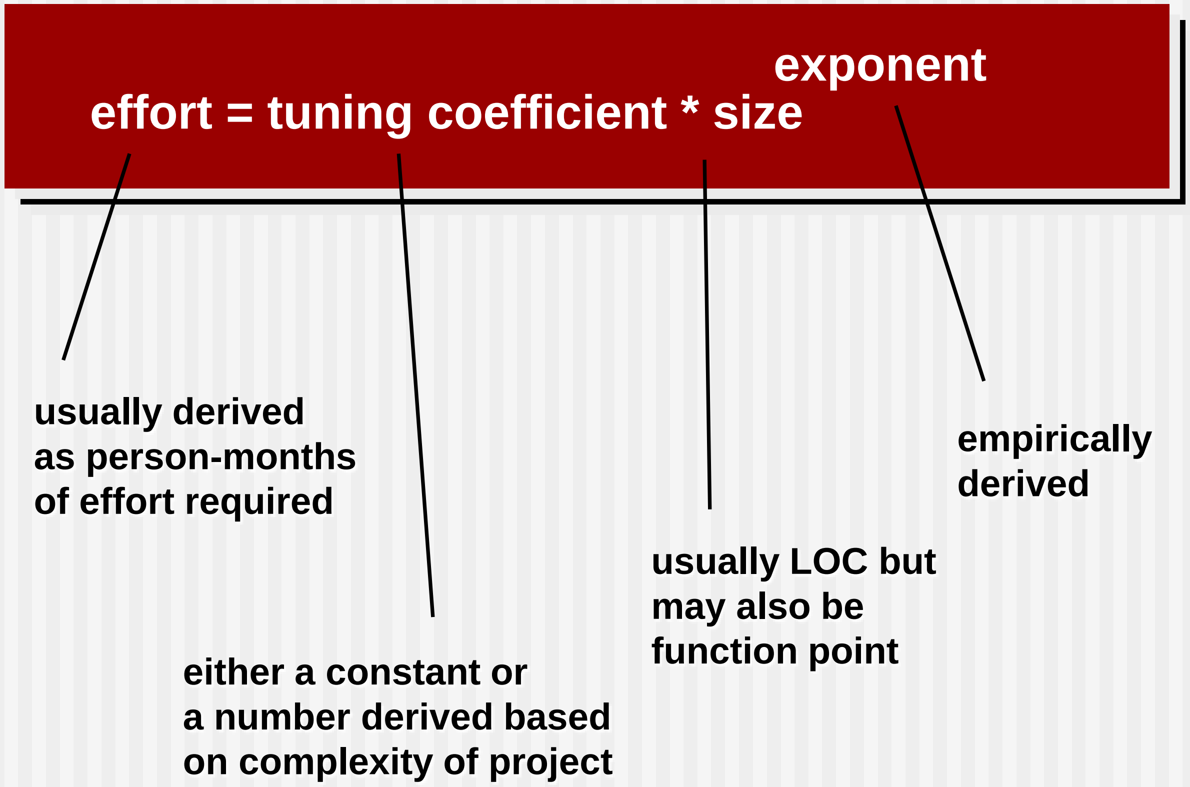
Estimation with Use-Cases

	use cases	scenarios	pages	Ê	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	Ê	12	5	560	3,366
Engineering subsystem group	10	20	8	Ê	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	Ê	10	6	1650	7,970
				Ê	Ê	Ê	Ê	
Total LOC estimate				Ê	Ê	Ê	Ê	42,568

Using 620 LOC/pm as the average productivity for systems of this type and a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the use-case estimate and the historical productivity data, **the total estimated project cost is $552,000 and the estimated effort is 68 person-months.**

Empirical Estimation Models

General form:


exponent
effort = tuning coefficient * size
usually derived as person-months of effort required
either a constant or a number derived based on complexity of project
usually LOC but may also be function point
empirically derived

COCOMO-II

- COCOMO II is actually a hierarchy of estimation models that address the following areas:
  - *Application composition model.* Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
  - *Early design stage model.* Used once requirements have been stabilized and basic software architecture has been established.
  - *Post-architecture-stage model.* Used during the construction of the software.

