CS510: Software Requirements and Specification (HANDOUTS)

Lecture # 1 (week # 1) TO Lecture # 15 (week # 15)

Complete Handouts

Arranged by:

Mubarak Ali bc140400995

&

Fatima Ali bc140400739

Overview

Why this course needed?

- **Let's visualize how it is difficult to understand others point of view**

What the customer really needed.

How the customer explain it.

How the projector leader understood it.

How the analyst design it.

How the projector was documented.

Course learning objectives

- **To understand the role of Software Requirement and Specification(SRS) in software projects.**
- **To understand the essential nature of SRS.**
- **To study current techniques, notations, methods, processes and tools used in SRS.**
- **To gain practical experience in writing of the SRS document**

Course Evaluation

- **Assignments**
- **Discussion/Presentation**
- **Quizzes**

◦

◦ Mid term exam and

Final term exam

Course Textbooks

- Requirements Engineering: Processes and Techniques, Gerald Kotonya and Sommerville, John-Wiley Sons, 1998 (or Latest Edition).
- Software Requirements, Karl E. Wiegers, Microsoft Press, 2003(or Latest Edition).
- Software Requirements Specification, David Tuffley, CreateSpace Independent Publishing Platform, 2010 (or Latest Edition).
- System Requirements Engineering, Loucopoulos and Karakostas, McGraw-Hill, 1995 (or Latest Edition).

Topics to be covered

○ Software Requirement basic concepts

❖ What, Why and Who ○ SRS processes

❖ Sequence of activities that

❖ need to be performed in

❖ the requirement phase ○

Requirement Elicitation

❖ Process of discovering, reviewing, documenting, and understanding the user's needs and constraints for the system.

○ Requirement Modeling

❖ Visualization of requirements for better understanding and analysis ○ Requirement Analysis

❖ Refining the user's needs and constraints ○ Requirement Specification

❖ Process of documenting the user's needs and constraints clearly and precisely.

○ Requirement verification and validation

❖ Process of ensuring that the system requirements are complete, correct, consistent, and clear.

○ Requirement Management

❖ Scheduling, coordinating, and documenting the requirements engineering activities ○

Requirement Traceability

- **if the source of the requirements can be identified**

<u>Lecture # 1 week # 2</u>

Introduction

- **What is a Requirement?**
  - **Something required, something wanted or needed**
  - **A statement of a system service or constraint**
  - **A condition or capability that must be possessed by a system (IEEE)**
- **Why requirement is needed?**
  - **Requirements form the basis for all software products**

Requirement Challenges

- **Challenges**
  - **Necessarily involves people interaction**
  - **Cannot be automated**
- **Why it is hard to Understand Requirements?**
  - **Visualizing a future system is difficult**
  - **Capability of the future system not clear, hence needs not clear**
  - **Requirements change with time**

Requirement Task

- **Input**
  - **Users need in mind of people**
- **Output**
  - **precise statement of what the future system will do**

Requirement Examples

-

  The system shall allow users to search for an item by title, author, or by International Standard Book Number

- **The system's user interface shall be implemented using a web browser**

Lecture # 1 week # 3:

Requirement Engineering (RE)

- What is a Requirement Engineering?
  - Requirement Engineering is a new area which is started in 1993.
  - The first International symposium was held On RE in 1993.
  - It is the process, which is used to determine the requirements for a software product systematically.

Requirement Engineering (RE)

- What is a Requirement Engineering?
  - The development and use of technology effective to elicit, specify and analyse requirements from stakeholders (clients/users) that shall be performed by a software system

Importance of RE

- *"26% of the Software projects were considered a success."*
- *Meaning that 74% have FAILED!* Standish Group, CHAOS Report, 2000
- "*56% of the errors in a software can be traced back to the requirements phase"* Tom De Marco (a US Software Engineer)

Importance of RE

- The hardest 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 of 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

Fred Brooks

Importance of RE(cont.)

- Software complexities
- Frequent change in user requirements
- Outsourcing offshore projects
- Cost of fixing errors
- Causes of failure

LECTURE # 2 WEEK # 1

Characteristics of Good Requirements

- How to judge good and bad requirements?
  - There are several criteria need to meet for good requirements.
  - Usually overlooked in requirement process
  - An excellent source to measure projects quality and progress.

Characteristics of Good Requirements

- How to judge good and bad requirements?
  - Characteristics of requirements vs Characteristics of Requirement Specification.
  - Meaning become somehow different when considering a single requirement or a set of requirements i.e. SRS

Key characteristics of good requirements

- ***Feasible***
- Valid
- Unambiguous

-
- Verifiable

  Modifiable
- Consistent
- Complete and
- Traceable

Requirement Feasibility

- Also considered as Realistic or possible
  - Requirement is feasible if it is implementable within the given constraints or resources like budget, time and available technology etc.
- Example:
  - Requirements to handle 10000 transactions/second might be feasible in given current technologies but might not be feasible with agreed platform or technology.

Key characteristics of good requirements

- Feasible
- ***Valid***
- Unambiguous
- Verifiable
- Modifiable
- Consistent
- Complete and
- Traceable

Requirement Validity

- Normally termed as correct
- Requirement should be valid if and only if the requirement is one that system shall meet.

- Validity can be done by reviewing with key stakeholders who decide the success or failure of project
- “must” and “nice to have” requirements should clearly be demarcated

Requirement Validity ▸

Example

- Car rental prices shall show all applicable taxes (including 6% state tax).
- Here mentioning 6% tax is incorrect because it is dependent

References

- Software & Systems Requirements Engineering - In Practice Mar 2009 ◦

  IBM press

LECTURE # 2 WEEK # 2

Key characteristics of good requirements

- Feasible
- Valid
- ***Unambiguous***
- Verifiable
- Modifiable
- Consistent
- Complete and
- Traceable

Unambiguous Requirements

- If a requirement has only one interpretation then it is called unambiguous requirements
- Source of ambiguity is:
- Natural language

-

- Ambiguity level shows the quality of requirements
- Can effect project schedule and budget

Unambiguous Requirements

- Example:
- Ambiguous statement:
- “The data complex shall withstand a catastrophe (fire, flood).”
- Unambiguous statement:
- The data complex shall be capable of withstanding a severe fire. It shall also be capable of withstanding a flood

Characteristics of Good Requirements

- Feasible
- Valid
- Unambiguous
- ***<u>Verifiable</u>***
- Modifiable
- Consistent
- Complete and
- Traceable

Verifiable requirements

- Also termed as testable requirements
- Requirements are verifiable if the developed system or application can be tested to ensure that it meets the requirements.
- But product features are not easy to be verified
- Proper analysis is needed to make it testable

Verifiable requirements

- Example:
  - The car shall have power brakes.
    - Abstract so Not testable

-
-
- Detailed testable requirement:

  The car shall come to a full stop from 60 miles per hour within 5 seconds.

Key characteristics of good requirements

- Feasible
- Valid
- Unambiguous
- Verifiable
- ***Modifiable***
- Consistent
- Complete and
- Traceable

Requirements Modifiability

- Requirements are modifiable if any changes can be made to the requirements easily, consistently and completely without any changes to the existing structure and style of document.
- Redundancy is a key factor

References

- Software & Systems Requirements Engineering - In Practice Mar 2009
- IBM press

LECTURE # 2 WEEK # 3

Key characteristics of good requirements

- Feasible
- Valid
- Unambiguous

-

- Verifiable
- Modifiable

  Consistent

  Complete and
- Traceable

Consistent Requirements

- A relationship among two or more requirements
- A requirement is consistent if it does not contradicts or in conflicts with other requirements
- These requirements should either be external documents, standards or other requirements.

Consistent Requirements

- Example:
  - Dates shall be displayed in the mm/dd/yyyy format.
  - Dates shall be displayed in the dd/mm/yyyy format.
  - Both internal and external consistency is requrired

Characteristics of Good Requirements

- Feasible
- Valid
- Unambiguous
- Verifiable
- Modifiable
- Consistent
- ***Complete*** and
- Traceable

-
-

Complete Requirements

- A requirement should be present for all conditions that can occur.
- Very difficult to Check
- Can effect project schedule and budget

-
- There is no way to be sure that all requirements has been captured

  Its because user can add new requirements at the end of the requirement engineering phase.

Characteristics of Good Requirements

- Feasible
- Valid
- Unambiguous
- Verifiable
- Modifiable
- Consistent
- Complete and
- ***Traceable***

Traceable requirements

- Requirements are traceable if the source of the requirements can be identified
- It is the ability to describe and follow the life of requirements in forward and backward direction
- Why tractability:
- Needed for requirement management and project tracking
- If requirements are atomic and having unique id then it would be traceable.

References

- Software & Systems Requirements Engineering - In Practice Mar 2009
- IBM press

LECTURE # 3 WEEK # 1:

◦

◦

Kinds of Software Requirements

<u>Functional requirements</u>

Non-functional requirements

- ◦ Domain requirements
- ◦ Inverse requirements
- ◦ Design and implementation constraints

Functional Requirements

- ◦ Statements describing what the system does
- ◦ Functionality of the system
- ◦ Functional requirements should be complete and consistent

- Example
  - ◦ The user shall be able to search either the entire database of patients or select a subset from it (admitted patients, or patients with asthma, etc.)

Kinds of Software Requirements

- ◦ Functional requirements
- ◦ ***<u>Non-functional requirements</u>***
- ◦ Domain requirements
- ◦ Inverse requirements
- ◦ Design and implementation constraints

Non-functional Requirements (NFR)

- What are Non-functional Requirements?
  - ◦ Quality factors, design criteria and metrics.
  - ◦ Non-functional requirements defines how the system suppose to be.
  - ◦ Most non-functional requirements relate to the system as a whole.
  - ◦ They include constraints on timing, performance, reliability, security, maintainability, accuracy, the development process, standards, etc

- Often more critical than individual functional requirements

Kinds of Software Requirements

- Functional requirements
- Non-functional requirements
- ***Domain requirements***
- Inverse requirements
- Design and implementation constraints

Domain Requirements

- Requirements that come from the application domain and reflect fundamental characteristics of that application domain
- These can be both the functional or non-functional requirements
- Example
- Most banks do not allow over-draw on most accounts, however, most banks allow some accounts to be over-drawn
-

Kinds of Software Requirements

- Functional requirements
- Non-functional requirements
- Domain requirements
- ***Inverse requirements***
- Design and implementation constraints

Inverse Requirements

- They explain what the system shall not do.
- Many people find it convenient to describe their needs in this manner

▸ Example

-
-
- The system shall not use red color in the user interface, whenever it is asking for inputs from the end-user

Kinds of Software Requirements

- Functional requirements

- ◦ Non-functional requirements
- ◦ Domain requirements
- ◦ Inverse requirements
- ◦ ***<u>Design and implementation constraints</u>***

Design and implementation constraints

- ◦ They are development guidelines within which the designer must work
- ◦ These requirements can seriously limit design and implementation options

▸ Example

- ◦ The system shall be developed using the Microsoft Dot Net platform
- ◦ The system shall be developed using open source tools and shall run on Linux operating system

References

- ◦ Software Requirements: Objects, Functions, and States by A. Davis, PH, 1993
- ◦ Software Engineering 6th Edition, by I. Sommerville, 2000

<u>LECTURE # 3 WEEK # 2</u>

Requirement Engineering (RE) Process

- ▸ What is process?
  - ◦ A process is an organized set of activities, which transforms inputs to outputs
  - ◦ Processes document the steps in solving a certain problem
- ▸ Why process?
  - ◦ They allow knowledge to be reused
  - ◦ Processes are essential for dealing with complexity in real world

Process

- Example
    - An instruction manual for operating a microwave oven
    - An instruction manual for assembling a computer or its parts

Software Processes

- Software engineering, as a discipline, has many processes
- These processes help in performing different software engineering activities in an organized manner

Software Processes

- Examples
    - Software engineering development process (SDLC)
    - Requirements engineering process
    - Design process
    - Quality assurance process

References

- Requirements Engineering: Processes and Techniques' by G. Kotonya and I. Sommerville, John Wiley & Sons, 1998

LECTURE # 3 WEEK # 3

Requirement Engineering (RE) Process

- What is RE process?
    - The process(es) involved in developing the system requirements collectively called

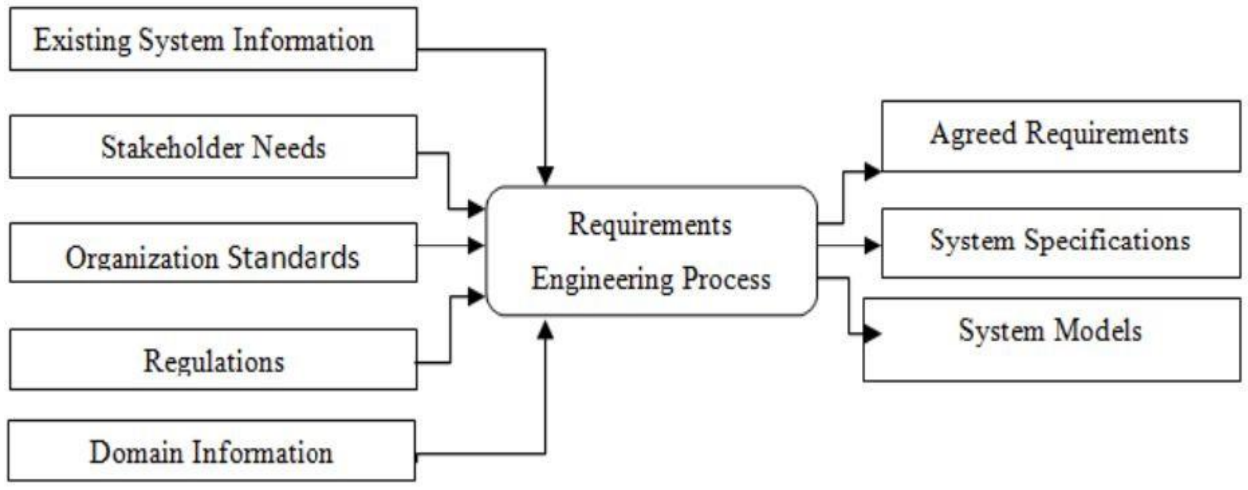
RE process(es) ▸ Which process to be used? ◦ depends on:

- application domain
- the people involved and
- the organisation developing the requirements.

