Software Requirements Engineering (SI-341)

LECTURE	TOPIC
Day 1	*Define Software Requirements and its types*
Day 2	*Sub types of Software Requirements*
Day 3	*Requirements Elicitation*
Day 4	*Method of Elicitation*
Day 5	*Types of Elicitation*
Day 6	*Types of Elicitation*
Day 7	*Levels of Requirements*
Day 8	*Requirement Process / Development and Management*
Day 9	
Day 10	*Data Flow Diagram*
Day 11	*ERD*
Day 12	*Requirement Traceability*
Day 13	*Requirement Prioritization*

Course Outline

Course Title	**Software Requirements Engineering**
Course Code	**SI-341**
Credit Hours	3
Category	IT Supporting
Prerequisite	CC-212: Software Engineering
Co-Requisite	None
Follow-up	None
Course Description	**Introduction:** Requirements Engineering. **Software Requirements:** Classification of requirements, Requirements process, Levels/layers of requirements, Requirement characteristics, and Analyzing quality requirements. **Software requirements in the context of systems engineering**: Requirement evolution, Requirement traceability, Requirement prioritization, Trade-off analysis, Risk analysis and impact analysis, Requirement management, Interaction between requirement and architecture. **Requirement elicitation**: Elicitation sources and techniques. **Requirement specification and documentation**: Specification sources and techniques, Requirements validation and techniques. **Management of Requirements**: Introduction to Management, Requirements management problems, Managing requirements in various organizations including acquisition, supplier and product-oriented organizations, Requirements engineering for agile methods.
Text Book(s)	1. Wiegers K. &Beatty J., Software Requirements, 3rd edition. Microsoft Press, 2013. 2. Elizabeth Hull, Ken Jackson and Jeremy Dick., Requirements Engineering, 3rd edition, Springer-Verlag London Limited, 2011.
Reference Material	1. Chemuturi M., Requirements Engineering and Management for Software Development Projects, Springer New York, 2013.

2 January 2024 **Tuesday**

Software Requirements

"A condition are capability needed by a user to solve archive and objective."

A condition is capability that must be met or processed by a system or system components of satisfy a contract standard specification or other formally imposed other documents.

There are three types of software requirements in following:

1. Functional Requirements
2. Non-Functional Requirements
3. Domain Requirements

Functional Requirement

These are the requirements that the end users specifically demand as basic facilities that the system should offer it can be a calculation, data manipulations, business process, user interactions or any other specific functionality which defined what function a system is likely to perform. Functional requirements is also known as functional specification.

For Example: A Hospital managements system a doctor should be able to retrieve the information of this patients.

Non- Functional Requirements

These are basically the quality constrain the system must satisfy according to project constrains. Non-Functional Requirements are not related to this system functionality but define how the system perform there are also called non behavioral requirements.

They basically deal with issues like.

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility
- Availability

NFR (Non-Functional Requirement) are classified to following types.

1. Interface Constraint
2. Performance Constraint (Response, Time Security, Storage/Space)
3. Operating Constraint
4. Life Cycle Constraint (Portability, Maintainability etc.)
5. Economy Constraint

Domain Requirements

Domain requirements are the requirements which are characteristics of a particular domain or category of projects. Domain's requirements can be functional or non-functional. The basic functions that a system of a specific domain must necessarily exhibits come under this category (domain).

For Example

In an academic software that maintain records of a school or college the functionality of being able to access. The list of facility and list of the student of each grade is a domain requirement. The requirements are there for identified from that domain model and not from user specific.

3 January 2024 **Wednesday**

Sub Types of Software Requirements

There are six (6) sub types of software requirements.

1. **User Requirements**
2. **System Requirements**
3. **Business Requirements**
4. **Regulatory Requirements**
5. **Interface Requirements**
6. **Design Requirements**

- **User Requirements**

  These requirements describe what the end user wants from software system. User Requirements are usually described in natural language and typically gather through interview, survey and user feedback.

- **System Requirements**

  These requirements specify the technical characteristics of the software system such its architecture, Hardware Requirements, Software components and interface.

- **Business Requirements**

  These requirements describe the Business goal and objectives that the software system expected to achieve. Business requirements are usually expressed in term of revenue, market share, customer satisfaction and business metrics.

- **Regulatory Requirements**

  These requirements specify the legal or regulatory standard that the software system must meet. Regulatory requirements many include data privacy, security, accessibility or other legal compliance requirements.

- **Interface Requirements**

  These requirements specify the interactions between the software and external system or components. Such as database and web services or other software application.

- **Design Requirements**

  These requirements describe the technical design of the software system. They include the software architecture data structure, algorithm and other technical aspects of the software.

9 January 2024 **Tuesday**

Requirements Elicitation

Definition:

Requirements elicitation is the process of the gathering and defining the requirements for a software system. The goals of requirements elicitation is to ensure that the software development process is based on the clear and comprehensive understanding of customers' needs and requirements.

What is Requirements Elicitation

Requirements elicitation is perhaps the most difficult most error-prone and most communication intensive software development phase:

1. It can be successed only through an effective customer-developer relationship. It is needed to know what the user is require
2. Requirements elicitation involves identifications, collections, analysis and refinements of the requirements for a software.
3. It is the critical part of the software development life cycle and are typically performed at the beginning of the project.
4. Requirements elicitation involves stack holder from different areas of the organization, including business owner. Endues and Technical experts.
5. The output of requirements elicitation Process is a set of clear, concise and well-defined requirements that serve the basic of the design and developments of software system.

Requirements Elicitation Methods

These are numbers of Elicitation method few of them listed below:

1. **Interview**

   Objective of conducting an interview understand for customer expectation from the software. It is Impossible to interview every stack holder hence representatives from groups are selected based on their expertise and credibility, interview may be open ended or structure.

   - In open ended interview these is no preset agenda. Context free questions may be asked to understand the problem.
   - In structured interview and agenda of fairly open questions is prepared. Sometime proper questions are designed for the interview.

10 January 2024 **Wednesday**

2. Brain Storming Session

→ It is group technique
→ It is intended to generate new idea, hence provide a platform to share views.
→ A highly trained facilitators is required to handle group bias and group conflict.
→ Every idea is documented and hence everyone can see it.
→ The output of this session is list of ideas and their priority if possible.

3. Facilitated Application Specification Technique (F.A.S.T) (important)

Its objective is to bridge the expectation gap-the difference between what the developer think. They are supposing to build and what customer think they are going to get. A team-oriented approach is developed for requirement gathering. Each attendee is asking to make a list of objectives that are.

