University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

هندسة برمجيات متقدمة

م. م. سامر رعد

cs.uotechnology.edu.iq

First course 2023-2024

Chapter One

Software Project Planning

Topics:

1.1 Introduction
1.2 Estimation Reliability Factors
1.3 Project Planning Objectives
1.4 Software Scope
1.5 Estimation of Resources
1.6 Software Project Estimation Options
1.7 Decomposition Techniques
1.8 Estimation Models
1.8.1 The Structure of Estimation Models
1.8.2 The COCOMO Model
1.8.3 The CoCoMo Model types
1.9. Automated Estimation Tools

1.1 Introduction

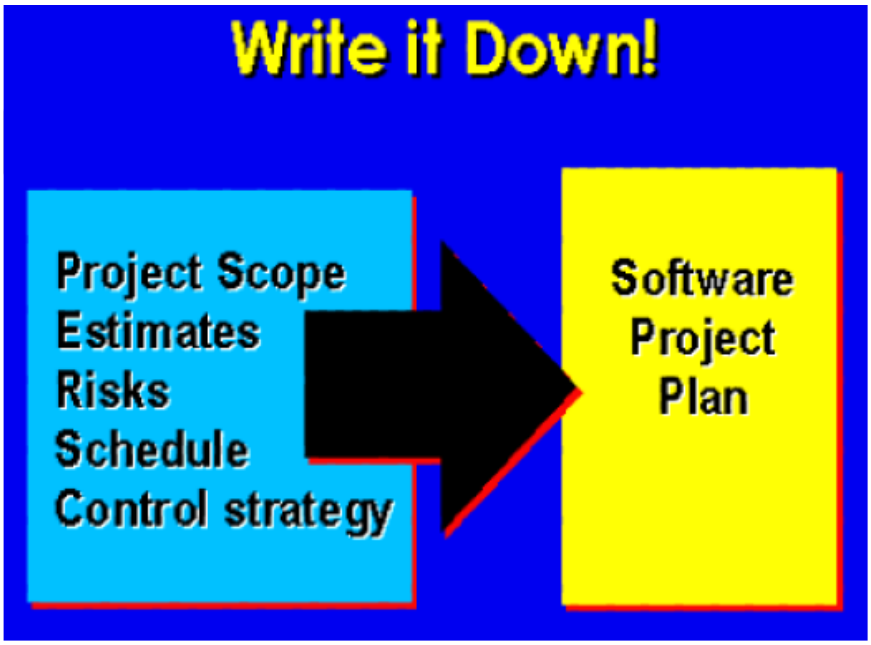
- Software planning involves estimating how much time, effort, money, and resources will be required to build a specific software system. After the project scope is determined and the problem is decomposed into smaller problems, software managers use historical project data (as well as personal experience and intuition) to determine estimates for each. The final estimates are typically adjusted by taking project complexity and risk into account. The resulting work product is called a project management plan.
- Why Planning a project ?
- Projects become more unpredictable .
- When working on a project, there are times when we run into issues that require us to redo parts of what we've already completed. This can be a problem because it can cause delays and extra work.
- We need to figure out what things rely on each other the most.
- We must reduce the problems caused by last-minute changes and surprises.
- We should organize tasks to deal with risks promptly.
- Arrange tasks to uncover assumptions and unknowns at the beginning of projects.

1.2 Estimation Reliability Factors

- Project complexity .
- Project size .
- Degree of structural uncertainty (degree to which requirements have solidified, the ease with which functions can be compartmentalized, and the hierarchical nature of the information processed) .
- Availability of historical information.
-

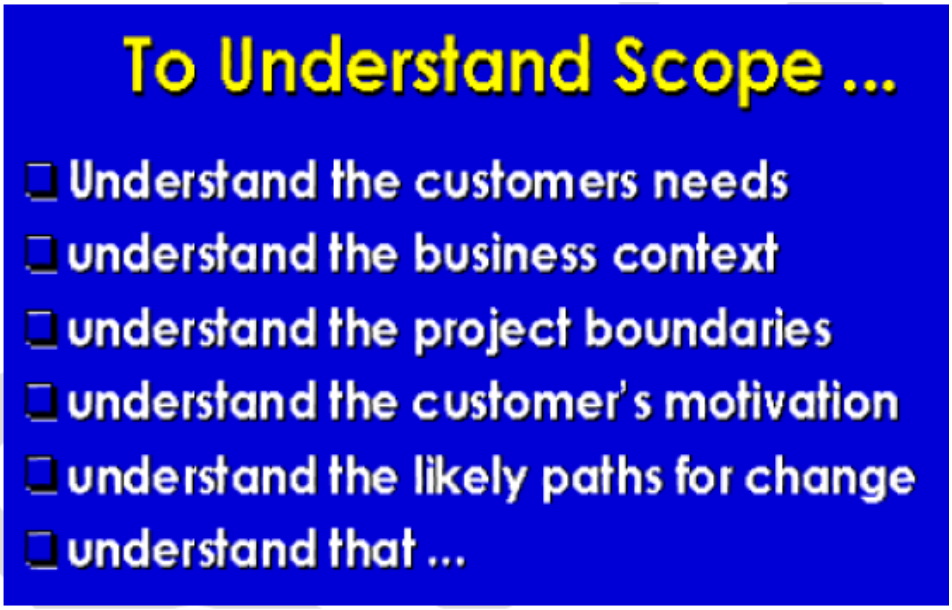
1.3 Project Planning Objectives

- To provide a framework that enables software manager to make a reasonable estimate of resources, cost, and schedule.
- Project outcomes should be bounded by 'best case' and 'worst case' scenarios.
- Estimates should be updated as the project progresses
- The overall goal of project planning is to establish a pragmatic strategy for controlling , tracking and monotoring a complex technical project.
  Why ?
  So the end result gets done on time , with quality .
  How ?
- Scoping – understanding the problem and the work that must be done
- Estimation – how much effort? How much time ?
- Risk – what can go wrong? How can we avoid it ? what can we do about it ?
- Schedule – how to allocate resources along the time line ? what are the milestones ?.
- Control strategy .