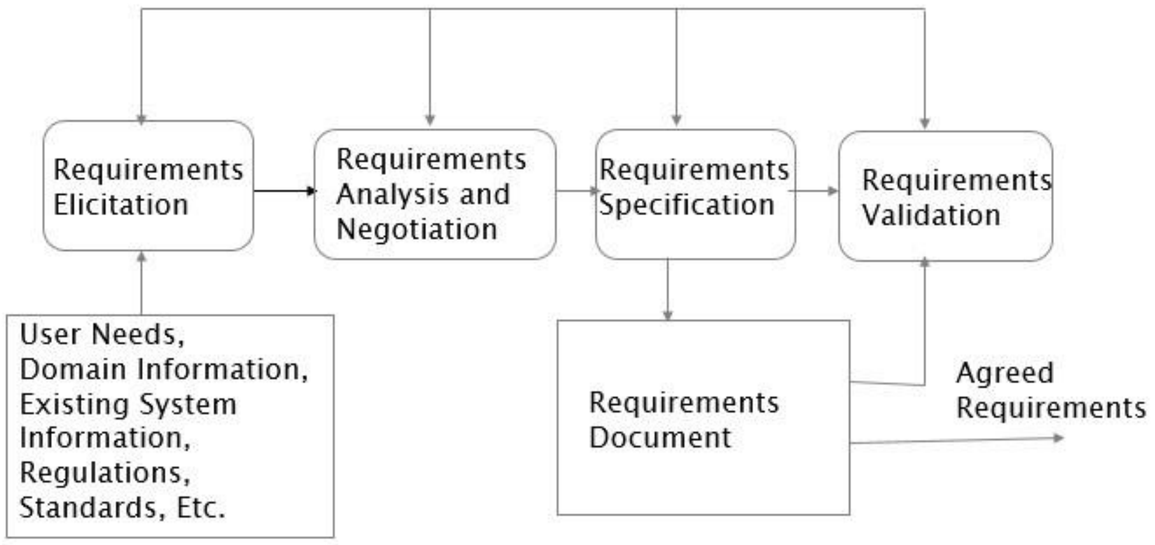
RE Process

- Generic activities which is common to all processes
- Requirements elicitation
- Requirements analysis
- Requirements validation
- Requirements management.

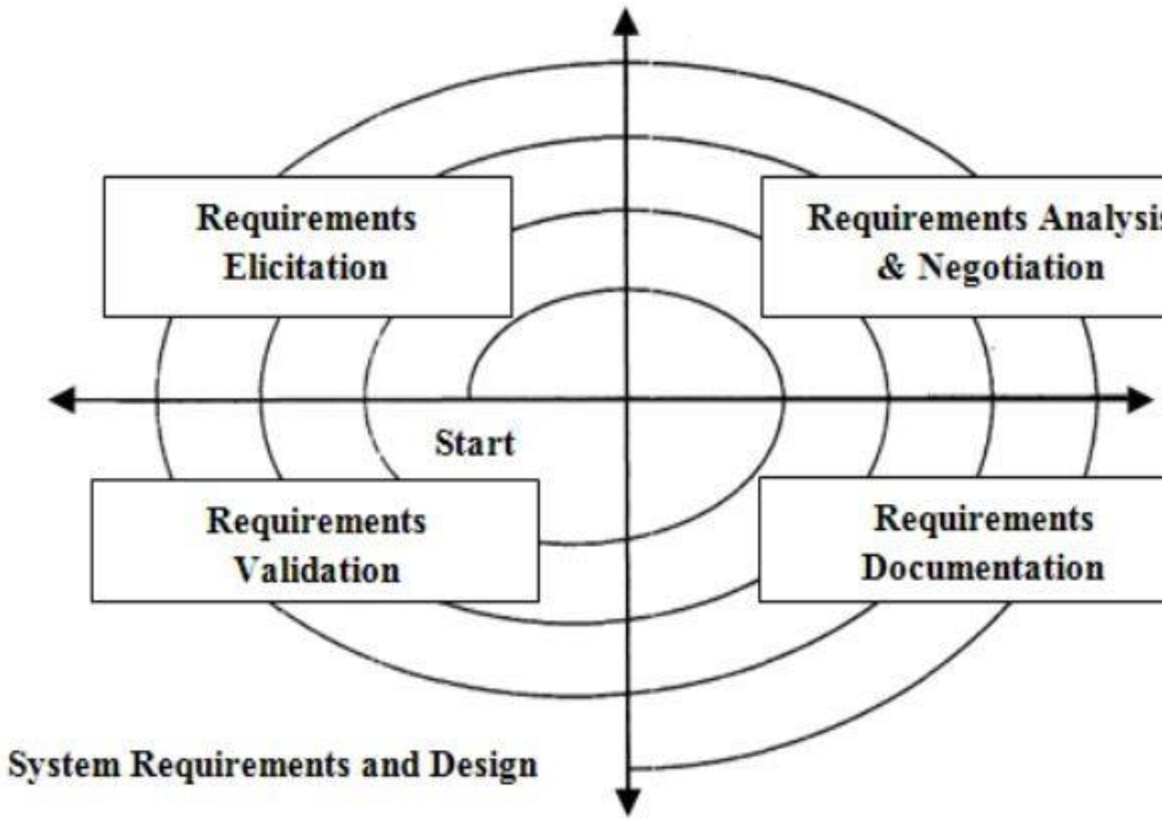
Input and Output to RE Processes



RE activities (Linear Model)


Requirements Elicitation
Requirements Analysis and Negotiation
Requirements Specification
Requirements Validation
User Needs, Domain Information, Existing System Information, Regulations, Standards, Etc.
Requirements Document
Agreed Requirements

RE activities (Spiral Model)


Requirements Elicitation
Requirements Analysis & Negotiation
Start
Requirements Validation
Requirements Documentation
System Requirements and Design

References

- Requirements Engineering: Processes and Techniques' by G. Kotonya and I. Sommerville, John Wiley & Sons, 1998
- Requirements Engineering Processes, Tools/Technologies & Methodologies S. Arif et al.

<u>LECTURE # 4 week # 4</u>

Requirements Elicitation Techniques

- Requirements Elicitation
- Purpose of Requirements Elicitation
- Basic Requirements to use
- Types of Requirements Elicitation Techniques
- Capability of Requirements Elicitation Technique

- Pros and Cons of different elicitation techniques

Requirements Elicitation

- Requirements Elicitation (RE) is defined as the process of obtaining a comprehensive understanding of stakeholder's requirements
- RE is the initial and main process of requirements engineering phase.
- RE is a complex process
- Criteria for obtaining High quality requirements

Requirements Elicitation methods Overview

- Interviews, Questionnaires, Observation, Joint Application Development (JAD), Brainstorming etc
- Which one is best?
- RE is considered an incomplete process in Requirement Engineering. • Applying inappropriate techniques

- Procedures to obtain user requirements, implement in the system to fulfill user's requirements.
- Selection of appropriate elicitation technique
- Factor (Business procedures, resources available, project type, individual preference etc)
- Characteristics of RE technique
- Type of Application

Classification of different requirements elicitation techniques

- 1) **Traditional Technique**

Interviews, Questionnaires/Surveys, and Document analysis.

- **2) Contextual Techniques**

Observation, Ethnography and Protocol Analysis.

- 3) **Collaborative/Group Techniques**

Prototyping, Joint Application Development, Brainstorming, and Group Work

4) Cognitive Techniques

Laddering, Card Sorting, Repertory Grids and Class Responsibility Collaboration

Interviews

- Basic concepts

Verbal method, easy and effective, most employed

Types of Interviews:

Structured or Closed Interviews:

General characteristics

Predefined questions, quantitative data, No generation of new idea,

- Pros:
- No biasing ,Few additional questions may be added to further add clarification, Interview can be repeated

Cons:

- interviewee may be uncomfortable

Semi-structured Interviews:

- combination of predefined and unplanned questions.
- Pros:
- Consistency,
- interviewee can share new ideas

Cons:

- Time consuming, interviewer may lose its focus ,Training required,
- Findings are hard to generalize

Unstructured or Open Interviews

- informal interview containing unplanned questions, Producing qualitative data
- Pros:
- New ideas and opinions are generated.
- Due to informal approach interviewer may feel ease to properly answer questions.

- Cons:
- Interviewer can be biased in asking questions.
- Difficult to repeat in case data reliability is checked.

Summary: Interviews

- Advantages:
- Good for complex topic, Rich in information, Ambiguities are clarified. Interviewer can analyze emotions .Non-responsiveness remains low. Provides overview of the whole system.
- Disadvantages:
- Small number of people involved, Information cannot be gathered from large population, Quality of data gathered depends on the skills of interviewer,

Document Analysis

- Analyzing and gathering information from existing documents
- Effective to initiate requirements elicitation process
- Why use this technique?
- An expert needs to study domain information thoroughly for the purpose of adapting when existing system needs to be replaced or enhanced.
- design documents, templates and manuals of existing systems.

Document Analysis

- Pros:
- · Helpful when stakeholders and users are not available.
- · Helps business analyst to get proper understanding of the organization before meeting the stakeholders there.
- · Provides useful historical data.
- · Can be useful to frame questions for interviews.
- · Can be used for requirements reuse.
- · Inexpensive technique.

Document Analysis

- Cons:
- · Time consuming to find information from huge amount of documentations.

- · Sometimes valid information may not be available i.e. documents may be outdated.
- · Periodic updation of documents is required.
- · Information might be incomplete.

Questionnaires/Surveys

-

  cheapest way of eliciting requirements

- When to use this technique?
- No face to face
- collect requirements from a larger group of population distributed over a large geographical area and from different time zones
- Questionnaires must be clear, well-defined and precise besides including the domain knowledge

Pros:

- · Reach large number of people within a short time.
- · Useful when same question is asked to large number of people.
- · No biasing occurs.
- · It is economical.
- · Easy because multiple choice questions or true false or fill in the blanks are included.
- Cons:
- · Cannot get further clarification regarding the problem what analyst actually wants from the user.
- · Questions can be misinterpreted.
- · Sometimes useful feedback isn't received.
- · To get further information other techniques like interviews can be used as follow ups.
- · Sometimes question ambiguities may arise.
- · Used for general purpose software.

Introspection

- Analysts work for what they imagine and observe by themselves how a system design should be.
- This technique is effective with users who have a lot of experience of their own fields but have less knowledge about the other fields as well as the new system.

Pros:

- · There are almost no costs for implementing this technique.
- · Easy to implement.

- · It can act as a good initial step to start requirements elicitation.
-

Cons:

- · It is hard for analysts to imagine the environment in which the new system works.
- · It doesn't allow discussion with stakeholders and other experts. Therefore, it is not encouraged if not used in combination with other techniques.
- · Analysts and stakeholders need to be well known about the domain.

LECTURE # 5 WEEK # 5

Contextual Techniques

- Techniques in this category are Observation, Ethnography and Protocol Analysis.
- Observation/Social Analysis :
- The requirements engineer observes the user's environment without interfering in their work.
- This technique is used when customer is not able to explain what they want to see in the system, how they work and when some ongoing processes are to be monitored.
- combination with other requirements elicitation techniques like interviews.
- Passive observation
- Active observation

Observation

- Pros:
- · Authentic and reliable because analysts by himself goes to observe the environment.
- · Can be useful to confirm and validate requirements collected through other methods.
- · It is inexpensive method.
- Gives idea about how users will interact with the system.
- · Helpful in work measurements i.e. how long particular task takes to be done
- Cons:

-

- · All the requirements cannot be checked in just a single session; multiple sessions may be required.
- · Users can behave indifferently while they are interrupted for asking questions in active observation.
- · In passive observation, it is difficult for analyst to make out why some decisions are made.

  · It is time consuming

Ethnography

- Study to understand Relationships between actors, workplace
- Used in combination with other elicitation techniques like interviews and questionnaires
- Pros:
- · Helps to discover certain features of a work place in a shorter time period.
- · Helps understand how people work in an organization and how they interact with each other.
- · Doesn’t need much resources to be effective.
- · Helps reveal critical events not observed by any other technique.
- · Useful in validating requirements
- Cons:
- · There is no detailed guide on how to perform ethnographic technique effectively and therefore, it all depends on the skills of the person performing it, the ethnographer.
- · It requires engineers to have a lot of experience to perform it.
- · New and unique features added to the system might not be discovered.
- · Fails to produce desirable results due to diverse population.
- · Focuses mainly on end-users.
- · Sometimes it can be time consuming.
- · Different backgrounds of users and ethnographers can result in misunderstanding problems between them.

Collaborative/Group Techniques

- Group elicitation techniques involve teams or groups of stakeholders who applying their individual expertise on a particular issue agree upon a set of decisions
- Prototyping :

- An iterative process
- Pros:
- · User involvement during development process.
- · Allows early user feedback for requirements refinement.
- · Saves development time and cost.
- · Users and analysts get better understanding of the system.

Collaborative/Group Techniques

- Cons:
- · The disadvantage is that when users get used to particular kind of system they often resist changes.
- · Effort and cost estimation may get high as calculated earlier.
- · For complex systems, it can be time consuming.

Joint Application Development (JAD)

- JAD sessions are basically collaborative workshops that last for 4-5 days and whose outcome is a proper set of user requirements.
- Pros:
- · Decreased time and cost of requirements elicitation.
- · Accelerates design of the system
- New and rapid idea generation leading to creative outputs.
- · Promotes user feedback.
- · More user satisfaction.
- · Good communication between stakeholders, analysts and other professionals.
- · Visual aids and case tools used make the session interactive.
- Cons:
- · If not properly planned can lead to wastage of time and resources.
- · Requires trained facilitators.
- · Requires lots of planning and effort.
- · It is an expensive technique.

-

Brainstorming

- It is an informal discussion where free expression of ideas is given to every participant for a new kind of system to be developed
- Pros:
- · Costs very little and not much resources are needed.
- · Participants need not to be high qualified and each participant takes part actively in the process.
- · It is comprehensible and easy to implement.

· Helps in new ideas generation.

· Helps in conflict resolution.

· Each participant is equally allowed to speak and share ideas.