- Part of the environment that surrounded the system.
- Produced by the system
- Used by the system

Each participants purpose his/her list, different lists are the combined, redundant entities are them eliminated and finally a draft of specification is written down using the input from meeting.

4. Quality Function Deployment

In this method customer satisfaction is prime concern hence it emphasizes on the requirements which valuable to the customer.

There are three types of requirements are identified

I. Normal Requirements

In this objective and goals of the proposed software and discussed with the customer.
Example: Normal requirements for a results management system may be entry of masks, calculation, results.

II. Expected Requirements

These requirements are so obvious that the customer need but not explicitly state them.
Example: Protection from unauthorized access.

III. Exciting Requirements

It is the features that are beyond the customers expectations and prove to be very satisfying when present.
Example: When authorized access is detected. It should backup and shutdown all process.

15 January 2024 **Monday**

5. Workshops

Workshops encourage stack holders' collaboration in defining requirements. A workshop is a structional meeting in which carefully selected group of stack holders and content expert work together to define, create, refine and reach closure on deliverable that represents user requirements.

Following are a few Tips for conducting effective elicitation workshop many of which may also applied on interview.

i. **Established and Enforce ground Rules**

The workshop participates should agree on some basic operating principles. Example Include starting and ending time. Returning form break promptly expecting everyone to contribute and focusing comments and criticisms or issue rather than induvial.

ii. **Fill All of the Team Roles**

A facilitator must make sure that the following tasks are covered by people or the workshop:

NOTE: Taking time, keeping, Ground Rule Management and make sure everyone heard.

iii. **Plan an agents**

iv. **Time Box discussion.**

v. **Keep the team small but include the right stack holders.**

6. Observations

Users are so familiar with executing a task that they cannot articulate what they do. Perhaps the tasks are so habitual that they do not even think about it. Sometimes you can learn a lot by observing directly how users perform their task. observation is time consuming so they are not suitable for every user or every task. If you use observation in the Agile project you have to demonstrate only the specific task related to the forthcoming iteration.

There are two types of Observations

i. **Silent observation** (non-participatory) is appropriate when busy user cannot be interrupted

ii. **Iterative observation** (participatory) allowed to interrupt the user mid task and ask questions some time to participate (if possible) in a field work.

16 January 2024 **Tuesday**

7. Questioners

Questioners is a way to survey large group of users to understand their needs. They are inexpensive making them a logical choice for eliciting information for large user population and they can be administered easily across geographical boundaries. The analyzed result of questioner can be used as a input to other elicitation techniques.

8. System Interface Analysis

Interface Analysis is an independent elicitation technique that entail examining the system to which user system connects system interface analysis reveals functional requirements regarding the exchange of data and services between system.

9. User Interface Analysis

User interface (UI) analysis and independent elicitation technique in which you study existing systems to discover users and functional requirements. If there is no existing system, you might be able to look at the user interface similar to your desired system.

10. Documents Analysis

Document Analysis entails examining Any existing documentation. for potential software requirement Most useful documentation include Requirement specification, business process, lesson, learned collection and manual for existing or similar Application.

17 January 2024 **Wednesday**

Levels of Requirements (important long question)

Software requirements include three different level

1. **Business Requirements**
2. **User Requirements**
3. **Functional Requirements**

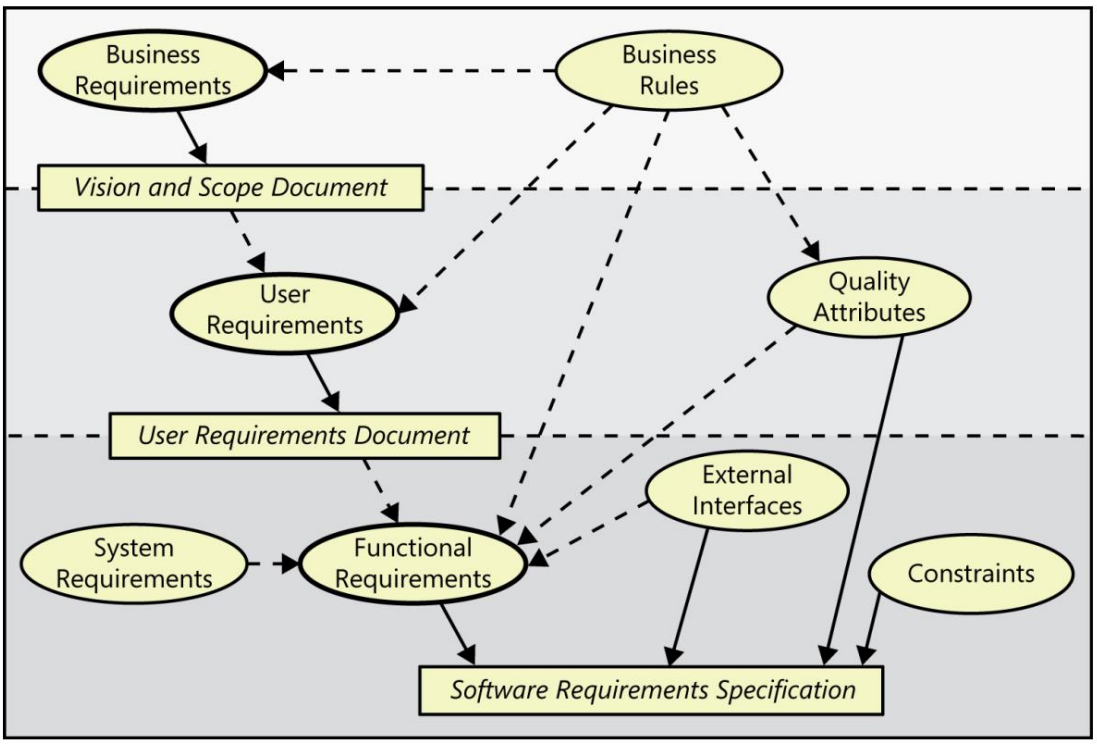
In addition, every system has an assortment of non-functional

→ Rectangle indicate documents in which to store that information

→ Solid arrow indicates that certain type of information

→ The dotted arrow indicate that one type of information is the origin of other type of requirements