- 

1.4 Software Scope (Range)

- -Describes the data to be processed and produced, control parameters, function, performance, constraints, external interfaces, and reliability.
- -Often functions described in the software scope statement are refined to allow for better estimates of cost and schedule.



1.5 Estimation of Resources

1. Human Resources (number of people required and skills needed to complete the development project)
2. Reusable Software Resources (off-the-shelf components, full-experience components, partial-experience components, new components)
3. Development Environment (hardware and software required to be accessible by software team during the development process).

1.6 Software Project Estimation Options

1. Delay estimation until late in the project.
2. Base estimates on similar projects already completed.
3. Use simple decomposition techniques to estimate project cost and effort.
4. Use empirical models for software cost and effort estimation.
5. Automated tools may assist with project decomposition and estimation.

1.7 Decomposition Techniques

Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece. For this reason, we decompose the problem, recharacterizing it as a set of smaller (and hopefully, more manageable) problems. There
are several types of decomposition techniques, these are:

1.Software sizing (fuzzy logic, function point, standard component, change)
2. Problem-based estimation (using LOC decomposition focuses on software functions, using FP decomposition focuses on information domain characteristics)
3. Process-based estimation (decomposition based on tasks required to complete the software process framework).

1.8 Estimation Models

1. The Structure of Estimation Models, typically derived from regression analysis of historical software project data with estimated person-months as the dependent variable and KLOC or FP as independent variables.
2. Constructive Cost Model (COCOMO) is an example of a static estimation model.
3. The Software Equation is an example of a dynamic estimation model.

1.8.1 The Structure of Estimation Models

A typical estimation model is derived using regression analysis on data collected from past software projects. The overall structure of such models takes the form [MAT94]
E=A+B *(E*V*) (1.1)
where *A, B,* and *C* are empirically derived constants, *E* is effort in person-months, and *ev* is the estimation variable (either LOC or FP). In addition to the relationship noted in

Equation (1.1), the majority of estimation models have some form of project adjustment component that enables *E* to be adjusted by other project characteristics (e.g., problem complexity, staff experience, development environment).

1.8.2 The COCOMO Model

It comes from Constructive Cost Model, CoComo is used for calculating :
1-the effort
2-development time
3-Average Staff size
4-productivity
CoCoMo Includes the following types:
-Basic Cocomo Model
-Intermediate CoCoMo Model
-Complete /detailed CoCoMo Model
So these 3 different types are included in the CoCoMo Model
Basic : can be used for small projects and Small team
Intermediate : is used for Intermediate projects
Complete / detailed :is used for large projects
CoCoMo applied on 3 classes of S. w. projects or modes
What are these classes
1-organic mode
2-Semidetached mode
3-embedded mode
Basic, intermediate and detailed can be applied on the above classes.
Basic CoCoMo Model
Basic CoCoMo Model
The formula to calculate the basic Model are
E = ab (kLoc) bb
D=Cb (E)db
E = effort applied
D= Development time
people required = P=E/D
they are constant value ab, bb, Cb, db are coefficients.

Coefficient values

Software project	a_b	bb	C_b	db
Organig	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
embedded	3.6	1.20	2.5	0.32

Example

•example: Supposthat a project was estimated tobe 400k Loc, Calculate the effort and development time for each of the a modes organic , semidetached and embeded

Solution:The basic Cocomoequation take the form

E = ab(kLoc) bb
D=Cb(E)db
estimated size of the project =400k Loc
Example
organic mode
E=2.4 (400) 1.05=1295,31
D=2.5 (1295.31.) 0.38 = 38, 07
Semidetached
E=3.0 (400) 1.12 =2462.79
D=2.5(2462.79)0.35=38.45
Embedded :
E=3.6(400)1.20 =4772.81
D=2.5(4772.81)0.32=38
Intermediate CoCoMoModel
Cost driver is rated for the given project Environment in the term of
l-very low < 1
2-low< 1

3 –High >1
4-very High> 1
5-Extra High >1
Cocomoequation take the form
E = ab(kLoc) bb* EAF
D=Cb(E)db
EAF = effort adjustment factor
•EAF : can be a calculated by multiplying all the values that have been obtained after categorizing each cost driver Cost driver.

Coefficient values

Software project	a_b	bb	C_b	db
Organig	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
embedded	2.8	1.20	2.5	0.32

•Suppose that a project was estimated to be 300kloc and EAF=2 or (very high) calculate the effort and development time for each of the three modes organic ,semidetached and embedded

•Sol :
Organic :
E = ab(kLoc) bb* EAF =3.2 (300) 1.05 * 2 =
D=Cb(E)db= 2.5 (E) 0.38 =
•Semidetached :
E = ab(kLoc) bb* EAF =3.0 *(300)1.12 * 2
D=Cb(E)db= 3.0 (E) 0.35 =
Embedded :
E = ab(kLoc) bb* EAF = 2.8 (300)1.20 * 2 =

D=Cb(E)db= 2.5 (E) 0.32 =

1.9 Automated Estimation Tools

The decomposition techniques and empirical estimation models described in the preceding sections are available as part of a wide variety of software These automated estimation tools allow the planner to estimate cost and effort and to perform "what-if" analyses for important project variables such as delivery date or staffing. Although many automated estimation tools exist, all exhibit the same general characteristics and all perform the following six generic functions .