- Cons:
- · It is not suitable to resolve major issues.
- · If not organized properly can be time consuming.
- · Quantity of ideas doesn't always equal their quality.
- · Can lead to repetition of ideas if participants are not paying proper attention.
- · Some people due to extrovert nature may take over all the session and all the time sharing their ideas and other people who are less outgoing will be afraid to take the time sharing their views.

Group Work

- In this technique, stakeholders are invited to attend a meeting to elicit requirements for projects
- Pros:
- · Quality requirements in a shorter period of time.
- · Saves cost as compared to conducting interviews of same number of people
- Cons:
- · It takes lot of effort to bring all the stakeholders on the same table at the same time because of their busy schedule and political aspects
- Participants may have issues related to trust and may feel hesitated to discuss critical or sensitive matters.
- · Members may get influenced by dominant people in the meeting leading to biased results.
-

User Scenarios

- Scenarios are representation of user's interaction with the system. It is a real world example of how a system is used.

-
-
-
- Pros:
- · Well-developed scenario helps organizations to be proactive and work specifically for the desired product.
- · Gives good clarifications regarding an activity or event its normal flow, exceptional behavior, alternative paths.

  · People with no technical knowledge can also understand it.

  · Easy to understand as no special language is used to write them.

  · Ensures system is designed properly as end-user's perspective is considered for requirements elicitation.

- Cons:
- · It is difficult to draw useful scenarios.
- · It is not suitable for all types of projects even if they capture more requirements.
- · They do not cover all the processes i.e. not the complete view of future system.

Cognitive Techniques •

Laddering

- It is an interviewing technique to elicit stakeholder's goals, values and attributes.
- Pros:
- · Easy to understand requirements because of hierarchical nature.
- · Reuse of requirements saves time and cost.
- · Not good for building a new system.

Laddering

- Cons:
- · Maintaining requirements is a difficult task while adding or deleting any user requirement anywhere in a hierarchy.
- · Technique becomes complex when requirements are in large number.
- · Expert opinion or initial data is must to elicit requirements.

- · It is too long and tiring technique

Card Sorting

- It is a knowledge elicitation technique in which stakeholders are asked to sort cards according to domain entity names using index cards or some software packages.
- Pros:
- · It is fast and inexpensive.
- · It is accessible through internet so the participants that are geographically remote can take part in it.
- · It is reliable and easy technique.

  · Helpful in providing good understructure.

  · It is an established technique.

  · Useful in gathering qualitative data.
- · It involves real inputs from the users.
- · Makes information structured to be fed into information process

LECTURE # 6 WEEK # 6

Requirements Modeling

Modeling

- A picture is worth 1000 words.
- A model is a representation of reality, like a model car, airplane.
- Most models have both diagrams and textual components.

•

•

•

ROOM 8.49
TERRACE
LIVINGROOM
31.0
4.15
WC
2.32
KITCHEN
8.18

Why Modeling?

- Visualization.
- Communicate with customer.

- Reduction of complexity.

Requirements modeling

- A requirements model is a set of these diagrams, each of which focuses on a different aspect of the users' needs.
- A requirements model provides greatest benefit if you use it to focus discussions with the users or their representatives.
- Each model provides a particular type of information.

Requirements modeling

Diagram / Model	**What it describes in that model**
Use case diagram	Who uses the system and what they do with it.
Activity diagram	Flow of work and information between activities performed by users and system or its parts.
Sequence diagram	Sequence of interactions between users and system or its parts.
Finite state machine	View of the possible states of a system or object and the changes between states that can take place under certain circumstances.

Why Requirements modeling?

- Modeling can guide elicitation.
- Modeling can provide a measure of progress.
- Modeling can help to uncover problems.
- Modeling can help us check our understanding.

Use case diagram

- A use case is a list of actions/tasks.
- Who uses the system and what they do with it.

-
-
-
- Use case diagram can identify the different types of users of a system and the different use cases

State machine diagram

- One of the challenges faced by requirements analysts is the need to communicate the complex behavior of systems in an understandable yet rigorous and verifiable way.
- State machine works well for this purpose.
- State machine captures information about states an object can go through during its lifecycle.

Summary

- Modeling.
- Benefits of Modeling.
- Requirements Modeling.
- Benefits of requirements modeling.
- Use case diagram and State machines.

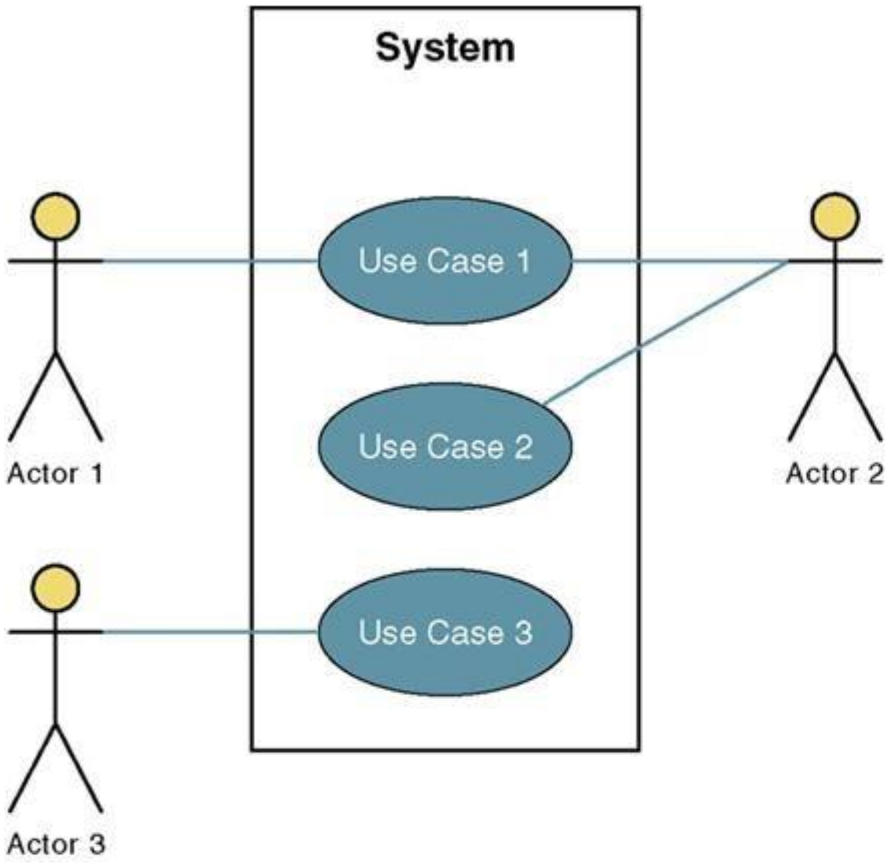
<u>LECTURE # 6 WEEK # 6 SLIDE 2:</u>

Use case modeling
(Part 1)

Use case modeling

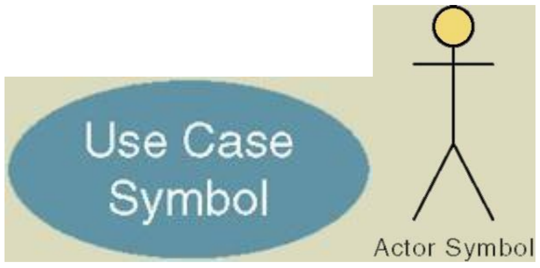
- Use case diagrams describe what tasks the system performs.
  - E.g. Order placement, a ticket reservation, assignment submission etc.
- Who uses the system
  - A customer, a librarian, a student etc.
- Which user interacts with which use case.

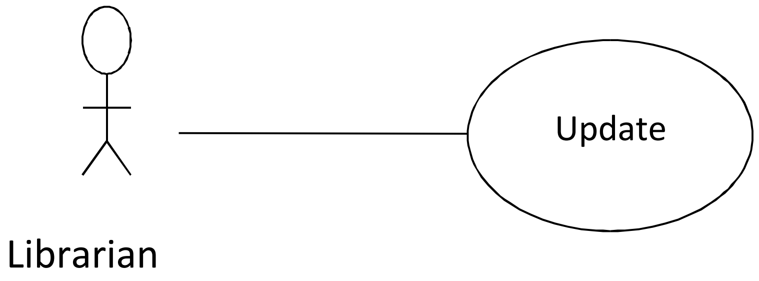
Sample Use case model



Components of use case

- Use case: subset of the overall system functionality.
- Actor: Anyone or anything that needs to interact with the system to exchange information.
- Association: which actor interacts with which use case.



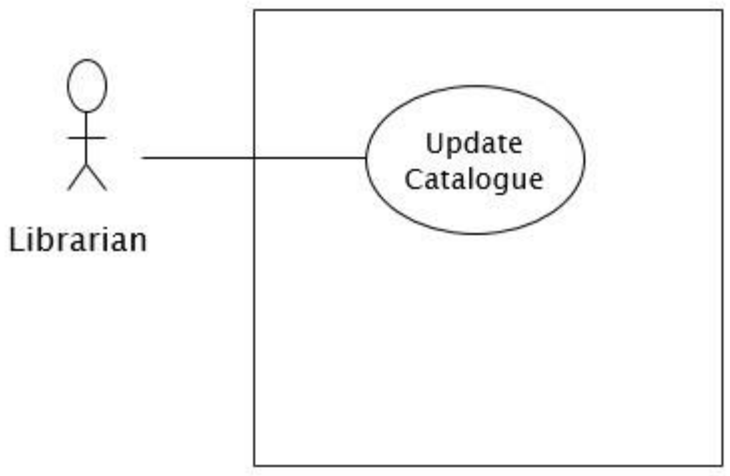


-
-
-

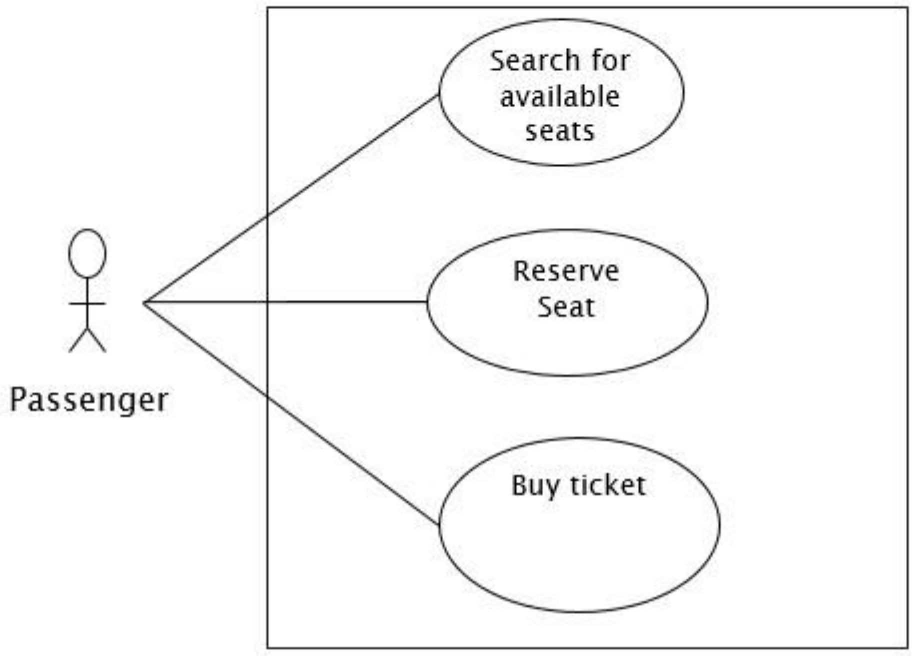
Sample use case diagram

- A Librarian updates a book catalogue

- A Librarian updates a book catalogue

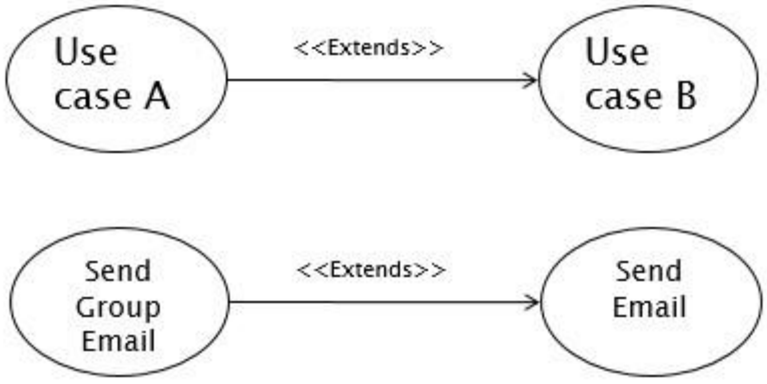


- A passenger buys ticket



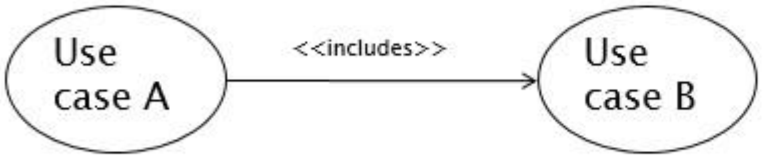
Reuse (dependency) in use case

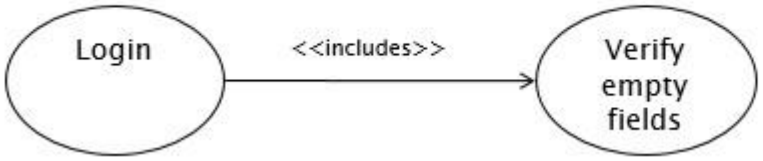
- Extends: An extend dependency, formerly called an Extends relationship is a generalization relationship where an extending use case continues the behavior of a base use case



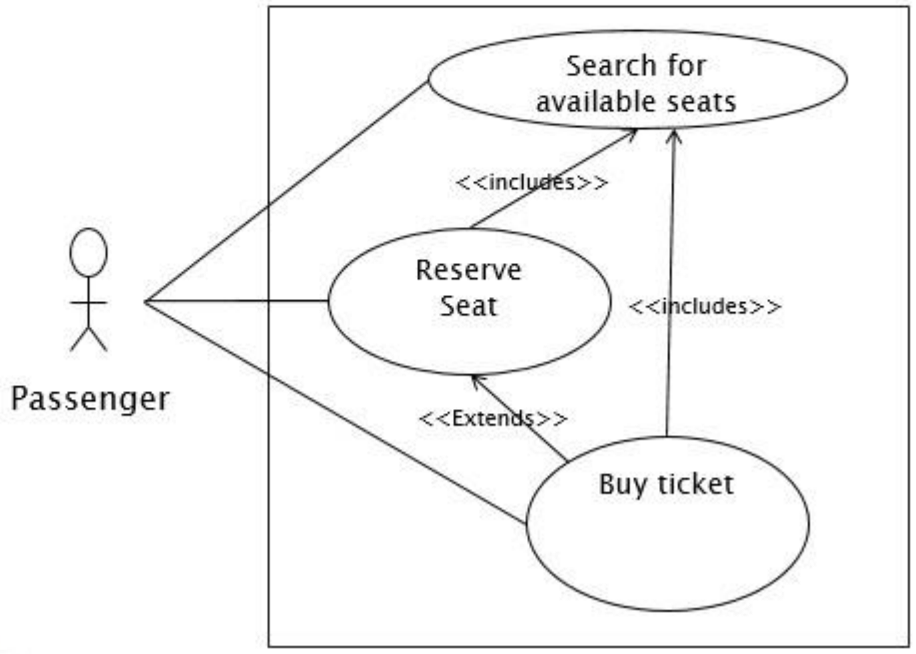
- Includes: An include dependency, is a generalization relationship denoting the inclusion of the behavior described by another use case.