→ Ovel represents requirements information



Level 1. Business Requirements

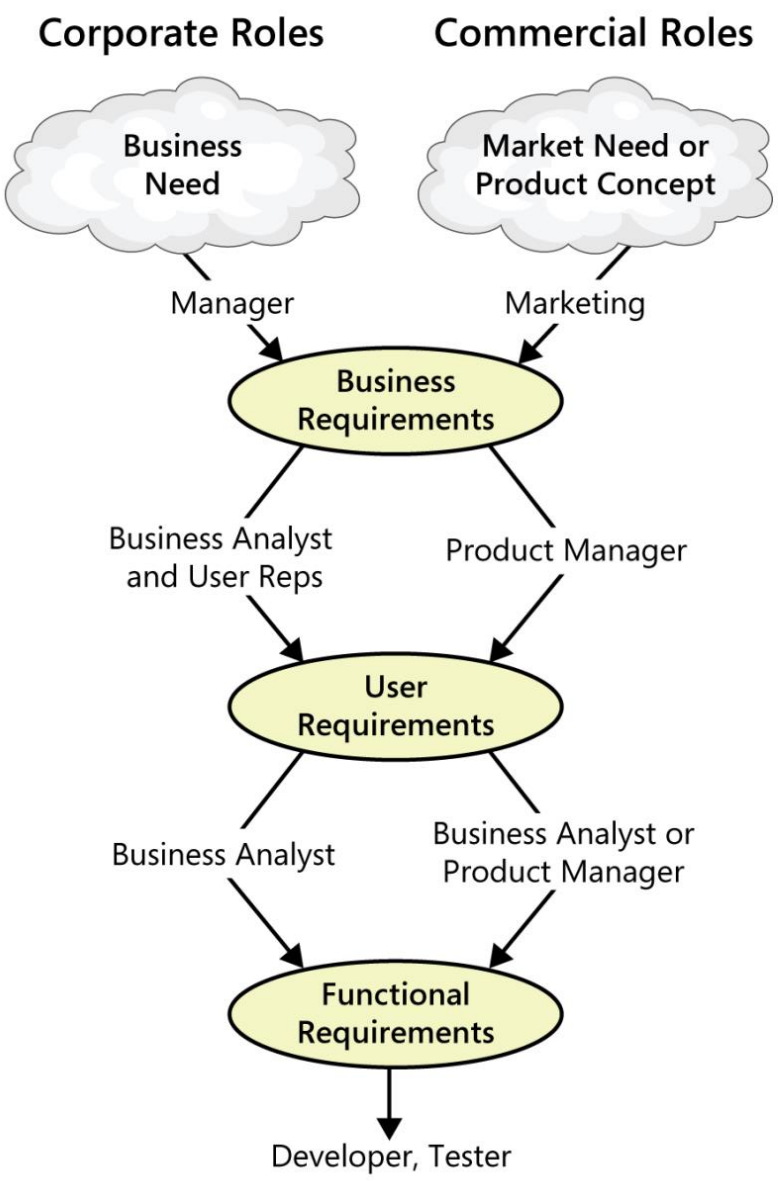
Business requirements describe why the organization implementing the system. The focus is on the business objective the organization or the customer who request the system.

Level 2. User Requirements

User requirements describe goals or tasks the user must be able to perform with the product that will provide value to someone. The domain of user requirements also includes description of product, attributes or characteristics that are important to user satisfaction ways to represents user requirements include usecase, user stories and response table.

Level 3. Functional Requirements

Functional requirements specify the behaviors the product will exhibits under specific conditions they describe what the developer must implements to enable user to accomplish their tasks (user requirements), thereby satisfying the business requirements.


Corporate Roles
Commercial Roles
Business Need
Market Need or Product Concept
Manager
Marketing
Business Requirements
Business Analyst and User Reps
Product Manager
User Requirements
Business Analyst
Business Analyst or Product Manager
Functional Requirements
Developer, Tester

22 January 2024 **Monday**

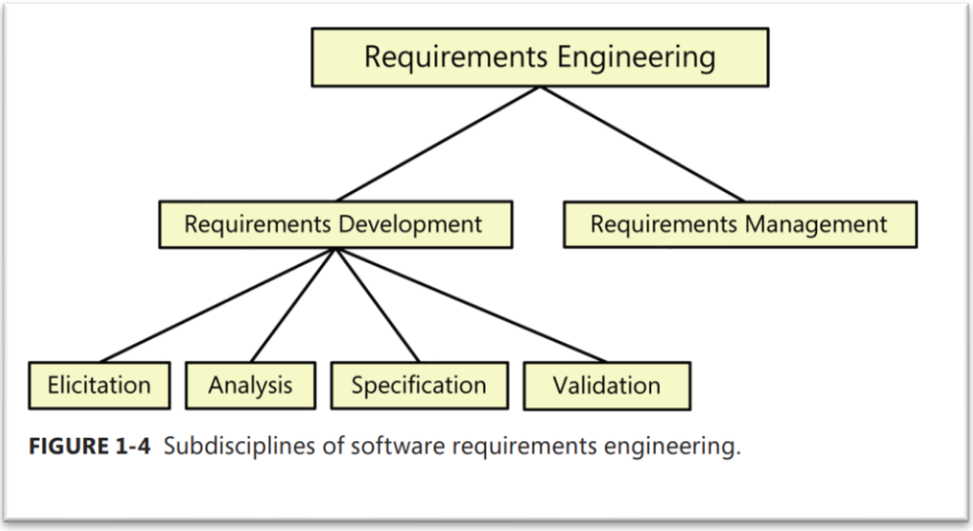


FIGURE 1-4 Subdisciplines of software requirements engineering.

Requirements Development and Management

We subdivide requirement development into Elicitation, Analysis, specification and validation. These subdisciplines encompass all the activities involved with exploring, evaluating documented and confirming the requirements for a product.

Following are the essential actions in each sub discipline.

Elicitation

Elicitation encompasses all of the activities involved with discovering requirements. Such as interview, workshops documents analysis prototyping and others. The key action are

- Identifying the products expected user class and other stakeholders.
- Understanding user tasks and goals and the business objective with which those tasks are align.
- Learning about the environment in which new product will be used
- Working with individuals who represent each user class to understand their functionality needs and their quality expectation.

Analysis

Analysis requirements involve reaching a richer and more precise understanding of each requirement and representing sets requirements in multiple ways following are principal activities.

- → Analysis the information received from users to distinguish their task goals form the functional requirements, quality expectation, business rules suggested solution and other information.
- → Documents High level requirements into an appropriate level of details.
- → Driving functional requirements from other requirements information.
- → Understanding the relative importance of quality attributes.
- → Negotiating implementation priorities.
- → Identifying gaps in requirements or unnecessary requirements as they relate to the define scope.