1. **Sizing of project deliverables.** The "size" of one or more software work products is estimated. Work products include the external representation of software (e.g., screen, reports), the software itself (e.g., KLOC), functionality delivered (e.g., function points), descriptive information (e.g. documents).
2. **Selecting project activities.** The appropriate process framework is selected and the software engineering task set is specified.
3. **Predicting staffing levels.** The number of people who will be available to do the work is specified. Because the relationship between people available and work (predicted effort) is highly nonlinear, this is an important input.
4. **Predicting software effort.** Estimation tools use one or more models.
5. **Predicting software cost.** Given the results of step4, costs can be computed by allocating labor rates to the project activities noted in step2.
6. **Predicting software schedules.** When effort, staffing level, and project activities are known, a draft schedule can be produced by allocating labor across software engineering activities.

When different estimation tools are applied to the same project data, a relatively large variation in estimated results is encountered. More important, predicted values sometimes are significantly different than actual values.

Chapter TWO

Analysis Concepts and Principles

Topics:

2.1 Introduction

2.2 Requirements Analysis

2.3 Software Requirements Analysis Phases

2.4 Software Requirements Elicitation

2.1 Introduction

After system engineering is completed, software engineers need to look at the role of software in the proposed system. Software requirements analysis is necessary to avoid creating software product that fails to meet the customer's needs.
Data, functional, and behavioral requirements are elicited from the customer and refined to create specification that can be used to design the system. Software requirements work products must be reviewed for clarity, completeness, and consistency.

2.2 Requirements Analysis

• Software engineering task that bridges the gap between system level requirements engineering and software design.
• Provides software designer with a representation of system information, function, and behavior that can be translated to data, architectural, components level design.
• Expect to do a little bit of design during analysis and a little bit of analysis during design.

Software Requirements Analysis

• Identify the "customer" and work Together to negotiate "product –level"
• Build an analysis model
-Focus on data
- define function
- represent behavior
• Prototype areas of uncertainty
• Develop a specification that will guide design
• Conduct formal technical reviews

2.3 Software Requirements Analysis Phases

• Problem recognition
• Evaluation and synthesis (focus is on what not how)
• Modeling
• Specification
• Review

2.4 Software requirements elicitation استنباط

• Customer meetings are the most commonly used technique.
• Use context free question to find out customers goal and benefits, identify stakeholders, gain understanding of problem, determine, customer reactions to proposed solution, and assess meeting effectiveness.
• If many users are involved, be certain that a representative cross section of users is interviewed.

2.4.1 Facilitated action Specification Techniques (FAST)

• Meeting held at neutral site, attended by both software engineering and customers.
• Rules established for preparation and participation.
• Agenda suggested to cover important points and to allow for brainstorming.
• Meeting controlled by facilitator (customer, developer, or outsider).
• Definition mechanism (flip charts, stickers, electronic device, etc.) is used.
• Goal is to identify problem, propose elements of solution, and negotiate different approaches, and specify a preliminary set of solution requirements.

Fast Guidelines

• Participants must attend entire meeting
• All participants are equal
• Preparation is as important as meeting
• All pre-meeting documents are to be viewed as " proposed"
• Off-sit meeting location is preferred
• Set an agenda and maintain it
• Don’t get mired in technical

2.4.2 Quality Function Deployment (QFD)

• Translates customer needs into technical software requirements.
• Uses customer interviews, observation, surveys, and historical data for requirements gathering.
• Customer voice table (contains summary of requirements)
• Normal requirements (must be present in product for customer to be satisfied)
• Expected requirement (things like ease of use or reliability of operation, that customer assumes will be present in a professionally developed product without having to request them explicitly)
• Exciting requirements (features that go beyond the customers expectations and prove to very satisfying when they are present)
• Function deployment (used during customer meeting to determine the value of each function required for system)
• Information deployment (identifies data objects and events produced and consumed by the system)
• Task deployment (examines the behavior of product within in environment)

• Value analysis (used to determine the relative priority of requirements during function , information, and task deployment) .

Quality Function Deployment

QFD is used to determine the following entities:

• Function deployment determines the value (as perceived by the customer)of each function required of the system

• Information deployment identifies data objects and events

• Task deployment examines the behavior of the system

• Value analysis determines the relative priority of requirements

2.4.3 Use - case

Use-cases definition and purpose

• A collection of scenarios that describe the thread of usage of a system

• Each of an "actor" – a person or device that interacts with the software in some way

• Each scenario answers the following questions :

- What are the main tasks of functions performed by the actor?
- What system information will the actor acquire, produce or Change
- Will the actor inform the system about environmental changes?
- What information does the actor require of the system?
- Does the actor wish to be informed

Use case features:

• Scenarios that describe how the product will be used in specific situations.

• Written narratives that describe the role of an actor (user of device) as interaction with the system occurs.

• Use-cases are designed from the actor's point of view.