The Software Equation

A dynamic multivariable model

$$E = [LOC \times B^{0.333}/P]^3 \times (1/t^4)$$

where

E = effort in person-months or person-years

t = project duration in months or years

B = “special skills factor”

P = “productivity parameter”

Estimation for OO Projects-I

- Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
- Using object-oriented requirements modeling (Chapter 6), develop use-cases and determine a count.
- From the analysis model, determine the number of key classes (called analysis classes in Chapter 6).
- Categorize the type of interface for the application and develop a multiplier for support classes:

Interface type	**Multiplier**
No GUI	2.0
Text-based user interface	2.25
GUI	2.5
Complex GUI	3.0

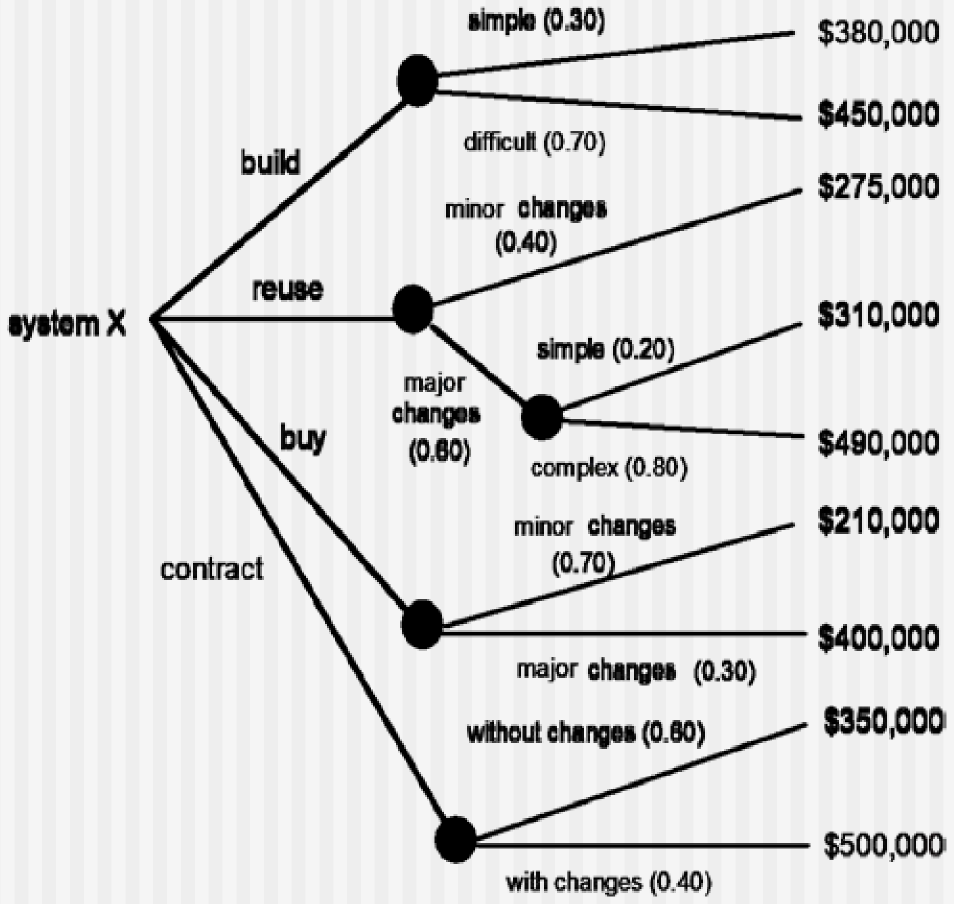
Estimation for OO Projects-II

- Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
- Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
- Cross check the class-based estimate by multiplying the average number of work-units per use-case

Estimation for Agile Projects

- Each user scenario (a mini-use-case) is considered separately for estimation purposes.
- The scenario is decomposed into the set of software engineering tasks that will be required to develop it.
- Each task is estimated separately. Note: estimation can be based on historical data, an empirical model, or “experience.”
  - Alternatively, the ‘volume’ of the scenario can be estimated in LOC, FP or some other volume-oriented measure (e.g., use-case count).
- Estimates for each task are summed to create an estimate for the scenario.
  - Alternatively, the volume estimate for the scenario is translated into effort using historical data.
- The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

The Make-Buy Decision



Computing Expected Cost

expected cost =

$$\sum \text{(path probability)}_i \times \text{(estimated path cost)}_i$$

For example, the expected cost to build is:

$\text{expected cost}_{\text{build}} = 0.30\ (\$380\text{K}) + 0.70\ (\$450\text{K})$