29 January 2024 **Monday**

Specification :-

Requirement specification involve representing and storing the collected requirement knowledge in a persistent and well-organized fashion. The principal activity is :

→ Translating the collected user needs into written requirements and diagram suitable for comprehension, review and used by their intended audiences.

Validation:-

Requirement validation confirms that you have the correct set of requirements information that will enable developers to build a solution that satisfies the business objectives. The control activities are:

→ Reviewing the documented requirements to correct set of requirements information before the development group accept them.

→ Developing **acceptance test** and criteria to confirm that a product based on the requirements would meet customer needs to achieve the business objectives.

Requirement Management:-

Requirement management activities involves:

→ Defining the requirement baseline, a snapshot in time that represent agreed-upon reviewed and approve set of functional and non-functional requirements offen for a specific product release or development iteration.

→ Evaluating the impact of proposed requirements changes and incorporating approved changes into a project in a controlled way.

→ Keeping project plan current with the requirements as they evolve.

→ Negotiating new commitments based on the estimated impact of requirement changes.

→ Defining the relationship and dependencies that exist between requirements.

→ Tracking requirements status and change activity throughout the project.

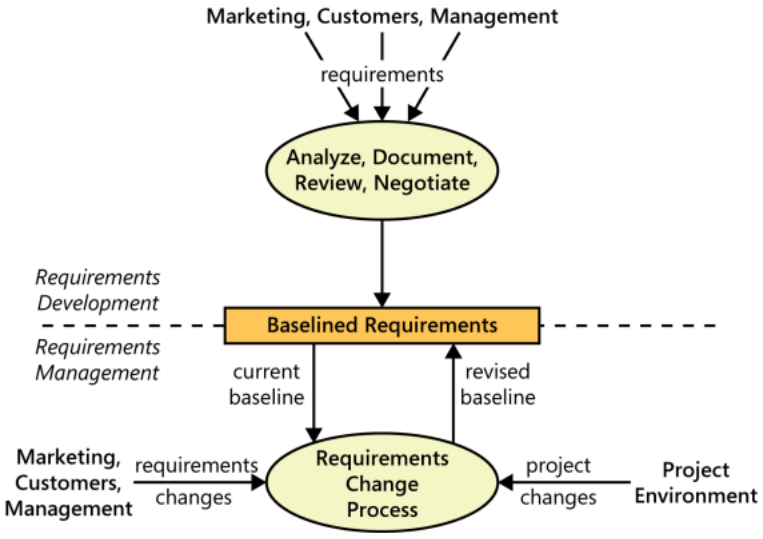


FIGURE 1-5 The boundary between requirements development and requirements management.

30 January 2024 **Tuesday**

Every project has requirements

Frederick Brooks eloquently stated the critical role of requirements to a software project in his classic 1987 essay, “No Silver Bullet: Essence and Accidents of Software Engineering”: The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

When bad requirements happen to good people

- Insufficient User involvement (pdf page no 53)
- Inaccurate Planning
- Creeping user Requirements
- Ambiguous Requirements
- Gold Plating
- Overlooked Stakeholders

Benefits from a high-quality requirements process

Some people mistakenly believe that time spent discussing requirements simply delays delivery by the same duration. This assumes that there’s no return on investment from requirements activities. In actuality, investing in good requirements will virtually always return more than it costs. Sound requirements processes emphasize a collaborative approach to product development that involves stakeholders in a partnership throughout the project. Eliciting requirements lets the development team better understand its user community or market, a critical success factor. Emphasizing user tasks instead of superficially attractive features helps the team avoid writing code that no one will ever execute. Customer involvement reduces the expectation gap between what the customer really needs and what the developer delivers. You’re going to get the customer input eventually; it’s far cheaper to reach this understanding before you build the product than after delivery.

Data Flow Diagram

3.2 Representations for Requirements Engineering

3.2.1 Data Flow Diagrams

Data flow diagrams (DFDs) are the basis of most traditional modelling methods. They are the minimalist graphical representation of the system structure and interfaces and although initially produced for use in data representation and flow, the diagrams can in fact be used to show any type of flow, whether a computer-based system or not. The one output which DFDs do not show is that of control flow.

The elements in a data flow diagram consist of

- Data flows (labelled arrows)
- Data transformations (circles or "bubbles")
- Data stores (horizontal parallel lines)
- External entities (rectangles)

The simple example in Fig. 3.1 shows the use of a data flow diagram in its traditional, information systems context.

Flows represent the information or material exchanged between two transformations. In real-world systems, this may be continuous, on demand, asynchronous etc. When using the notation, diagrams must be supported by textual descriptions of each process, data store and flow.

A *data dictionary* is used to define all the flows and data stores. Each leaf node bubble defines the basic functionality provided by the system components. These are described in terms of a *P-spec* or mini-spec. This is a textual description often written in a pseudo-code form.

The context diagram is the top-level diagram of a DFD and shows the external systems interacting with the proposed system, as in Fig. 3.2.

Bubbles can be decomposed another layer down. Each bubble is exploded into a diagram which itself may contain bubbles and data stores. This is represented in Fig. 3.3.

To illustrate the use of a DFD, consider an example of a context diagram for an Ambulance Command and Control system (Fig. 3.4). This is the starting point for a data-flow analysis of the system.