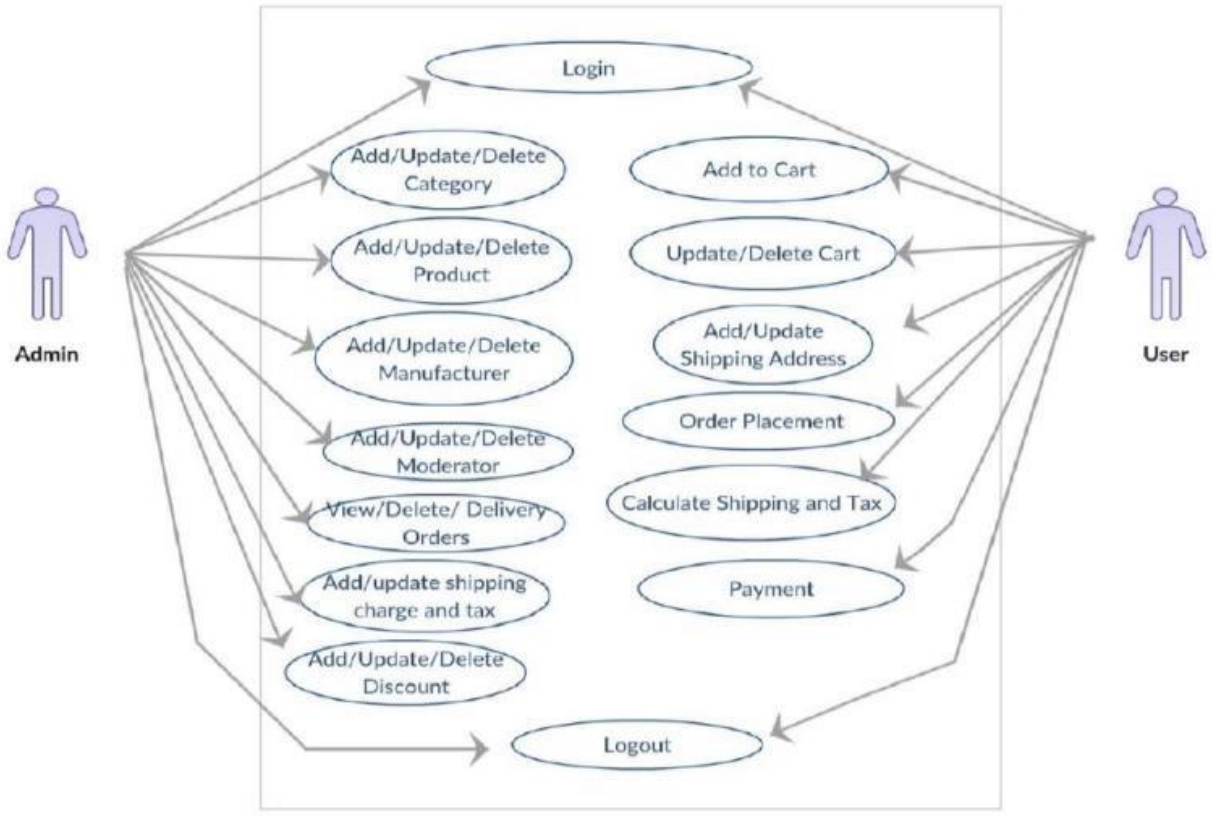
• Not all actors can be identifying the primary actors before developing the use cases.

Case study

Use Case Description

Use case interacts between user and system without specify the user interface. It describe the system for
external user and in a manner that can be understand.
List of use cases are:

- Registration
- Login
- Select Category
- Add to Cart
- List the Items
- Order Item
- Final Price with Quantity
- Authorization
- Security
- Feedback



Registration Use Case

Use Case Name	Registration
Primary Actor	Everyone
Description	Registration the System
Precondition	Users fill up all required field
Post condition	Users are Registered

Login Use Case

Use Case Name	Login
Primary Actor	Registered users
Description	Login into the System
Precondition	All authorized has the user name and password and verified by the system
Post condition	System accept the User

Select Category Use Case

Use Case Name	Item Selection
Primary Actor	General Users
Description	Select item from menu
Precondition	All users, registered or unregistered can select any item for order products
Post condition	System show the options, menu category