-
-
-





Sample use case diagram



Includes vs Extends

▸ Includes ◦ You have a piece of behavior that is similar across many use cases ◦ Break this out as a separate use-case and let the other ones "include" it. ◦ <<Includes>> keyword is used.	▸ Extends ◦ A use-case is similar to another one but does a little bit more ◦ Put the normal behavior in one usecase and the exceptional behavior somewhere else. ◦ <<Extends>> keyword is used

Summary

- Use case modeling.
- Components of use cases
- Reusability in use cases.

<u>LECTURE # 7 WEEK# 7 Slide # 1</u>

Use case modeling (Case Study)

<u>Air Ticket Reservation System</u>

- <u>Reservations on local system</u>
- <u>Passenger goes to client terminal in local office</u>
- <u>Searches flights/seats.</u>
- <u>Takes print of available seats.</u>
- <u>Booking staff confirms seat.</u>
- <u>Client terminal also displays flash news/updates.</u>
- <u>Admin can Add/Edit/Cancel flight schedule (Email is sent to passengers)</u>
- <u>Admin can cancel ticket.</u>
- <u>Admin can Add/Edit/Cancel Reservation</u>

<u>Air Ticket Reservation System</u>

- <u>Actors</u>
  - <u>Passenger</u>
  - <u>Admin</u>
  - <u>?</u>

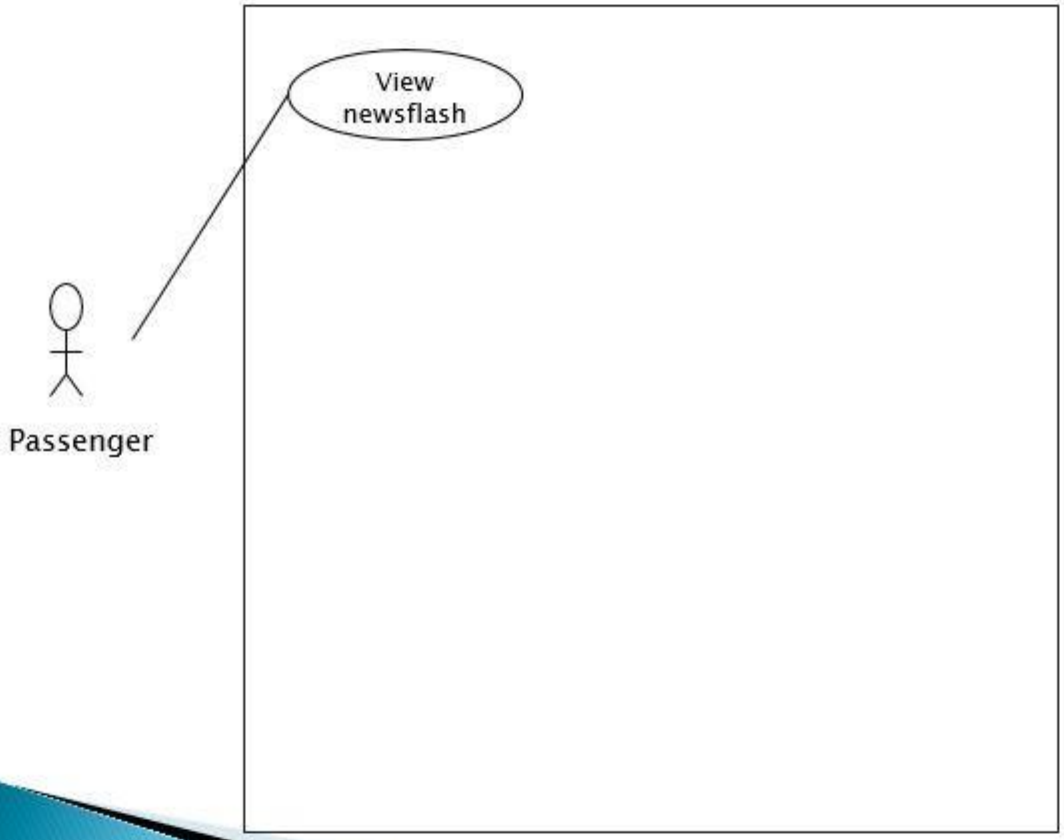
<u>Air Ticket Reservation System</u>

- <u>Use cases</u>
  - <u>ViewNewsFlash</u>

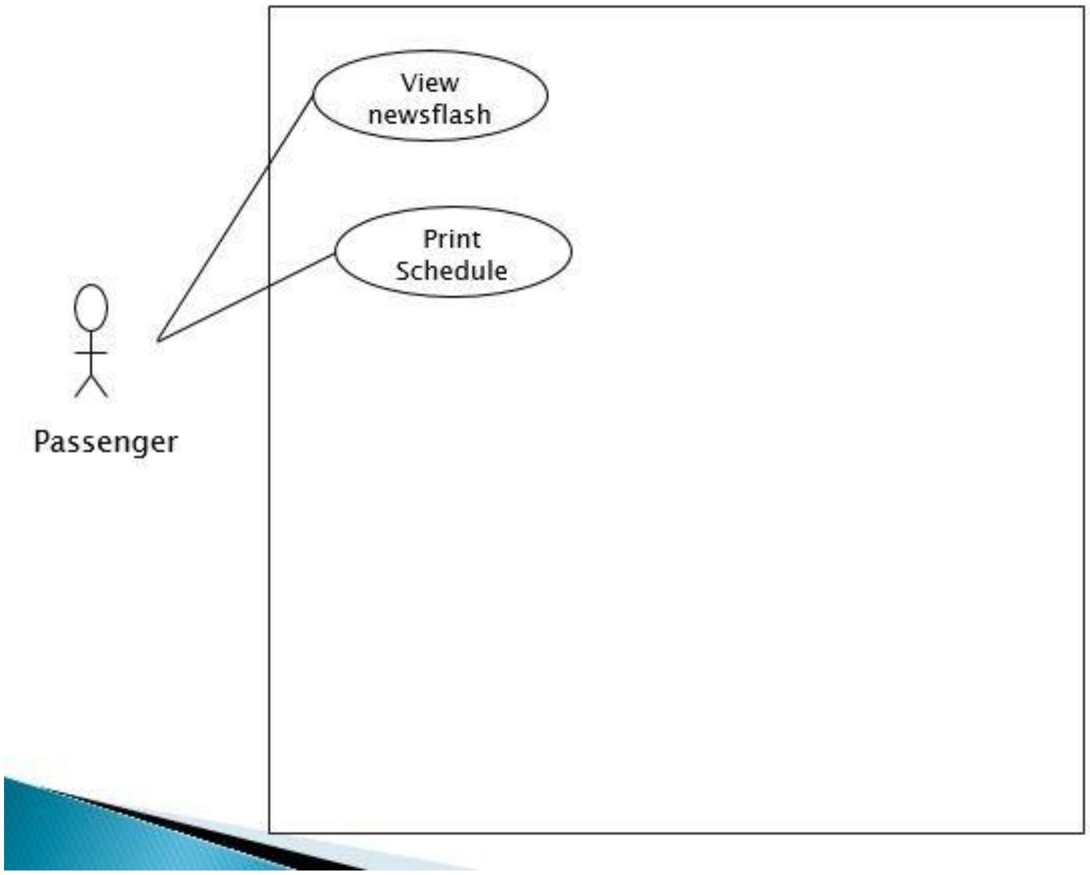
-
-
-
    - PrintSchedule
    - SearchSeat
    - AddFlight
    - ReserveSeat
    - EditReservation
    - CancelReservation

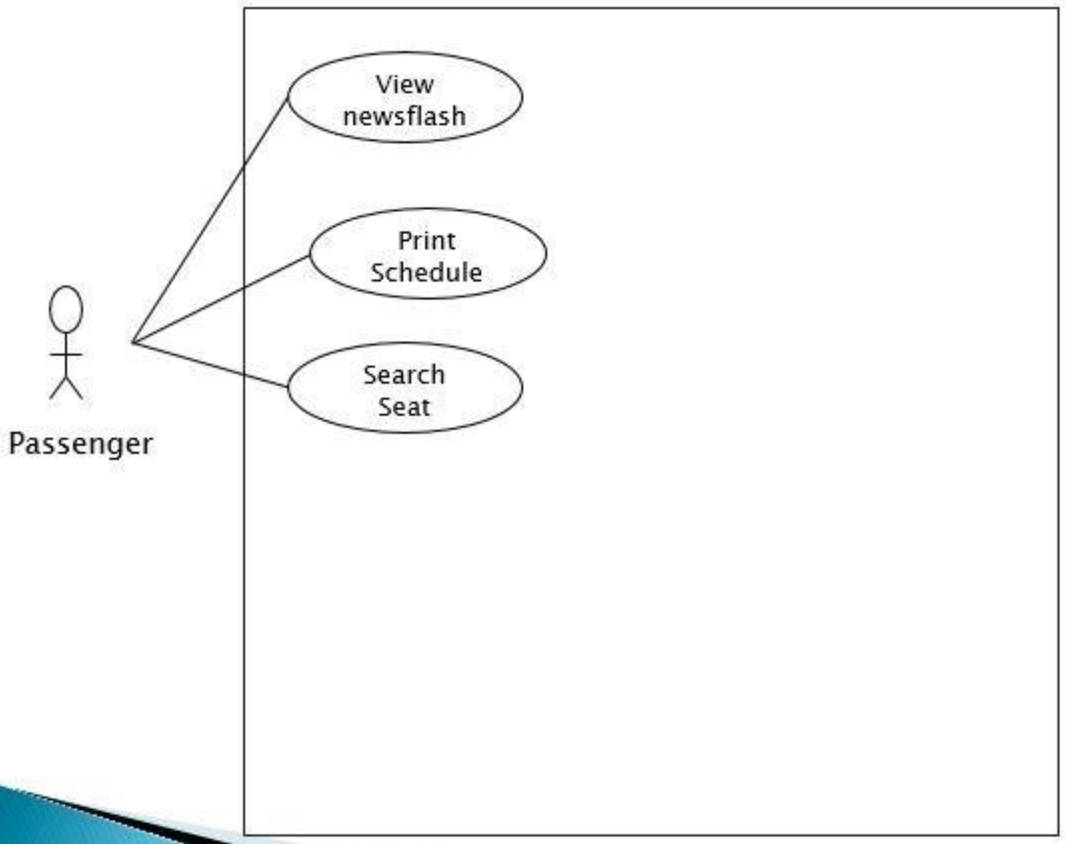
Air Ticket Reservation System

- Use cases
    - SendEmail
    - AddFlight
    - EditFlight
    - CancelFlight
    - AddUser
    - EditUser
    - DeleteUser