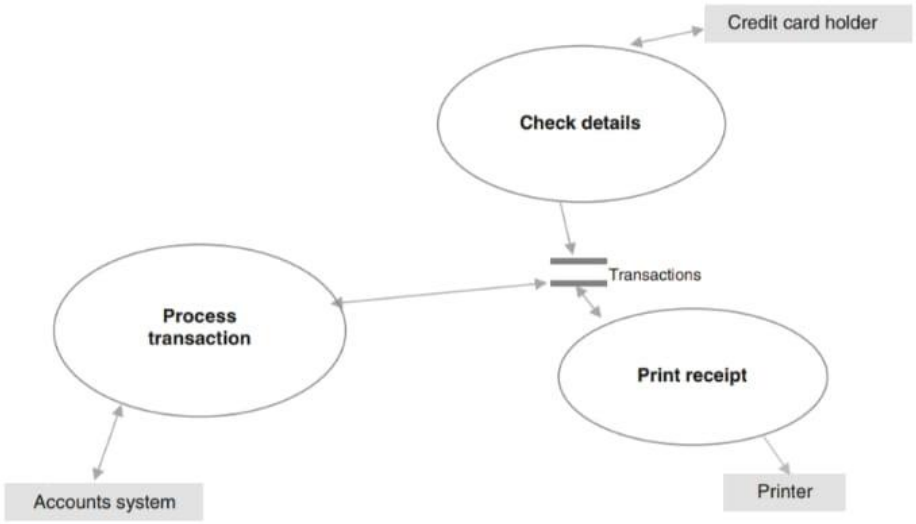


Fig. 3.1 Data flow diagram

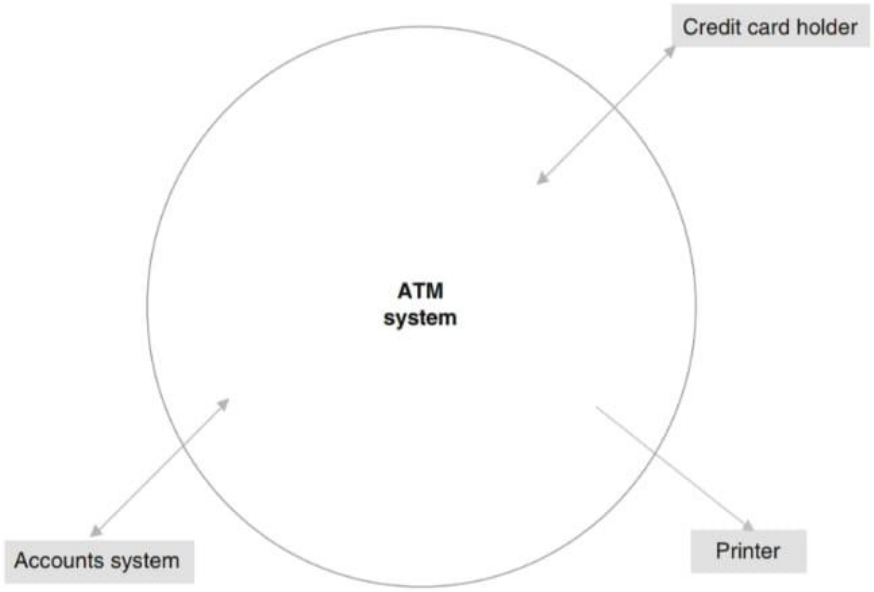


Fig. 3.2 Context diagram

The primary external entities are the *callers*, who make the emergency calls, and the *ambulances*, which will be controlled by the system. Note that *records* are an important output of the system (in fact a legal requirement) and a very important means of measuring "performance".

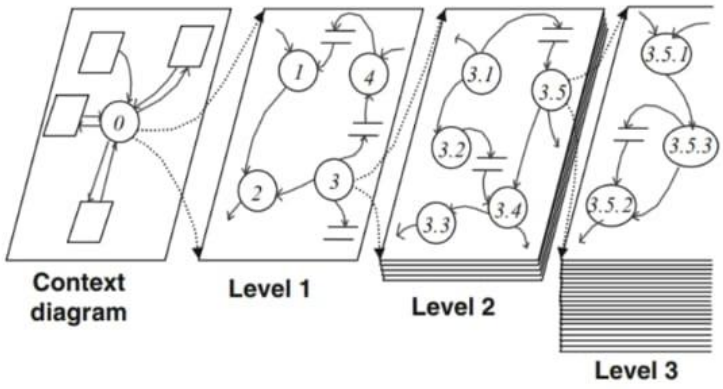


Fig. 3.3 Functional decomposition

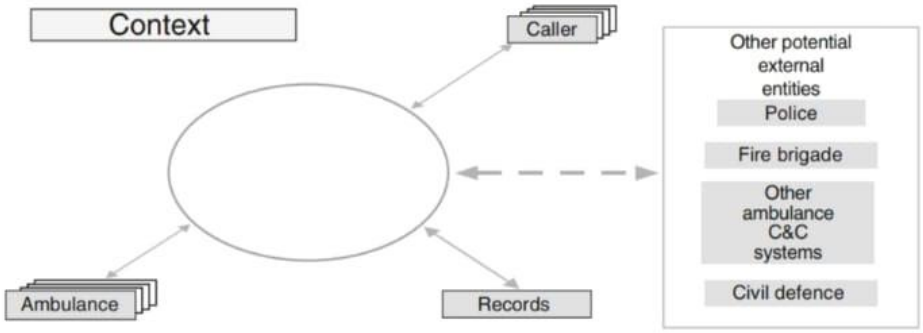


Fig. 3.4 Context diagram for Ambulance C&C System

Other potential external entities that would be required for a real system are shown in the diagram, but for simplicity we shall ignore them.

The next step is to identify the internal functionality of the system. Usually starting by drawing a function for each external entity as the minimal decomposition and then drawing the basic data that must flow between these top-level functions – see Fig. 3.5.

Following this, decomposition of the top-level functions takes place thus including more detail, as shown in Fig. 3.6.

The functional hierarchy in a set of data flow diagrams can be used as a framework for deriving and structuring system requirements. Figure 3.7 shows the functional structure for the Ambulance Command & Control example derived from Fig. 3.6.

Figure 3.7 also indicates some examples of requirements derived from this structure.

The hierarchical breakdown and interfaces give a good view of the component model, but they give a poor view of the "transactions" across the system i.e. from input to output (or to complete some system action) as can be seen in Fig. 3.8.