List the Item Use Case

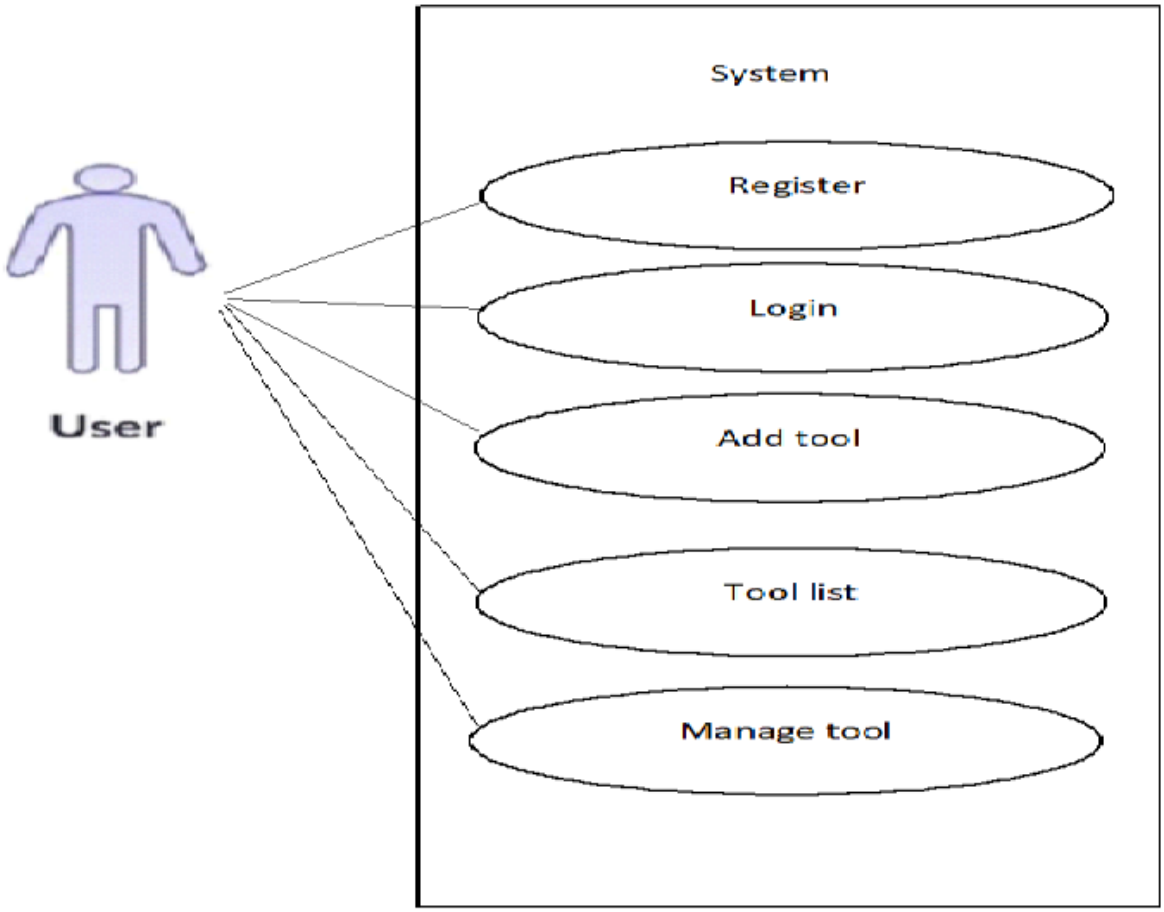
Use Case Name	List the item
Primary Actor	Selected item will be listed
Description	List item with quantity and price
Precondition	Any user can add items
Post condition	System will create a database for user

Authorization Use Case

Use Case Name	Authorization
Primary Actor	System Admin
Description	System will check users address, email, phone number etc is valid or not
Precondition	User must enter address, email and phone number
Post condition	System will automatically place the order

Case study 2

• Build a system that allows any one to sell his tool by taking picture to the tool and describe it using phone based app. The user should have account to use this system and he could set up the limit of cities that he want to look for buy tool he will search for any tool he want to buy and the system will display available tools in his area .