$= \$429\ \text{K}$

similarly,

$\text{expected cost}_{\text{reuse}} = \$382\text{K}$

$\text{expected cost}_{\text{buy}} = \$267\text{K}$

$\text{expected cost}_{\text{contr}} = \$410\text{K}$

PROBLEMS

Problem 1- wide band delphi method

Given,

Optimistic LOC, $S_{opt} = 2600$

Pessimistic LOC,$S_{pess} = 4000$

Most Likely LOC, $S_m = 3800$

Calculate the expected value for the estimation variable

Problem 1 Solution

The expected value of the estimation variable,

$S = (S_{opt} + 4 * S_{m} + S_{pess}) / 6$

$= (2600 + 4 * 3800 + 4000)/6$

$S = 3633$

Problem 2

Given

- The estimated LOC count is 56,100
- From the review of historical data, below two values were derived
  - Average productivity for this category of systems = 693 LOC/pm
  - Burdened labor rate = $8000/month

Compute the below,

- Cost per LOC
- Total Estimated Project Cost
- Estimated Effort in person months

Problem 2 Solution

– Cost per LOC = **Labor rate per month/LOC per pm**

= 8000/693 = $11.5

– Total Estimated Project Cost = **Estimated LOC * Cost per LOC**

= 56,100 * 11.5 = $6,45,150

– Estimated Effort in pm = **Total Estimated Project Cost/ Labor rate per month**

= 6,45,150/8000 = 81 person months

Function Points

Information Domain Value	Count		Weighting factor simple	average	complex		
External Inputs (EIs)	☐	3	3	4	6	=	☐
External Outputs (EOs)	☐	3	4	5	7	=	☐
External Inquiries (EQs)	☐	3	3	4	6	=	☐
Internal Logical Files (ILFs)	☐	3	7	10	15	=	☐
External Interface Files (EIFs)	☐	3	5	7	10	=	☐

Count total $\longrightarrow$ ☐

Problem 3- Function Point

Information Domain Value Count	Opt	Likely	Pess	Estimated Count	Weight Average	FP Count
Inputs	20	24	34	?	4	?
Outputs	12	17	22	?	5	?
Inquiries	16	26	33	?	5	?
Files	4	5	7	?	10	?
Interfaces	2	4	6	?	7	?
Count Total						

Given the above table, with default simple weight for all categories and the Complexity Factor = 1.19.
compute
- The estimated count for each value
- Total Unadjusted FP
- Adjusted FP Count

Problem 3 - Contd

- Also, if the below historical data are given,
  - Organizational Average Productivity of this type of system = 5.8 FP/m
  - Burdened Labor Rate = $8000/m
- Compute
  - Cost per FP
  - Total Estimated Project Cost
  - Estimated Effort in person months

Problem 3 - Solution

Information Domain Value Count	Opt	Likely	Pess	Estimated Count	Weight	FP Count
Inputs	20	24	34	24	4	96
Outputs	12	17	22	17	5	85
Inquiries	16	26	33	26	5	130
Files	4	5	7	5	10	50
Interfaces	2	4	6	4	7	28
Count Total						**389**

UFP(Unadjusted FP) = 389
VAF(*value adjustment factors*) = 0.65 + 0.01 * $\sum F_i$ = 0.65 + 0.01 * 1.19 = 0.66
AFP(Adjusted FP Count) = UFP * VAF = 389 * 0.66 = 257

Problem 3 – Solution Contd

- Cost per FP = Labor rate per month/FP per month

  = 8000/5.8 = $1379
- Total Estimated Project Cost = AFP * Cost per FP = 257 * 1379 = $ 3,54,403
- Estimated Effort in person months = Total Estimated Project Cost/Labor rate per month = 3,54,403/8000 = 44 pm

Problem 4- COCOMO II

For the given data, compute the effort in person months using the COCOMO model

1. Objects' Count

Object Type	Count of each object		
	Simple	Medium	Difficult
Screen	5	4	5
Report	6	7	5
3GL Comp			6

2. Objects' Complexity Weight

Object Type	Complexity Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Comp			10

Also given,
Percentage of reuse is 30%,
Value of PROD = 7

Compute the effort in person months.