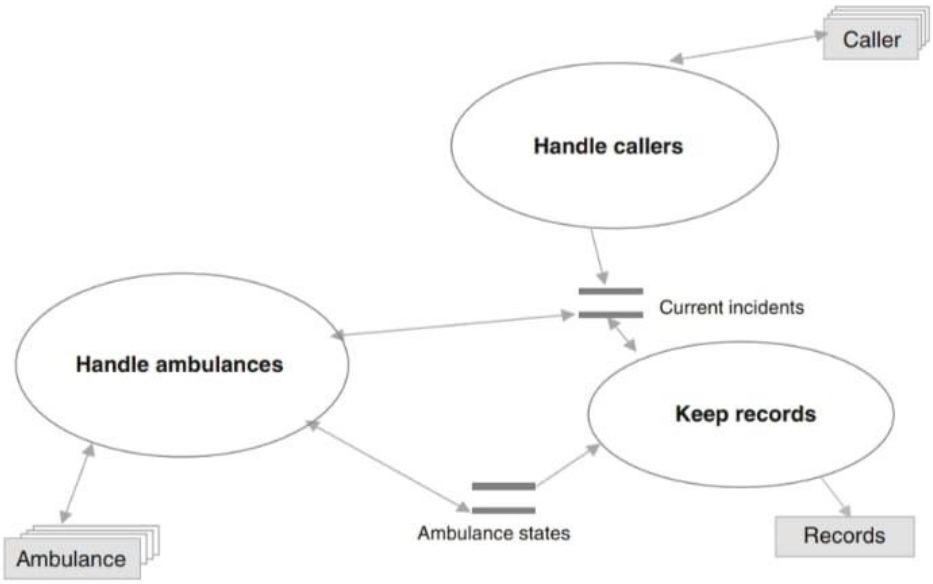


Fig. 3.5 Model for Ambulance C&C system

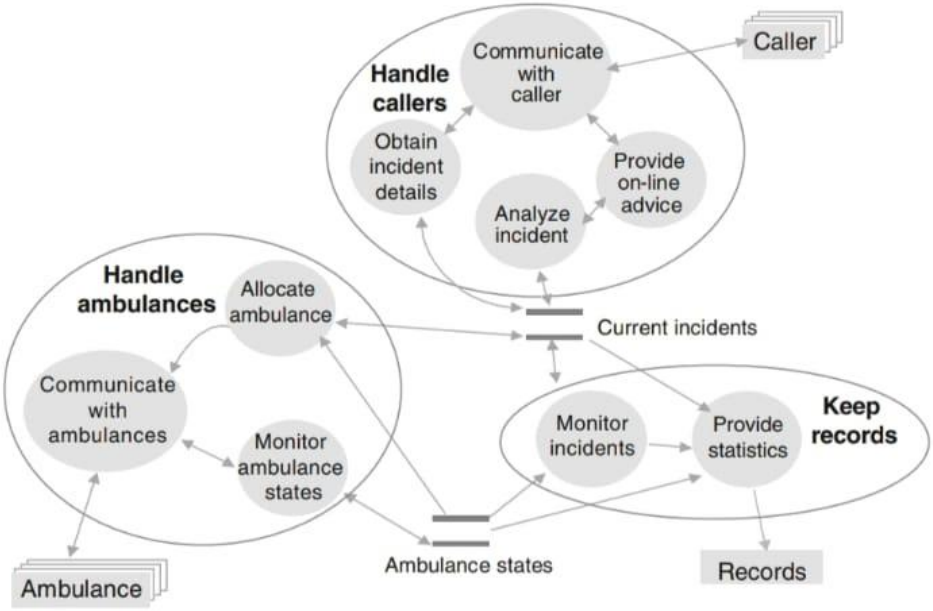


Fig. 3.6 Detailed model for Ambulance C&C system

It is therefore necessary to observe these transactions across the system in terms of the path(s) they follow, the time they take and the resources they absorb. Animating the stakeholder requirements and being able to see which functions are

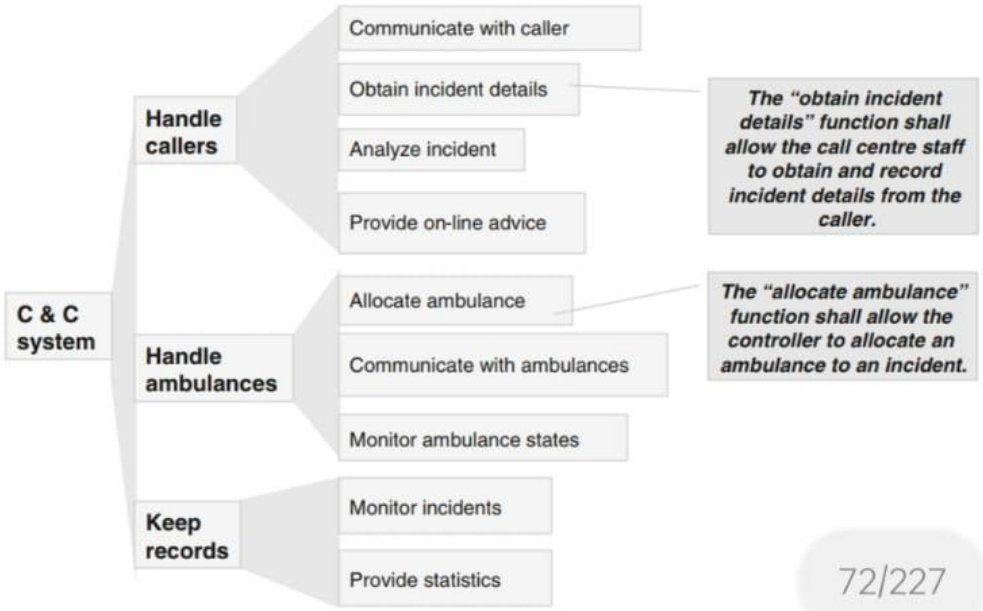


Fig. 3.7 Functional structure for Ambulance Command & Control system

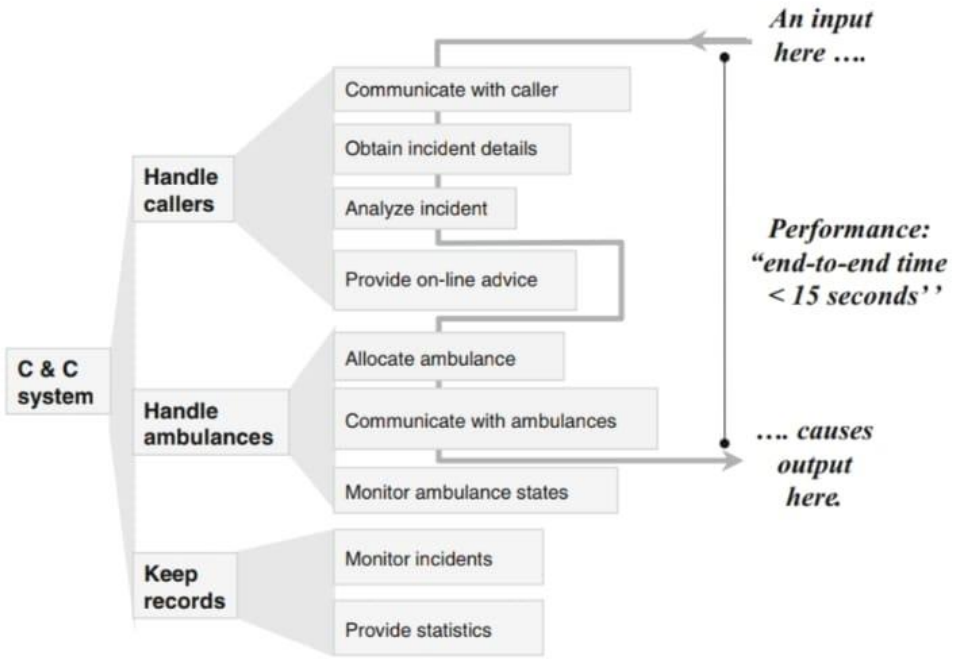


Fig. 3.8 System transactions

operating, will illustrate major transactions, but an alternative way of showing the system transactions is to mark them on to a data flow diagram as shown in Fig. 3.9, using the thick arrows.