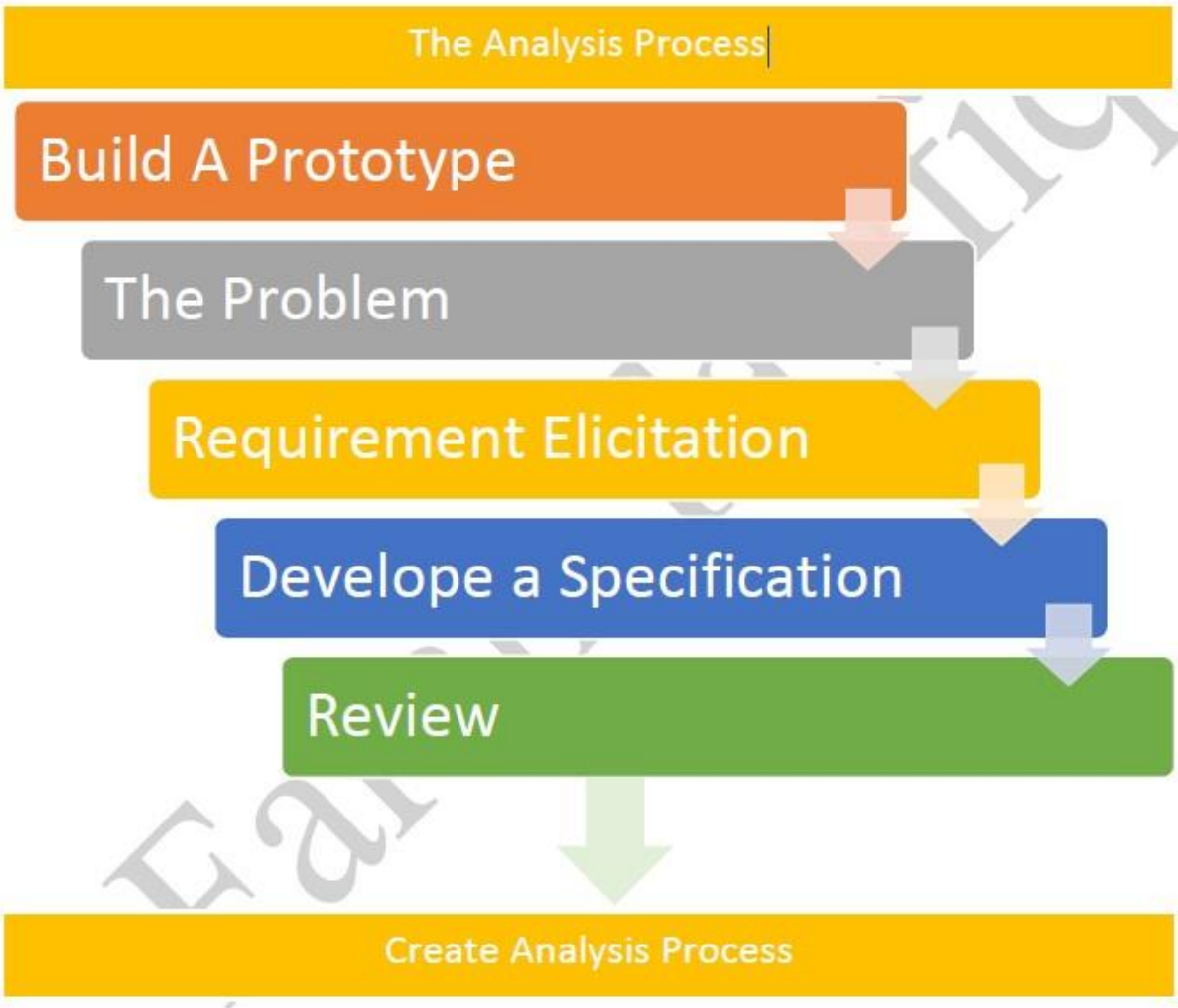
Registration for use case

Use case number	01
Use case name	Register
Overview	User register new account into the system
Actor(s)	User
Pre-condition	- System is uploaded to the internet - App installed on his (user)phone
Scenario flow	1. Open app 2. Click register 3. Enter his name , email ,phone number , and address 4. Click register 5. Display message “you are registered ”
Alternative flows	5a- display message “ user name or password is reserved “
Post condition	Display login page

2.5 Analysis Principles

• The information domain of the problem must be represented and understood.
• The functions that the software is to perform must be defined.

• Software behavior must be represented.
• Models depicting information, function, and behavior most be partitioned in a hierarchal manner detail.
• The analysis process should move from the essential information toward implementation details.
The analysis process is explained



2.5.1 Information Domain

• Encompasses all data objects that contain numbers, text, images, audio, or video
• Information content or data model (shows the relationships among the data and control objects that make up the system)
• Information flow (represents the manner in which data control objects

change as each moves through the system)
• Information structure (representations of the internal organizations of various data and control items)

2.5.2 Modeling

• Data model (shows relationships among system objects)

•Function model (description of the functions that enable the transformation of system objects)
• Behavioral model (manner in which software responds to events from the outside world.

Analysis Principle Model - The Data Domain

• Define data object
• Describe data attributes
• Establish data relationships

Analysis Principle – The Model Function

• Identify functions that transform data objects
• Indicate how data flow through the system
• Represent producers and consumers of data

Analysis Principle Model Behavior

• Identify deferent states of the system
• Specify events that cause the system to change state

2.5.3 Partitioning

• Process that result in the elaboration of data, function, or behavior.
• Horizontal partitioning is a breadth –first decomposition of the system function, behavior, or information, one level at a time.
• Vertical portioning is a depth – first elaboration of the system function, behavior, or information, one subsystem at a time.

2.6 Software Prototyping

• Throwaway prototyping (prototype only used as a demonstration of product requirements, finished software is engineered using another paradigm)
• Evolutionary prototyping (prototype is refined to build the finished system)
• Customer resources must be committed to evaluation and refinement of the prototype.
• Customer must be capable of making requirements decisions in a timely manner.

2.6.1 Prototyping Methods and Tools

• Fourth generation techniques (4 GT tools allow software engineer to generate executable code quickly)
• Reusable software components (assembling prototype from a set of existing software components)
• Formal specification and prototyping environments (can interactively create executable programs from software specification models)

2.7 Specification Principles

• Separate functionality from implementation.
• Develop a behavioral model that describes functional responses to all system stimuli.
• Define the environment in which the system operates and indicate how the collection of agents will interact with it.
• Create a cognitive model rather than an implementation model.
• Recognize that the specification must be extensible and tolerant of incompleteness.
• Establish the content and structure of a specification so that it can be changed easily.

Chapter Three
Risk Analysis and Management

3.1 Introduction

• A series of steps that help a software team to understand and manage uncertainty.
• A risk is a potential problem—it might happen, it might not. But, regardless of the outcome, it's a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan should the problem actually occur.
• Everyone involved in the software process—managers, software engineers, and customers—participate in risk analysis and management.

What are the steps?

 Recognizing what can go wrong is the first step, called "risk identification."

 Risk is analyzed to determine the likelihood that it will occur and the damage that it will do if it does occur.

 Once this information is established, risks are ranked, by probability and impact.

 A contingency plan is developed to manage those risks with high probability and high impact.

3.2 Software Risks

1) There is general agreement that risk always involves two characteristics:
• ***Uncertainty*** -- the risk may or may not happen.
• ***Loss*** -- if the risk becomes a reality, unwanted consequences or losses will occur).
2) Different types/categories of risks are considered:
• ***Project risks*** -- threaten the project plan.
• ***Technical risks*** -- threaten product quality and the timeliness of the schedule.
• ***Business risks*** -- threaten the viability of the software to be built (market risks, strategic risks, management risks, budget risks).
• ***Known risks*** -- predictable from careful evaluation of current project plan and those extrapolated from past project experience.
• ***Unknown risks*** -- some problems simply occur without warning.