View newsflash
Passenger

-
-
-

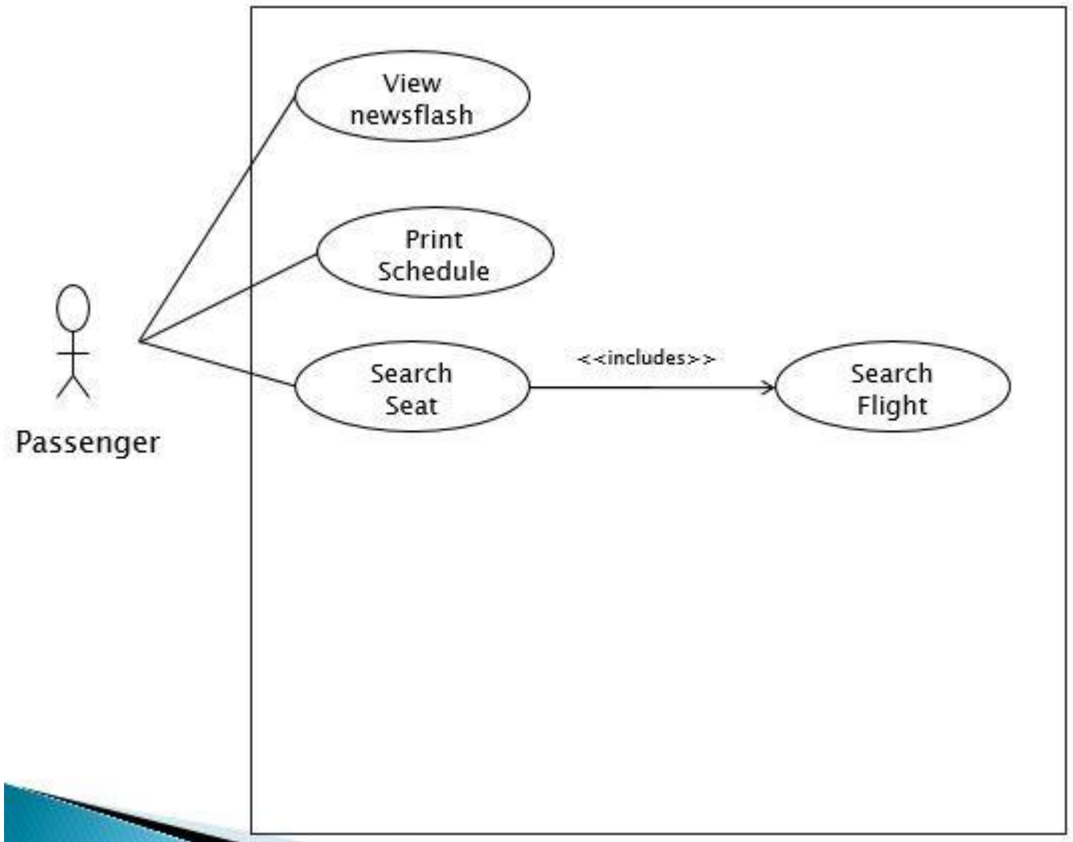

View newsflash
Print Schedule
Passenger

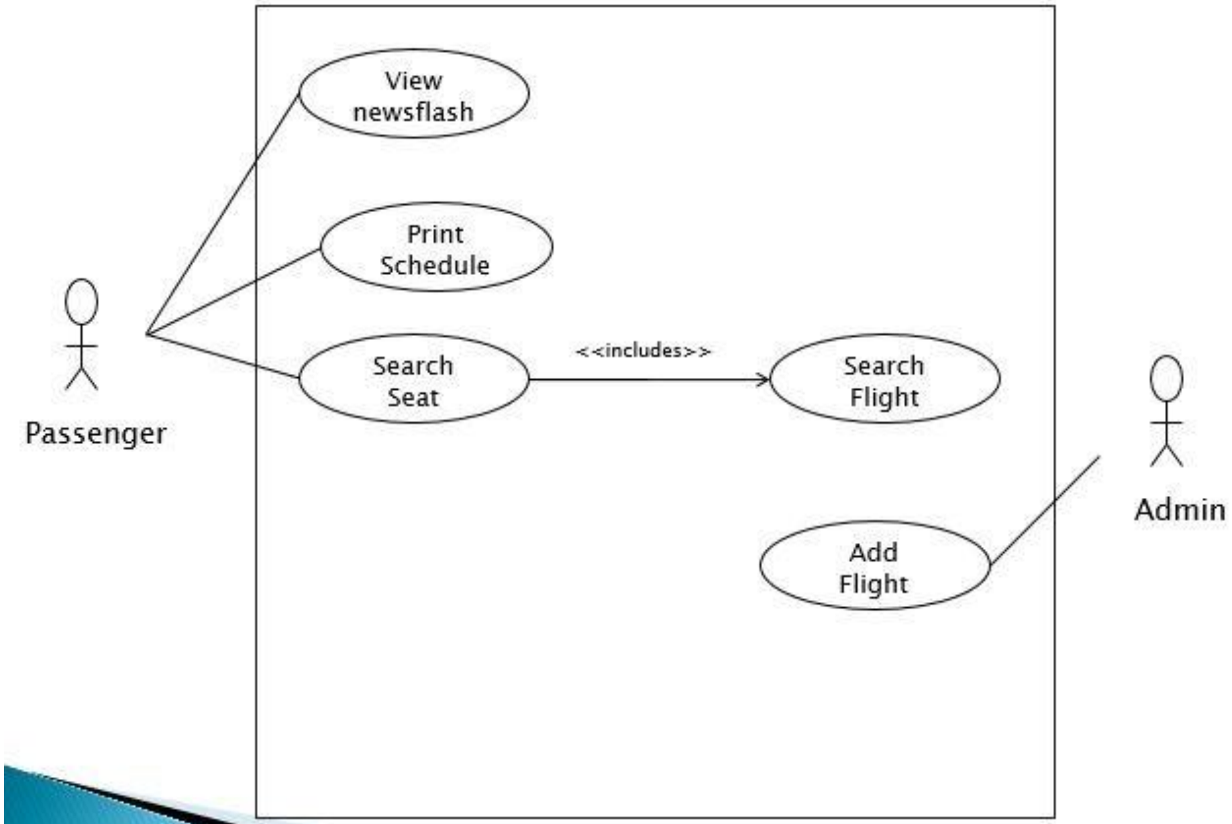

View newsflash
Print Schedule
Search Seat
Passenger

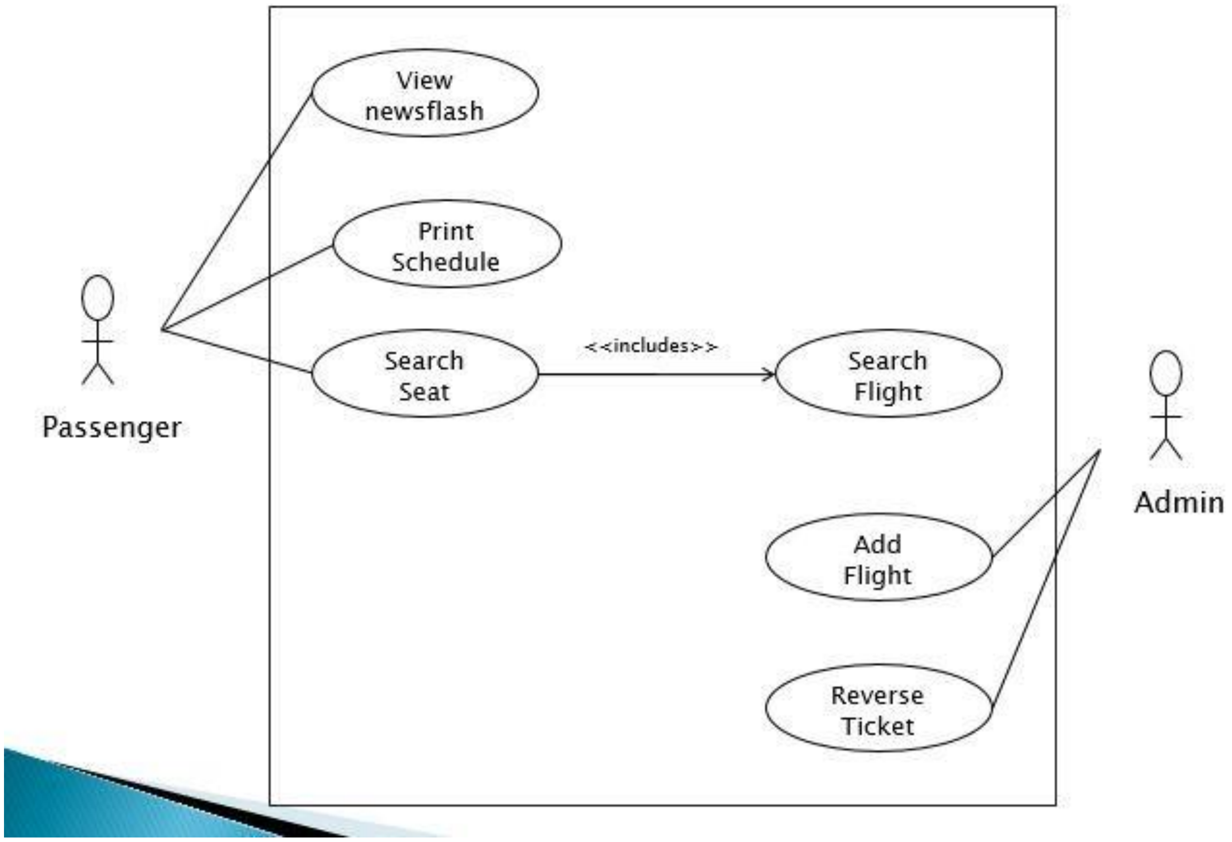
•

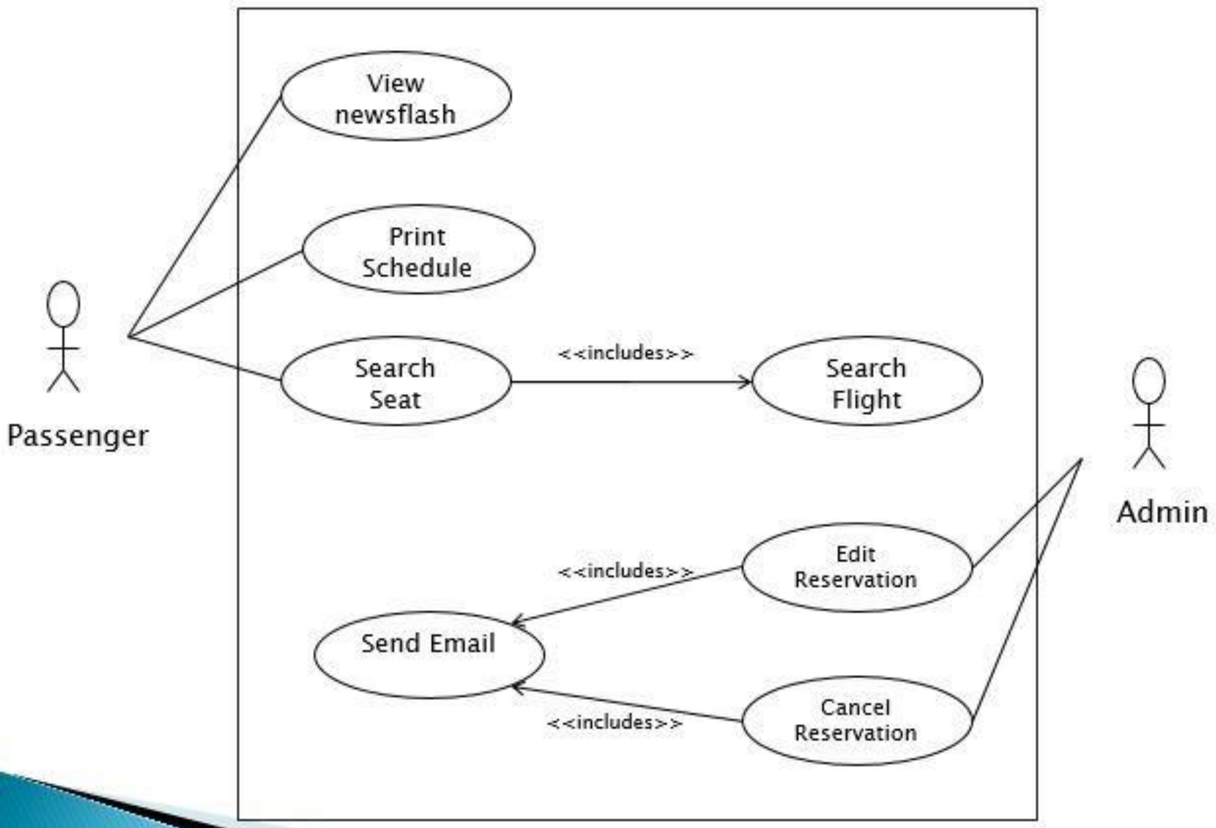
•

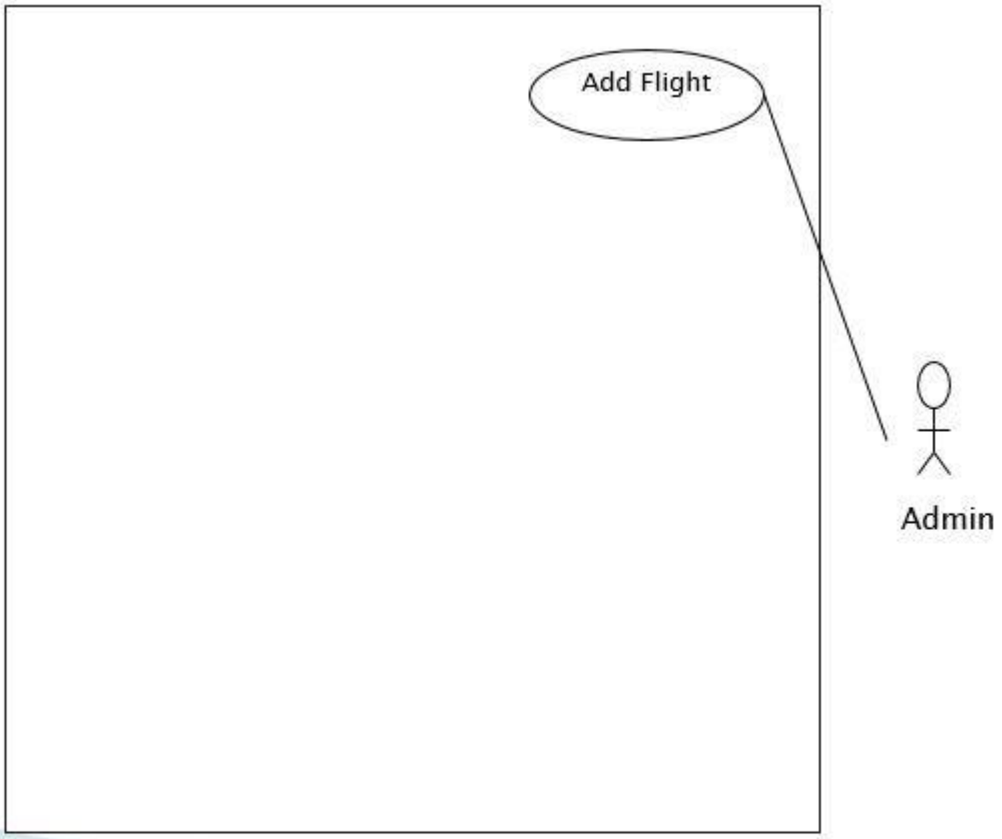
•

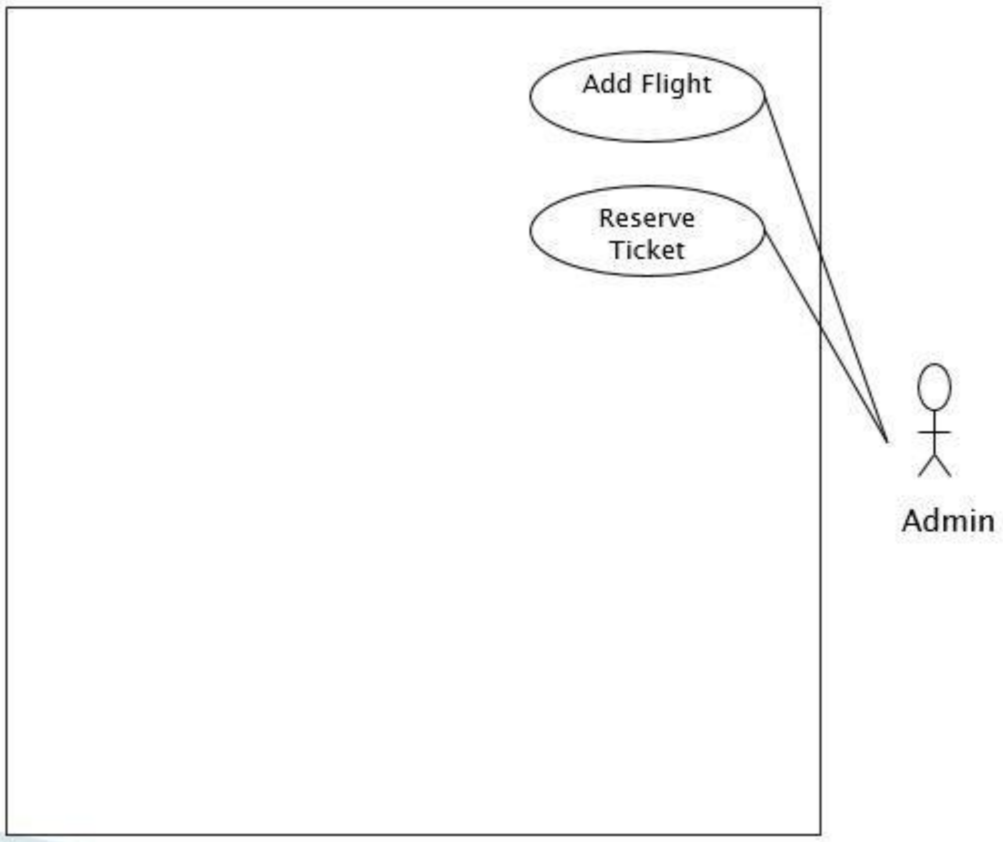

View newsflash
Print Schedule
Search Seat
<<includes>>
Search Flight
Passenger