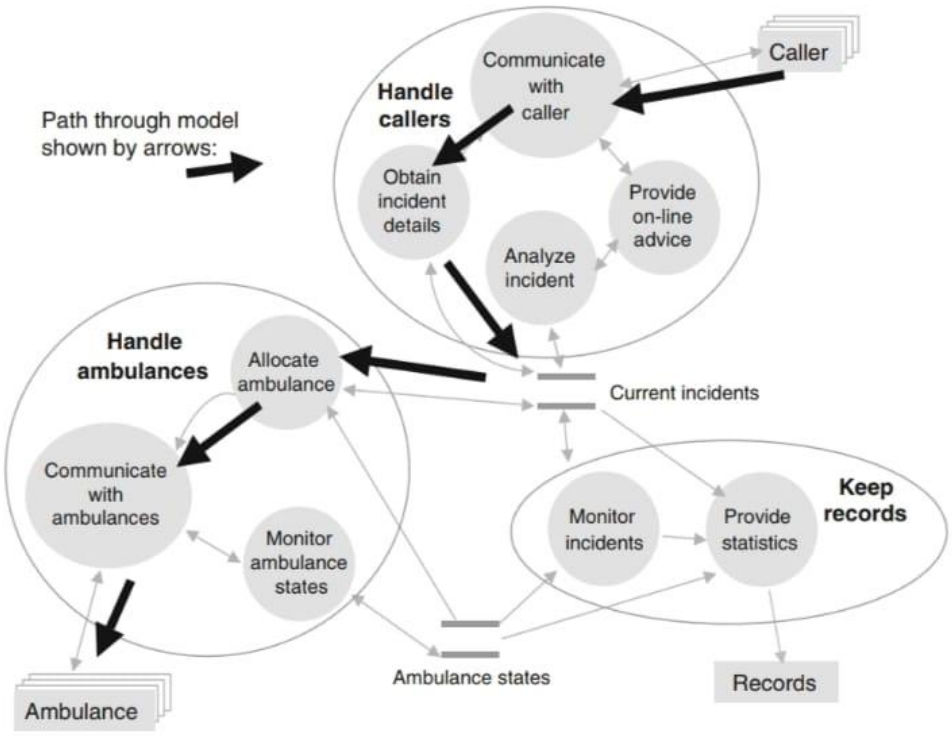


Fig. 3.9 System transactions for Ambulance Command & Control system

DFDs are good at presenting structures but they are not very precise. DFDs are less precise than text for developing a complete definition of a system – interface lines can mean anything, and single words can summarize anything. They cannot handle constraints properly.

A DFD clearly shows functions and interfaces. It can be used to identify end-to-end transactions, but does not directly show them. Ideally we would like to view the diagrams with an "expand in place" approach so that it is possible to view the context in which each level of decomposition is intended to work. Few CASE tools provide this level of facility.

Figure 3.6 actually breaks the conventions for drawing DFDs, because it shows a decomposition of the overall system into several processes and it also shows external agencies with which the system must interact. We advocate a pragmatic use of DFDs, rather than strict adherence to a conceptually pure ideal. To follow precisely the rules for drawing DFDs, the external agencies should appear only in the context diagram, and hence should not be visible at this level. However, the diagram would be far less meaningful if the external agencies were not shown and the flows to them left dangling (which is the defined convention for them).

In summary, DFDs:

- Show overall functional structure and flows.
- Identify functions, flows and data stores.

- Identify interfaces between functions.
- Provide a framework for deriving system requirements.
- Tools are available.
- Widely used in software development.
- Applicable to systems in general.

ERD

3.2.2 *Entity-Relationship Diagrams*

Modelling the retained information in a system, for example flight plans, system knowledge and data base records, is often important. Entity relationship diagrams (ERDs) provide a means of modelling the entities of interest and the relationships that exist between them. Chen (1976) initially developed ERDs. There is now a very wide set of alternative ERD notations.

An *entity* is an object that can be distinctly identified such as: customer, supplier, part, or product. A *property* (or *attribute*) is information that describes the entity. A *relationship* has cardinality, which expresses the nature of the association (one-to-one, one-to-many, many-to-many) between entities. A *subtype* is a subset of another entity, i.e. a type X is a sub-type of Y if every member of X belongs to Y.

ERDs define a partial model of the system by identifying the entities within the system and the relationships between them. It is a model that is independent of the processing which is required to generate or use the information. It is therefore an ideal tool to use for the abstract modelling work required within the system requirements phase. Consider the example Ambulance C&C system in Fig. 3.10.

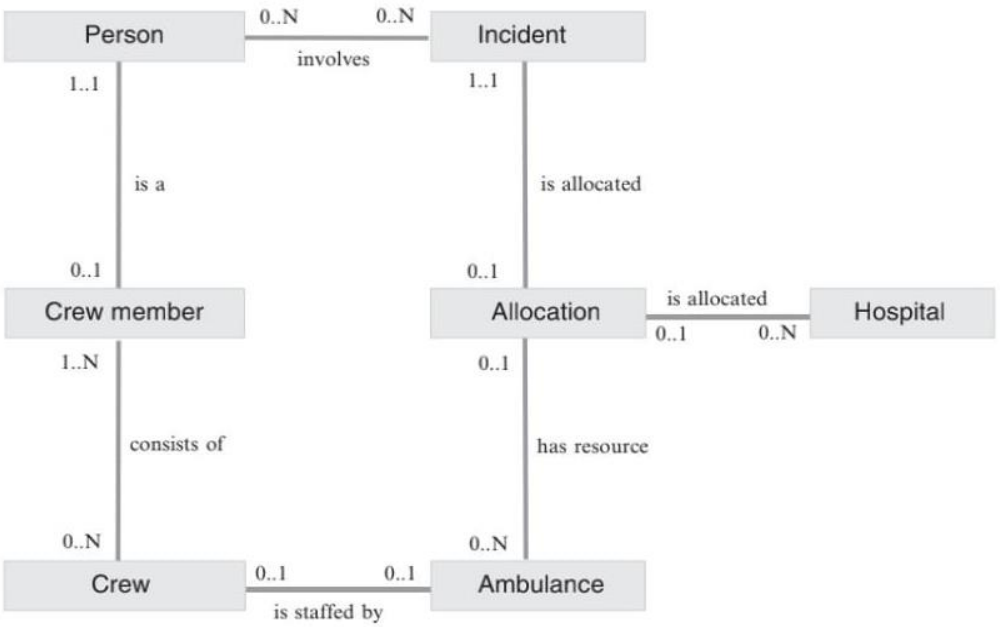


Fig. 3.10 ERD for Ambulance C&C system

3.2.3 *Statecharts*

Functionality and data flows are not enough for requirements definition. It is also necessary to be able to represent the behaviour of the system and in some circumstances consider the system as having a finite number of possible 'states', with external events acting as triggers that lead to transitions between the states.