3.4 Risks Identification

• Is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.).
• One method for identifying risk is to create *risk item checklist*. The check list can be used for risk identification and focuses on known and predictable risks in the following generic subcategories:
1. ***Product size*** -- risks associated with the overall size of the software to be built or modified.
2. ***Business impact*** -- risks associated with constraints imposed by management or the marketplace.
3. ***Customer characteristics*** -- risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
4. ***Process definition*** -- risks associated with the degree to which the software process has been defined and is followed by the development organization.
5. ***Development environment*** -- risks associated with the availability and quality of the tools to be used to build the product.
6. ***Technology to be built*** -- risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
7. ***Staff size and experience*** -- risks associated with the overall technical and project experience of the software engineers who will do the work.

3.5 Risks Projection

1. Risk projection, also called ***risk estimation***, attempts to rate each risk in two ways:

• The likelihood or probability that the risk is real.

• The consequences of the problems associated with the risk, should it occur.

2. There are four risk projection steps:

• Establish a scale that reflects the perceived likelihood of a risk

- Delineate the consequences of the risk
- Estimate the impact of the risk on the project and the product
- Note the overall accuracy of the risk projection so that there will be nomisunderstandings.

3.6 Building Risk Table

• A ***risk table*** provides a project manager with a simple technique for risk projection.

• List all risks in the first column of the table.

• Classify each risk and enter the category label in column two.

• Determine a probability for each risk and enter it into column three. Enter the severity of each risk (negligible, marginal, critical, and catastrophic) in column four.

• Sort the table by probability and impact value.

Determine the criteria for deciding where the sorted table will be divided into the first priority concerns and the second priority concerns. Table 1 shows Sample risk table prior to sorting

Risks	Category	Probability	Impact
Size estimate may be significantly low	PS	60%	2
Larger number of users than planned	PS	30%	3
Less reuse than planned	PS	70%	2
End-users resist system	BU	40%	3
Delivery deadline will be tightened	BU	50%	2
Funding will be lost	CU	40%	1
Customer will change requirements	PS	80%	2
Technology will not meet expectations	TE	30%	1
Lack of training on tools	DE	80%	3
Staff inexperienced	ST	30%	2
Staff turnover will be high	ST	60%	2
•			
•			
•			

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Catastrophic = كارثي
Critical = حرج
Marginal = ثانوي او هامشي
Negligible = ضئيلة

3.7 Risks Refinement

❖ During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk.

❖ Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor, and manage.

❖ CTC (condition-transition-consequence) format may be a good representation for the detailed risks (e.g. given that <condition> then there is a concern that (possibly) <consequence>).

❖ The consequences associated with refined sub-conditions helps to isolate the underlying risks and might lead to easier analysis and response.

3.8 Risk Mitigation, Monitoring, and Management

All of the risk analysis activities presented to this point have a single goal – to assist the project team in developing a strategy for dealing with risk. An effective strategy considers three issues: avoidance, monitoring, management and contingency planning.

• *Risk mitigation* (proactive planning for risk avoidance).

• *Risk monitoring* (assessing whether predicted risks occur or not, ensuring risk aversion steps are being properly applied, collect information for future risk analysis, attempt to determine which risks caused which problems).

• *Risk management and contingency planning* (actions to be taken in the event that mitigation steps have failed and the risk has become a live problem).

Chapter Four
Project Scheduling and Tracking

4.1 Introduction

4.1.1 What is Project scheduling and tracking?

Selecting an appropriate process model, identifying software engineering tasks that have to be performed, estimation the amount of work and number of people, knowing the deadline, considering risks. Once identified. It's time to create a network to SE tasks that will enable you to get the job done. Assign responsibility for each task, and make sure it gets done.

4.1.2 Why it's important?

To build complex software systems, many engineering tasks need to occur in parallel with one another to complete the project on time. The output from one task often determines when another may begin; these interdependencies are very difficult to understand without a schedule.

4.2 Basic Concepts

There are several reasons why software projects are not completed on time: (Why Are Projects Late?)

1. An unrealistic deadline established by someone outside the software engineering group.
2. Changing customer requirements that are not reflected in schedule changes.
3. The amount of effort and/or the number of resources that will be required to do the job.

4. Predictable and/or unpredictable risks that were not considered when the project commenced.
5. Technical difficulties that could not have been foreseen in advance.
6. Human difficulties that could not have been foreseen in advance.
7. Miscommunication among project staff that result in delay.
8. A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.

4.3 Software Project scheduling principles

• The project manager's objective is to define all project tasks, build a network that depicts their
interdependencies, identify the tasks that are critical within the network, and then track their progress to ensure that delay is recognized.

• • Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

• • It is important to note, that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedules identifies all major software engineering activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here,
Specific software tasks (required to accomplish an activity) are identified and scheduled.

4.4 A number of basic principles guide software project scheduling (developing a schedule):

1. **Compartmentalization** - the product and process must be decomposed into a manageable number of activities and tasks.
2. **Interdependency** - tasks that can be completed in parallel must be separated from those that must completed serially, other activities can occur independently.
3. **Time allocation** - every task has start and completion dates that take the task interdependencies into account. And each task to be scheduled must be allocated some number of work units (e.g., person-days of effort).
4. **Effort validation** – every project has a defined number of staff members. As time allocation occurs, the project manager must ensure that on any given day there are enough staff members assigned to completed the tasks within the time estimated in the project plan.
5. **Defined Responsibilities** - every scheduled task needs to be assigned to a specific team member.