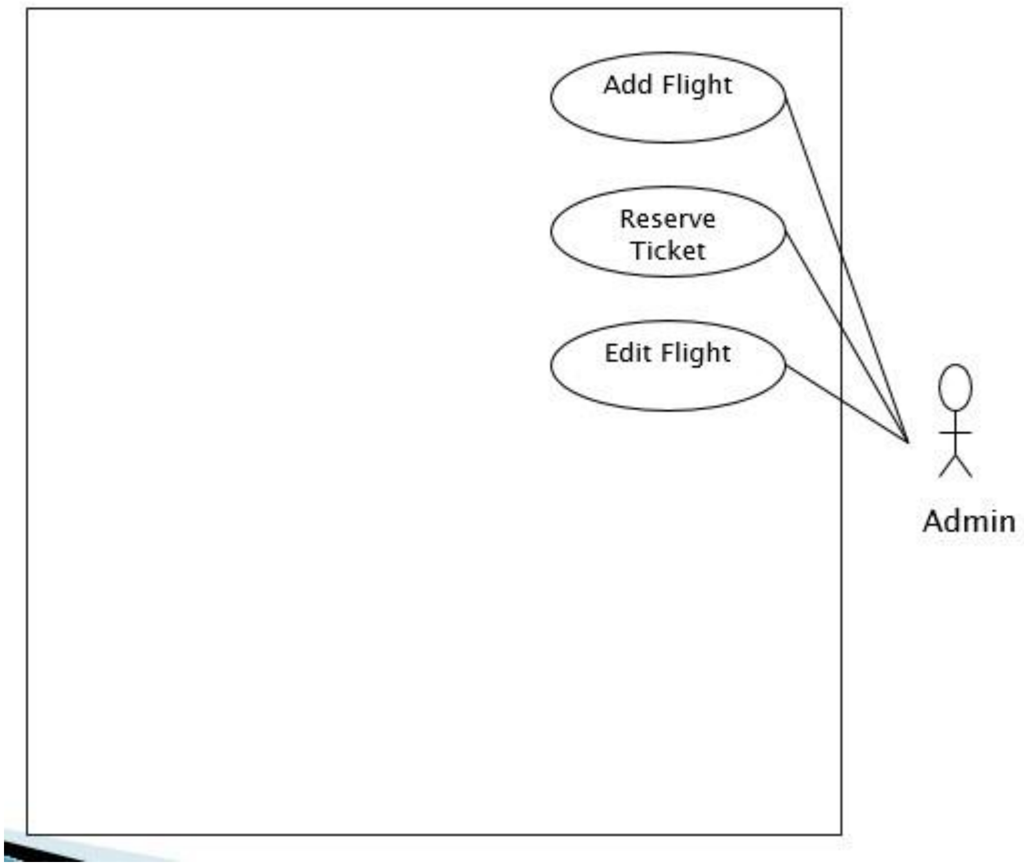

View newsflash
Print Schedule
Search Seat
<<includes>>
Search Flight
Add Flight
Passenger
Admin

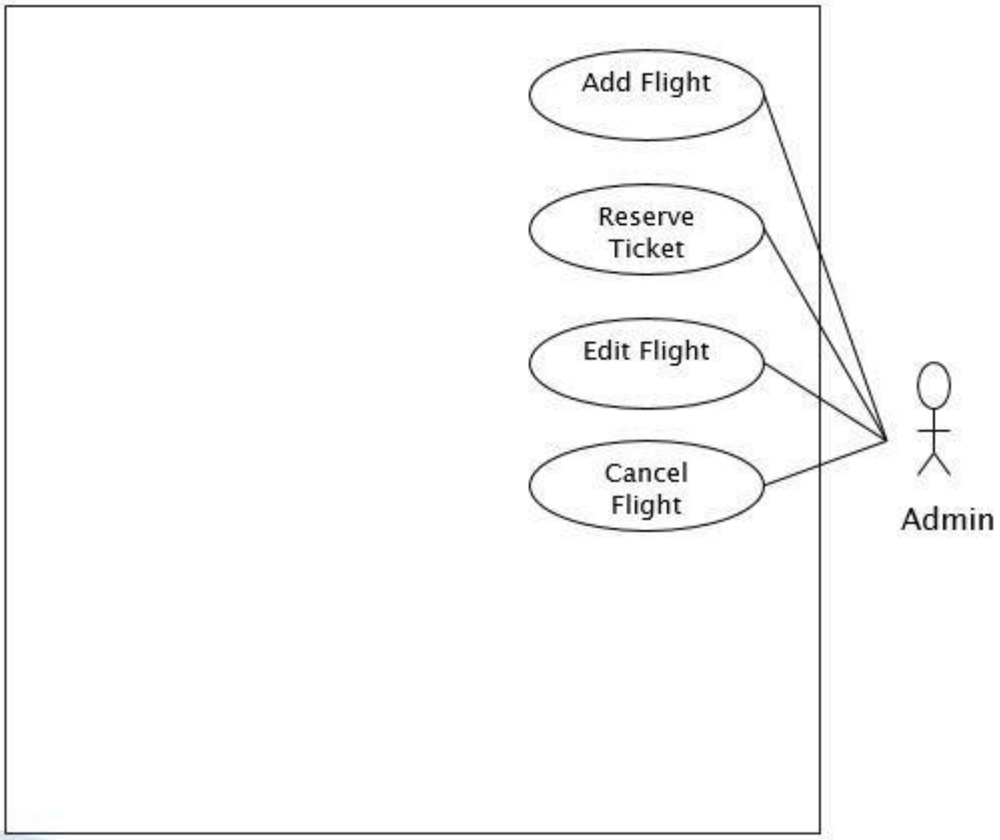

View newsflash
Print Schedule
Search Seat
<<includes>>
Search Flight
Passenger
Admin
Add Flight
Reverse Ticket

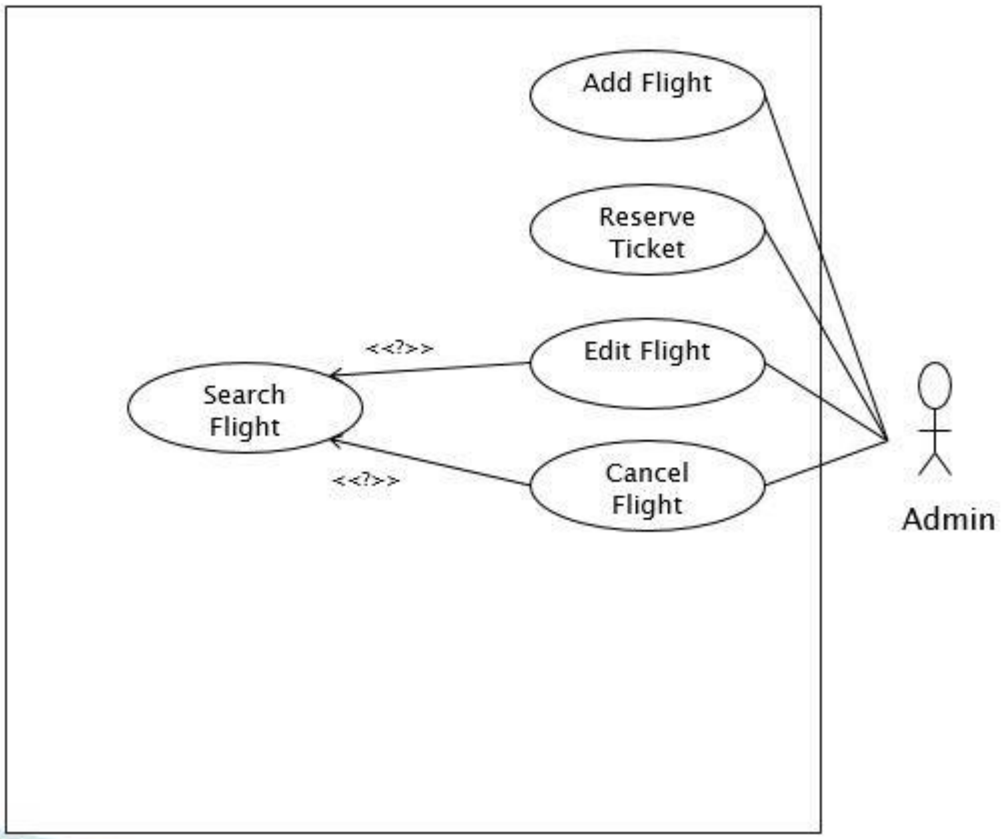

View newsflash
Print Schedule
Search Seat
<<includes>>
Search Flight
Passenger
Admin
Edit Reservation
<<includes>>
Send Email
<<includes>>
Cancel Reservation