To represent these aspects it is necessary to examine what states the system can be in and how it responds to events in these states. One of the most common ways of doing this is to use Harel's Statecharts (Harel 1987).

Statecharts are concerned with providing a behavioural description of a system. They capture hierarchy within a single diagram form and also enable concurrency to be depicted and therefore they can be effective in practical situations where parallelism is prevalent. A labelled box with rounded corners denotes a state. Hierarchy is represented by encapsulation, and directed arcs, labelled with a description of the event, are used to denote a transition between states.

The descriptions of state, event and transition make statecharts suitable for modelling complete systems.

Figure 3.11 presents a statechart for an aircraft flight. The two top-level states are "Airborne" and "On Ground", with defined transitions between them. Inside the "Airborne" state, there are three independent sets of states, while within the

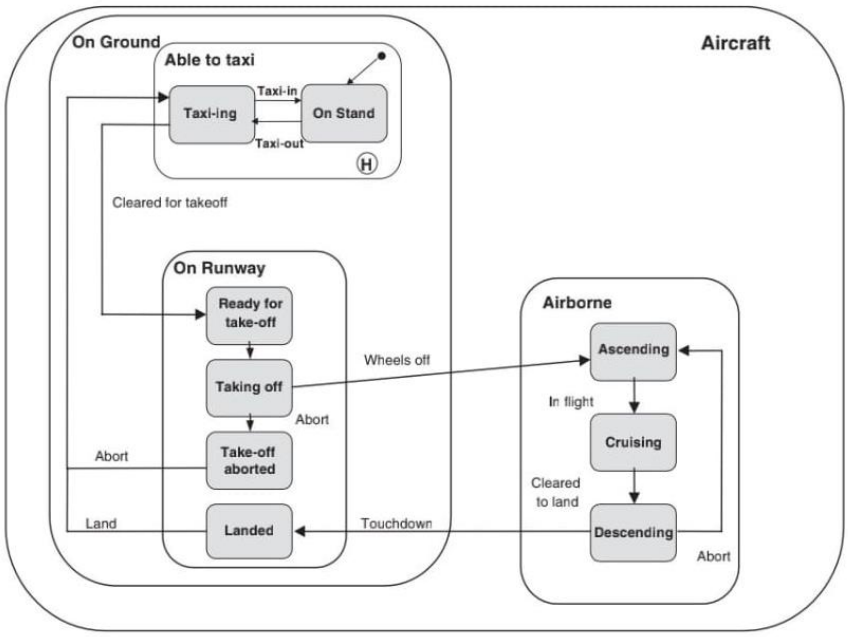


Fig. 3.11 Statechart for aircraft flight

19 February 2024 **Monday**

Requirements Traceability

Traceability is the ability to trace something as it move through a progress. In product development it refers to the ability to track an trace requirements to artifacts test, run and anything else in the product life cycle. Traceability is the important because it create transparently of every step of development including who completed what task and when.

Purpose of Traceability

The purpose of traceability is to keep track of record the history of are item which is of ten used to comply with regulation and minimize risk. More specifically some high-level examples of traceability include:

- Risk mitigation
- Quality Control
- Faster Ship or Release
- Operational Efficiency
- Proving Compliance with time and Budget

Traceability in Software Engineering

Traceability in software engineering is the ability to trace work item across the development lifecycle. It is used to keep track of what's going on in the development life cycle.

Requirements Traceability

Traceability works by linking two or more work items application development. This link indicate a depending between the items. Requirements and test cases are often traced. Requirements are traced as requirements forwards and backwards in development lifecycle. Requirements are traced forward through other development artifacts including testcases, test runs and issues. Requirements are traced back ward to source of the requirements such as a stakeholders or regulatory compliance menilite.

20 February 2024 **Tuesday**

Requirements Prioritization (LONG)

An act of giving procedure or priority to one item over another item. Requirements prioritization means giving procedure to some requirements over other requirements based on feedback from system stakeholders.

Benefits of Requirements Prioritization

- Stakeholders can decide on the core requirements for the system
- Planning and selection of ordered, optimal set of software requirements for implementation in successive releases
- Helps in trade-offs of conflicting constraints such as schedule, budget, resources, time to market, and quality.
- Balances the business benefit of each requirement against its cost
- Balances the implications of requirements on the software architecture and future evolution of the product and its associated cost.
- Selects only a subset of the requirements and still produce a system that will satisfy the customers.

Prioritization Process

• Prioritization is an iterative process and might be performed at different abstraction levels and with different information in different phases during the software lifecycle.

• Prioritization techniques can roughly be divided into two categories:

- Methods
- Negotiation approaches

Prioritization Methods

• The methods are based on quantitatively assigning values to different aspects of requirements

• Quantitative methods make it easier to aggregate different decision variables into an overall assessment and lead to faster decision.

Analytical Hierarchy Process (AHP)

- AHP is a systematic decision-making method that has been adapted for prioritization of software requirements
- It involves comparing all possible pairs of hierarchically classified requirements, in order to determine which has higher priority, and to what extent 30 Analytical Hierarchy Process (AHP).

- The total number of comparisons to perform with AHP are n * (n-1)/2; where n is the number of requirements; at each hierarchy level, which results in a dramatic increase in the number of requirements.
- Studies have shown that AHP is not suitable for large number of requirements.

Cumulative Voting, the 100-Dollar Test

- The 100-dollar test is a very straightforward prioritization technique where the stakeholders are given 100 imaginary units (money, hours, etc.) to distribute between the requirements
- The result of the prioritization is presented on a ratio scale 32 Cumulative Voting, the 100-Dollar Test
- One should only perform the prioritization once on the same set of requirements, since the stakeholders might bias their evaluation the second time around if they do not get one of their favorite requirements as a top priority