Problem 4 - Solution

Object Type	Object Points Calculation			Count
	Simple	Medium	Difficult	
Screen	5 * 1	4 * 2	5 * 3	28
Report	6 * 2	7 * 5	5 * 8	87
3GL Comp			6 * 10	60

Value of Productivity rate

v.Low- 4	
Low-7	
Normal – 13	
High -25	
V.High -50	

Object Points Count = 28 + 87 + 60 = 175
NOP = Object Points * [(100 - % reuse)/100] = 175 * 70/100 = 122.5
Estimated Effort = NOP/PROD = 122.5/7 (LOW) = 17.5 pm

Problem 5

Using the simplified **Software Equation Model**, compute the effort in person months, given the below data,

- Estimated LOC = 33580
- Productivity Parameter, P = 12,000
- Special Skills Factor ,B = 0.28

Problem 5 Solution

Simplified Software Equation Estimation Model

$t_{min} = 8.14\ (LOC/P)^{0.43}$

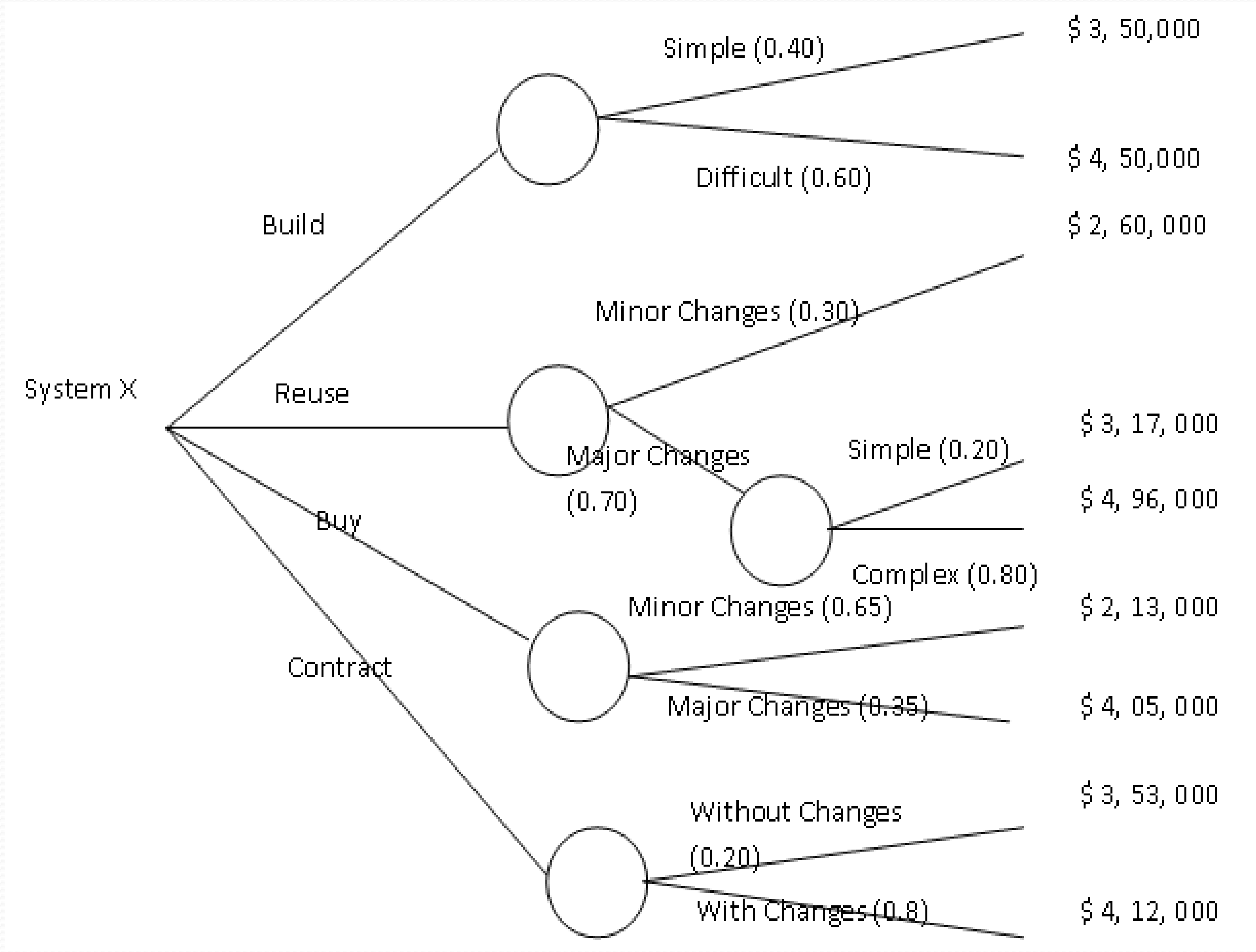
$E = 180*B*t^3$

Substituting,

$t_{min} = 8.14\ (33580/12000)^{0.43} = 8.14 *(2.8)^{0.43} = 8.14 * 1.6 =$ 13 pm = 1.08 person years