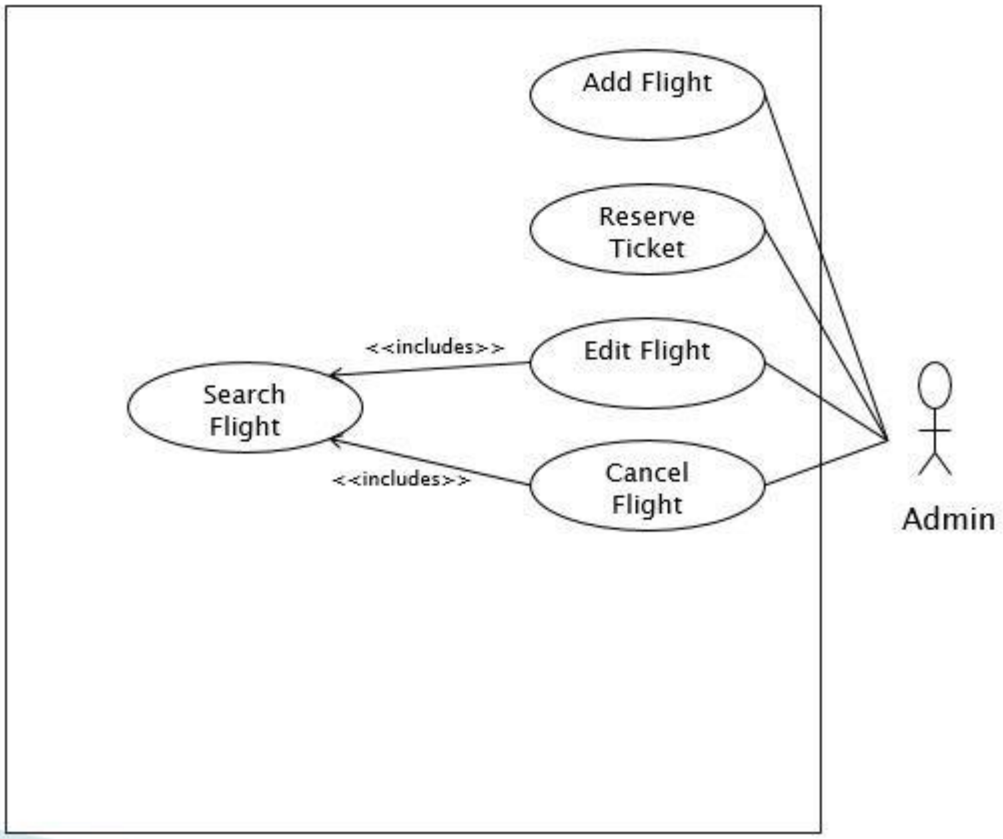

Add Flight
Admin


Add Flight
Reserve Ticket
Admin

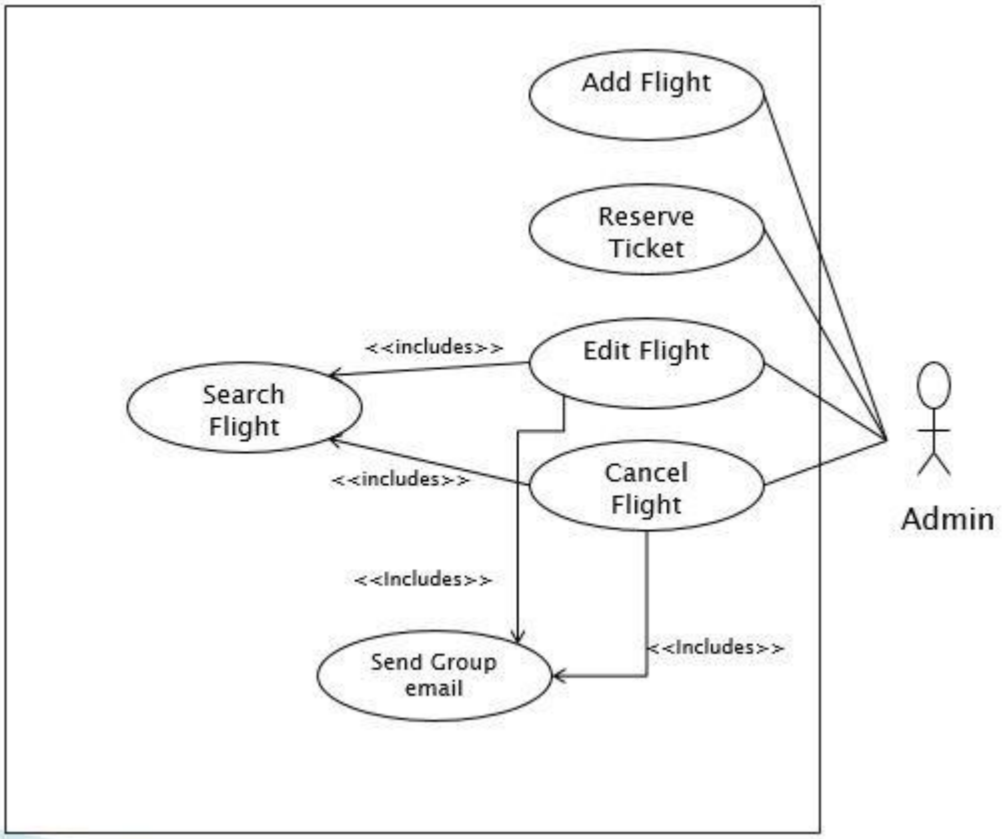

Add Flight
Reserve Ticket
Edit Flight
Admin


Add Flight
Reserve Ticket
Edit Flight
Cancel Flight
Admin

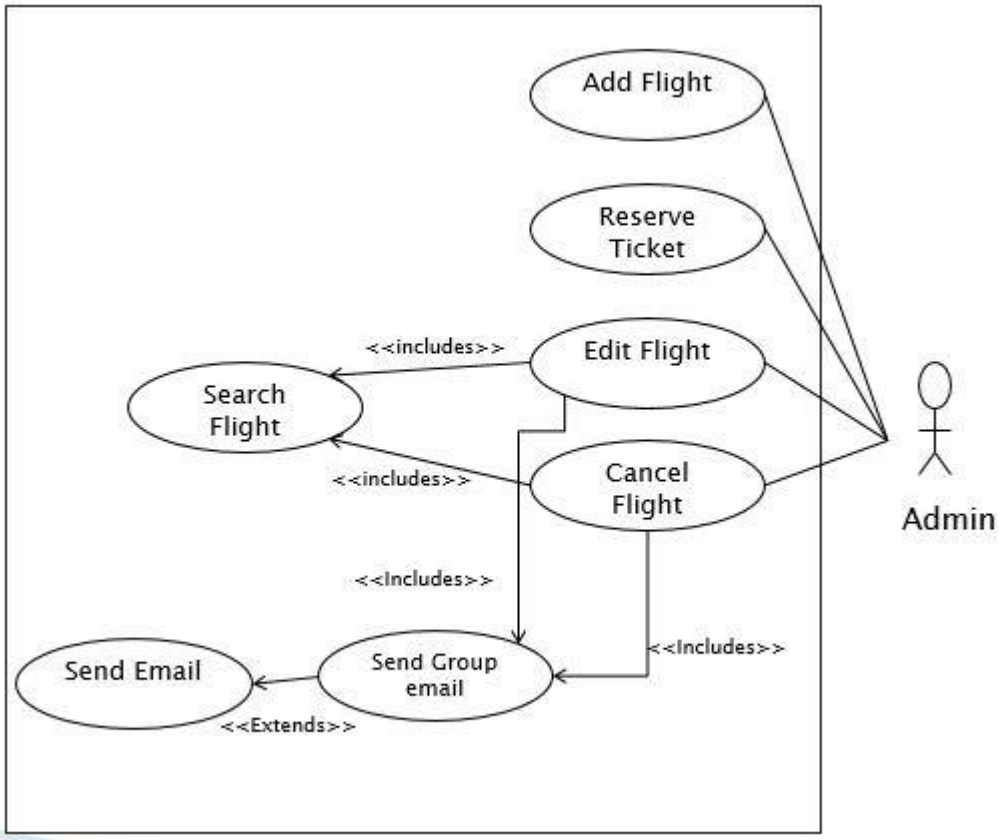

Add Flight
Reserve Ticket
<<?>>
Edit Flight
Search Flight
<<?>>
Cancel Flight
Admin

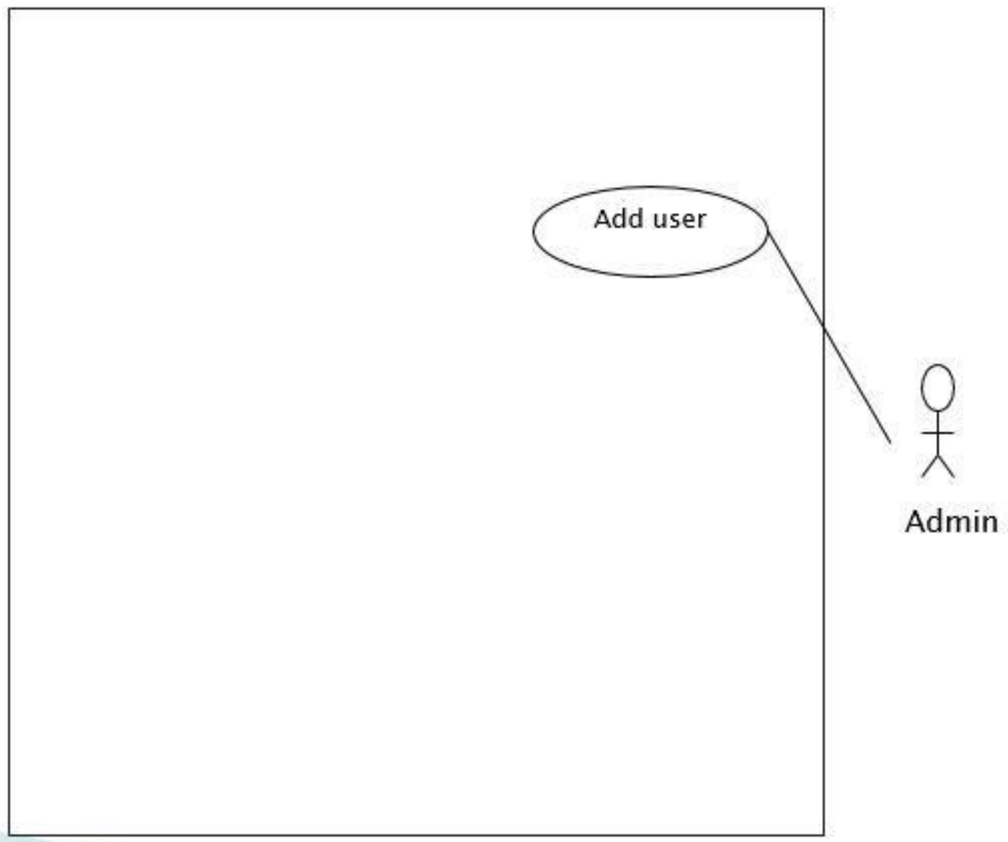

Add Flight
Reserve Ticket
<<includes>>
Edit Flight
Search Flight
<<includes>>
Cancel Flight
Admin

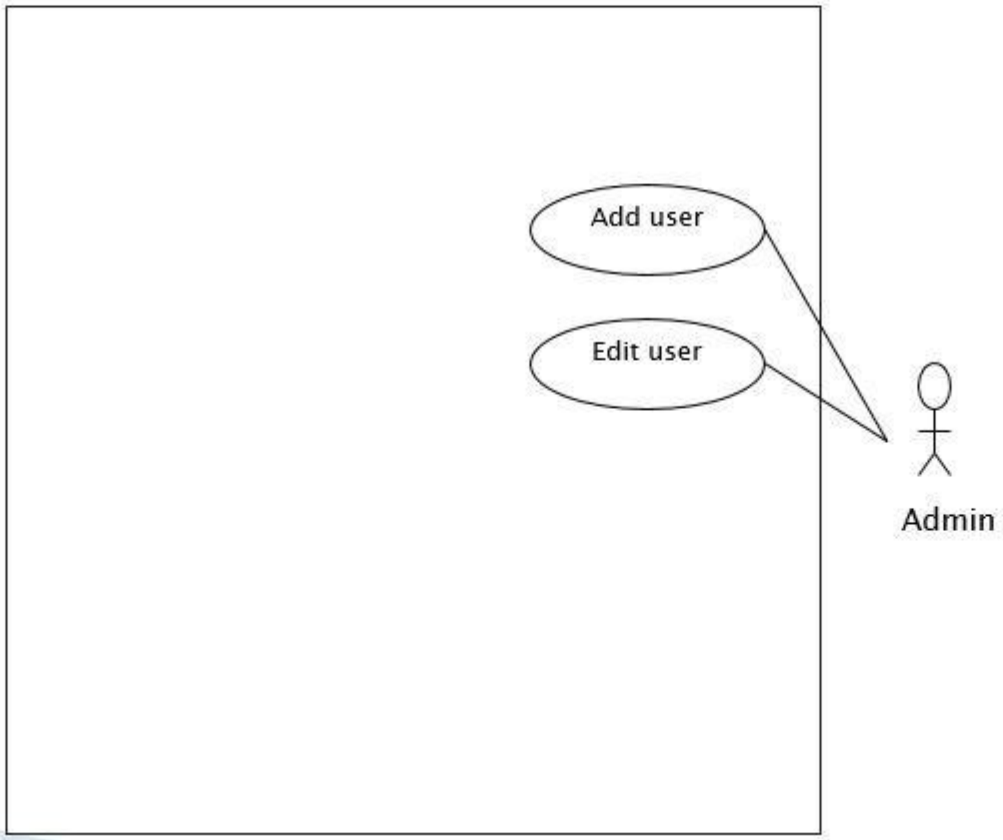
-
-
-



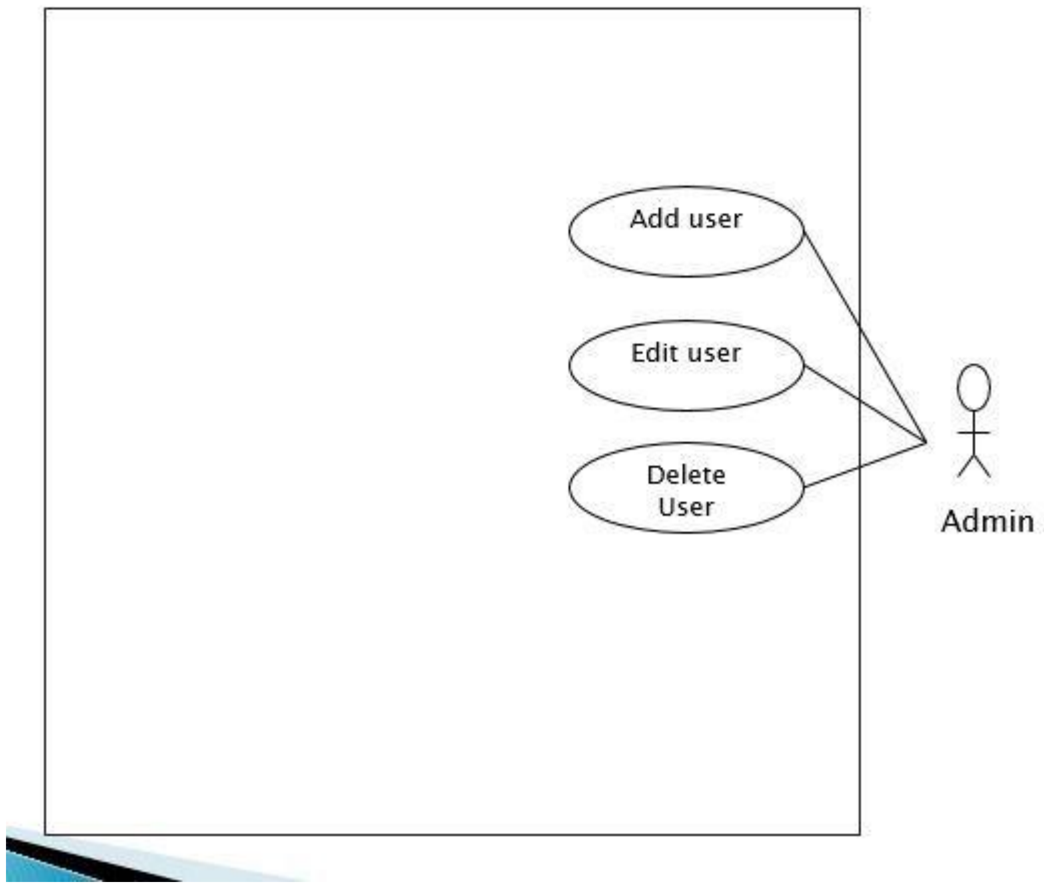
Add Flight
Reserve Ticket
Edit Flight
Search Flight
Cancel Flight
Send Group email
<<includes>>
<<includes>>
<<Includes>>
<<Includes>>
Admin