6. **Defined outcomes** - every task in the schedule needs to have a defined outcome (usually a work product or deliverable, for example the design of a module).
7. **Defined milestones** – every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products from an engineering task have passed quality review.

Scheduling Principles

- **compartmentalization**—define distinct tasks
- **interdependency**—indicate task interrelationship
- **effort validation**—be sure resources are available
- **defined responsibilities**—people must be assigned
- **defined outcomes**—each task must have an output
- **defined milestones**—review for quality

4.5 Error Tracking

• Throughout the software process, a project team creates work products (e.g., requirements specification or prototype, design documents, source code).
• The important point to get across regarding error tracking is that metrics need to be defined so that the project manager has a way to measure progress and proactively deal with problems before they become serious. This also implies the need have historical project data to determine whether the current metric values are typical or atypical at a given point in the development process.
• Allows comparison of current work to past projects and provides a quantitative indication of the quality of the work being conducted. The more quantitative the approach to project tracking and control, the more likely problems can be anticipated and dealt with in a proactive manner.
• The software team perform formal technical review (and, later, testing) to find and correct error, E, in work product produced during software engineering tasks. Any errors that are not uncovered (but found in later tasks) are considered to be defects, *D*. Defect removal efficiency has been defined as **DRE = E / (E + D)**
• Defect Removal Efficiency is a process metric that provides a strong indication of the effectiveness of quality assurance activities, but DRE and the error and defect counts associated with it can also be used to assist a project manager in

determining the progress that is being made as a software project moves through its scheduled work tasks.

• • Let us assume that a software organization has collected error and defect data over the past 24 months and has developed averages for the following metrics:

1. Errors per requirements specification page, Ereq
2. Errors per component-design level, Edesign
3. Errors per component-code level, Ecode
4. DRE-requirements analysis
5. DRE-architectural design
6. DRE-component level design
7. DRE-coding

• As the project progresses through each software engineering step, the software team records and reports the number of errors found during requirements, design, and code reviews. The project manager calculates current values for Ereq, Edesign, and Ecode.

• These are then compared to averages for past projects. If current results vary by more than 20% from the average, there may be cause for concern and there is certainly cause for investigation.

Chapter Five

Software Quality Assurance

5.1 Introduction

❖ ***Software Quality Assurance SQA*** is an umbrella activity that is applied throughout the software process.

❖ SQA encompasses (1) a quality management approach, (2) effective software engineering technology (methods and tools), (3) formal technical reviews that are applied throughout the software process, (4) a multitier testing strategy, (5) control of software documentation and the changes made to it, (6) a procedure to ensure compliance with software development standards, and (7) measurement and reporting mechanisms.

5.2 Quality

Quality**:** refers to measurable characteristics or attributes of software. These properties include cyclomatic complexity, number of function points, and lines of code.

5.2.1 Two kinds (types) of quality:

1. Quality of design: the characteristics that designers specify for an item. It includes: the grade of materials (requirements), tolerance, performance specifications, and design of the system.

2. *Quality of conformance:* the degree to which the design specification are followed during manufacturing. It focuses on implementation based on the design.

5.3 Quality Concepts

❖ ***Variation control***: is the heart of quality control (software engineers strive to control the process applied, resources expended, and end product quality attributes).

5.3.1 How might a software development organization need to control variation?

From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time.
In general, minimize the number of defects that are released to the field, we'd like to ensure that the variance in the number of bugs is also minimized from one release to another. Minimize the differences in speed and accuracy of our hotline support responses to customer problems.
• • ***Quality control:*** is series of inspections, reviews, and test used throughout the develop cycle of a software product. It includes a feedback loop to the process created work product. That is, *(objective)* minimizes the produced defects, increase the product quality.
• • **Quality assurance** - consists of the auditing and reporting procedures used to provide management with data needed to make proactive decisions.

5.4 Statistical Software Quality Assurance

Statistical quality assurance implies the following steps:
1. Information about software defects is collected and categorized
2. Each defect is traced back to its cause (i.e. non conformance to specifications, design error, violation of standards, and poor communication with the customer).
3. Using the Pareto principle (80% of the defects can be traced to 20% of the causes) isolate the 20 percent "vital few" defect causes.
4. Move to correct the problems that caused the defects.
This represents an important step towards the creation of an adaptive software engineering process in which changes are made to improve those elements of the process that introduce error.

5.5 Software Reliability

• Defined as the probability of failure-free operation of a computer program in a specified environment for a specified time period.
• A factor can be measured directly and estimated using historical and developmental data (unlike many other software quality factors).
• The term ***failure*** is non-conformance to software requirements. Failure can be only annoying or catastrophic. One failure can be corrected within seconds while another requires weeks or even months to correct.
• The correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.
• Software reliability problems can usually be traced back to errors in design or implementation.
• A simple measure of reliability is *mean-time-between-failure* (MTBF), where:
MTBF = MTTF + MTTR
• The acronyms MTTF and MTTR are *mean-time-to-failure* and *mean-time-to-repair*, respectively.

5.6 Why is MTBF a more useful metric than defects/KLOC or defects/FP?

• Because end-users are concerned with failures, not with the total error count. Because each error contained within a program does not have the same failure rate, and that provides little indication of the reliability of a system.

5.7 Software availability

• *Software availability* is the probability that a program is operating according to requirements at a given point in time and is defined as:
Availability = [MTTF/(MTTF + MTTR)] x 100%