$E = 180 * 0.28 * (1.08)^3 = 63.5$ pm

Problem 6-Make /buy decision



Problem 6 - Solution

Expected Cost = $\sum$ (Path Probability$_i$ * Estimated Path Cost$_i$)

Expected Cost$_{build}$ = 1, 40, 000 + 2, 70, 000

= $ 4, 10, 000

Expected Cost$_{reuse}$ = 78000 + 0.70 [0.20 * 3,17,000 + 0.80 * 4, 96,000] = $ 4, 00, 140

Expected Cost$_{buy}$ = 1, 38, 450 + 1, 41, 750 = $ 2, 80, 200

Expected Cost$_{contract}$ = 70600 + 329600 = $ 4, 00, 200

The decision would be to buy the software.

Problem 7

Given,

Probability of occurrence of a risk, P = 0.5

Cost to the project if the risk occurs,

C = $ 20, 600

Calculate Risk Exposure

Problem 7 - Solution

Risk Exposure = P * C

= 0.5 * 20, 600

= $10, 300

Problem 8

Adaptation Criteria	Grade	Weight	New Dev – Entry Point Multiplier
Size	2	1.2	1
No of Users	3	1.1	1
Business Criticality	4	1.1	1
Longevity	3	0.9	1
Req. Stability	4	1.2	1
Ease of Communication	2	0.9	1
Maturity of Tech	2	0.9	1
Perf Constraints	3	0.8	1
Embedded/Non Emb.	3	1.2	1
Project Staffing	1	1.0	1
Interoperability	3	1.1	1
Engg. Factors	0	1.2	0

Given, the below adaptation criteria, graded for a new dev project, with low risk, Calculate TSS and conclude on the degree of rigor that you choose for applying process.

Problem 8 - Solution

Adaptation Criteria	Grade	Weight	New Dev – Entry Point Multiplier	TTS
Size	2	1.2	1	2.4
No of Users	3	1.1	1	3.3
Business Criticality	4	1.1	1	4.4
Longevity	3	0.9	1	2.7
Req. Stability	4	1.2	1	4.8
Ease of Communication	2	0.9	1	1.8
Maturity of Tech	2	0.9	1	1.8
Perf Constraints	3	0.8	1	2.4
Embedded/Non Emb.	3	1.2	1	3.6
Project Staffing	1	1.0	1	1
Interoperability	3	1.1	1	3.3
Engg. Factors	0	1.2	0	0

Average TSS = 2.7

TSS < 1.24 – Casual
TSS between 1 and 3 – Structured
TSS > 2.4 - Strict

So structured, application is chosen

Problem 9

Task ID	Planned Effort	Actual Effort	Scheduled Cost	Actual Cost
1	17	18	1700	1800
2	10	12	1000	1200
3	15	13	1500	1300
4	16	19	1600	1900
5	11	-	1100	-
6	9	-	900	-
..	…	..	..	..
..	…	..	..	..

Given, the above project table, while you are asked to perform the EVA, 6 tasks should have been completed as per schedule. But only 4 tasks have bee completed. Calculate Scheduled Performance Index (SPI), ScheduledVariance (SV), Percent scheduled for complete, Percent Complete, Cost Perfprmance Index(CPI), Cost Variance(CV), Total budget is $15, 000

Problem 9 - Solution

- *budgeted cost of work scheduled(*BCWS) = $ 7800
- *budgeted cost of work performed(*BCWP)= $ 5800
- Actual cost of work performed(ACWP) = $ 6200
- Scheduled performance index(SPI) = BCWP/BCWS
  = 5800/7800 = 0.75
- Scheduled Variance (SV) = BCWP – BCWS
  = 5800 – 7800 = -2000
- Percent Scheduled = BCWS/BAC = 52%
- Percent Complete = BCWP/BAC = 39%
- CPI = BCWP/ACWP = 0.93
- CV = BCWP – ACWP = -400