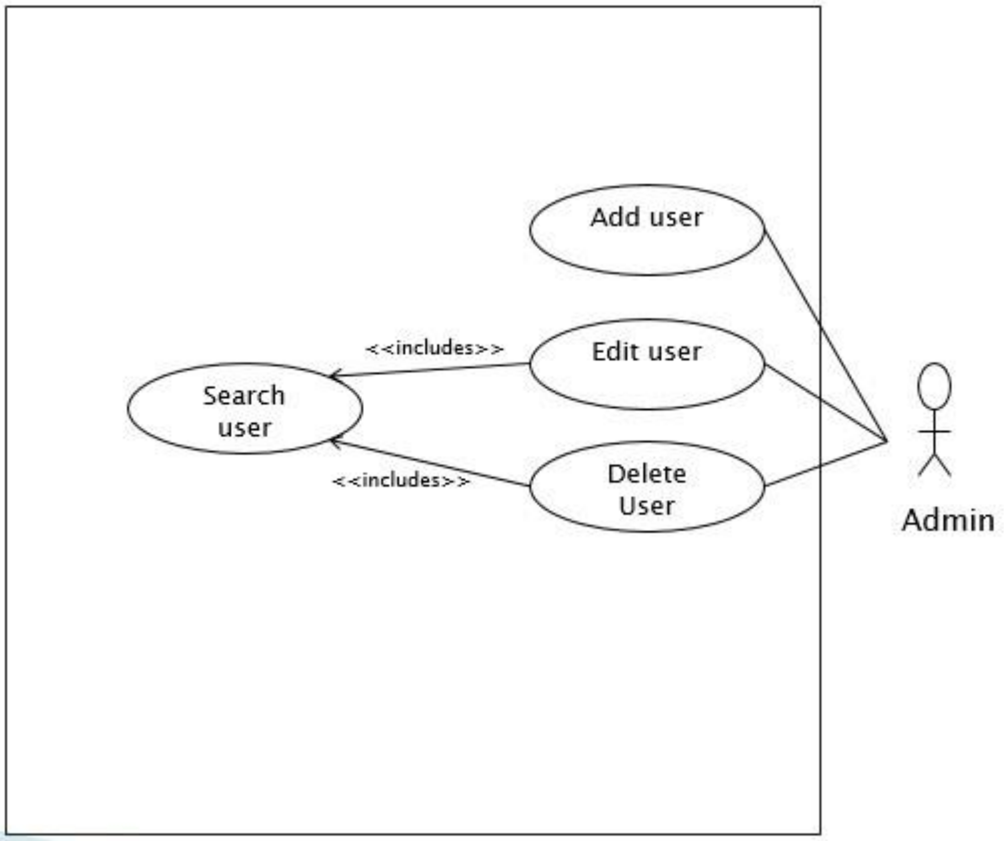

Add Flight
Reserve Ticket
Edit Flight
Search Flight
Cancel Flight
Admin
<<includes>>
<<includes>>
<<Includes>>
<<Includes>>
Send Email
Send Group email
<<Extends>>


Add user
Admin


Add user
Edit user
Admin

-
-
-


Add user
Edit user
Delete User
Admin



Use Case Name: SearchSeat

Priority: Normal

Actors: Passengers

Summary: This use case enables passenger to search flight as per his/her convenience

Precondition: flights exist in database

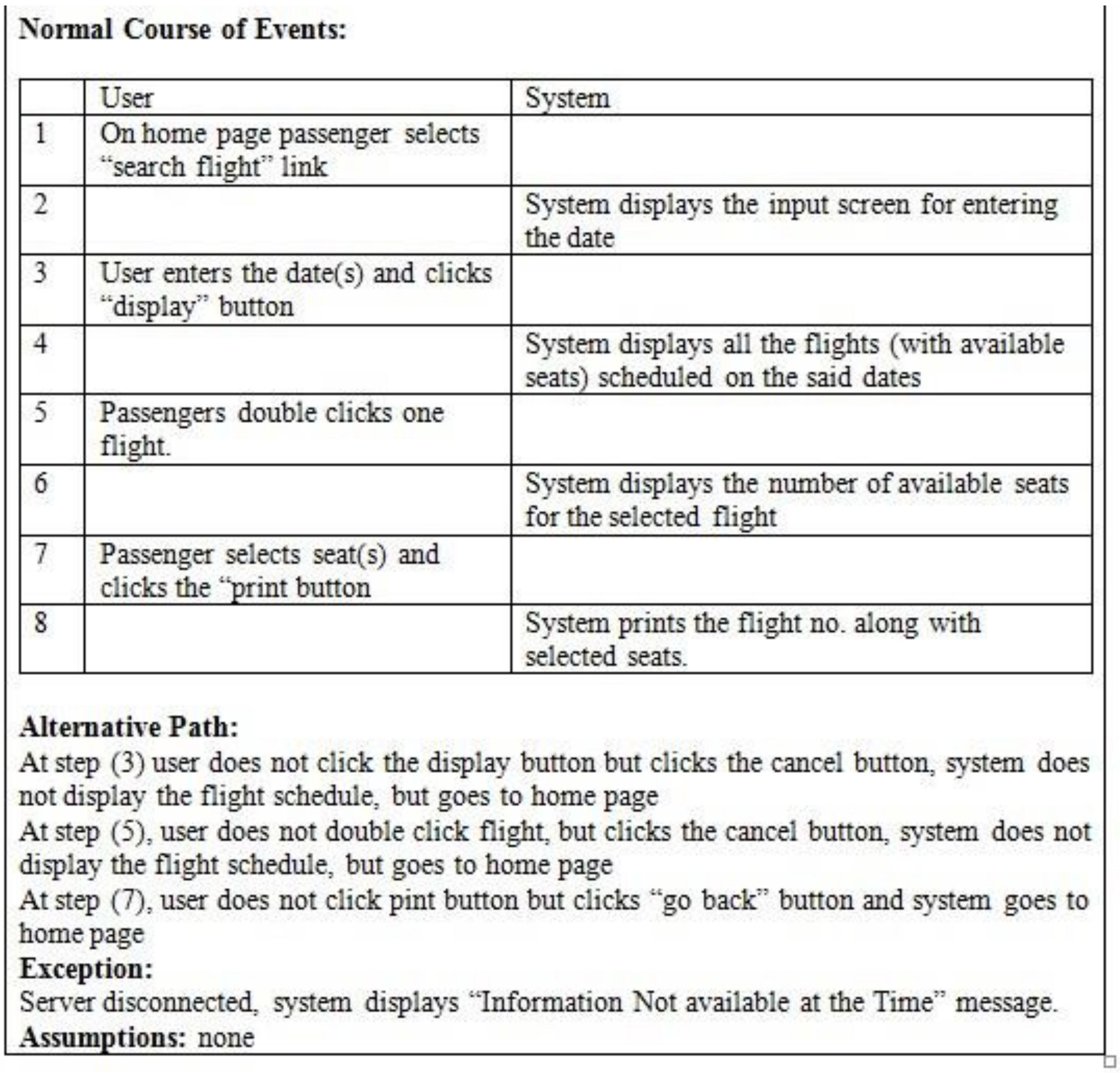
Post-Condition: Flight schedule is displayed for the said date(s)

Extends: none

Uses: SearchFlight

Normal Course of Events:

-
-
-



Normal Course of Events:

	User	System
1	On home page passenger selects "search flight" link	
2		System displays the input screen for entering the date
3	User enters the date(s) and clicks "display" button	
4		System displays all the flights (with available seats) scheduled on the said dates
5	Passengers double clicks one flight.	
6		System displays the number of available seats for the selected flight
7	Passenger selects seat(s) and clicks the "print button	
8		System prints the flight no. along with selected seats.

Alternative Path:
At step (3) user does not click the display button but clicks the cancel button, system does not display the flight schedule, but goes to home page
At step (5), user does not double click flight, but clicks the cancel button, system does not display the flight schedule, but goes to home page
At step (7), user does not click pint button but clicks "go back" button and system goes to home page
Exception:
Server disconnected, system displays "Information Not available at the Time" message.
Assumptions: none

Summary

- Case study of use case

LECTURE # 7 Slide # 2 Use case modeling (Part 2)

Steps in usecase modeling

- Step-1: Identify business actors.
    - Who or what provides inputs to the system?
    - Who or what receives outputs from the system?

- Are interfaces required to other systems?
- Who will maintain information in the system?

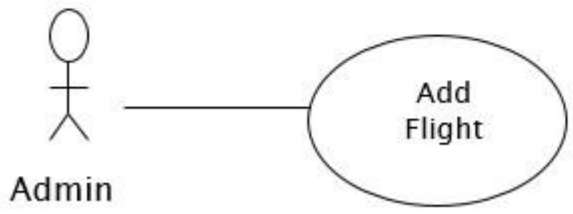
❖ Actors should be named with a noun or noun phrase

▸ <u>Step-2: Identify business use cases.</u>

- What are the main tasks of the actor?
- What information does the actor need from the system?
- What information does the actor provide to the system?

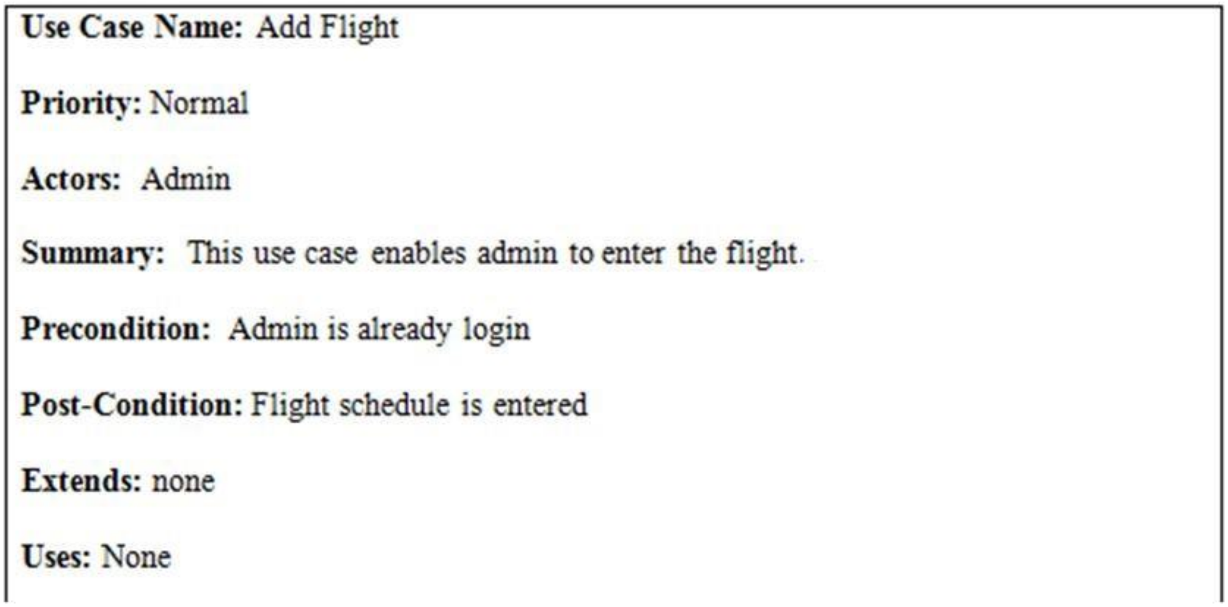
❖ Use cases should be named with a verb phrase specifying the goal of the actor (e.g. PlaceOrder)

▸ <u>Step-3: Construct use-case model diagram.</u>



▸ <u>Step-4: Documents business requirements use-case narratives.</u>

-
-
-



Use Case Name: Add Flight

Priority: Normal

Actors: Admin

Summary: This use case enables admin to enter the flight.

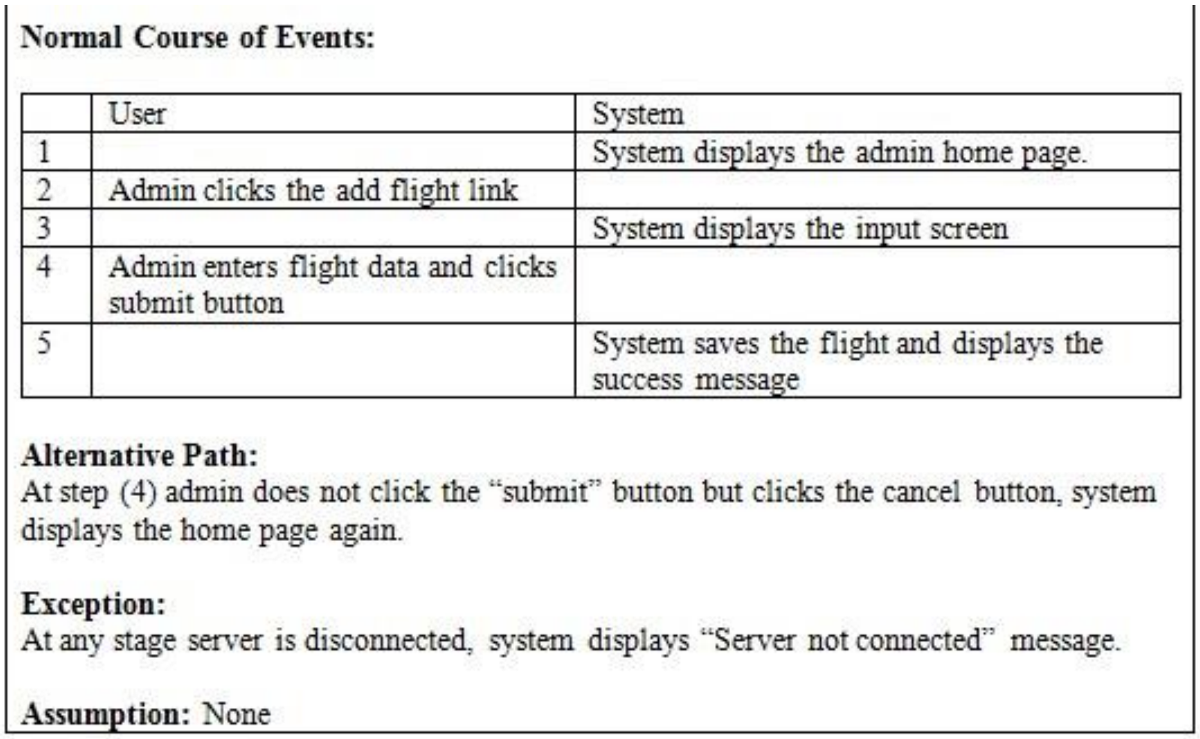
Precondition: Admin is already login

Post-Condition: Flight schedule is entered

Extends: none

Uses: None

Steps in usecase modeling



Normal Course of Events:

	User	System
1		System displays the admin home page.
2	Admin clicks the add flight link	
3		System displays the input screen
4	Admin enters flight data and clicks submit button	
5		System saves the flight and displays the success message

Alternative Path:
At step (4) admin does not click the "submit" button but clicks the cancel button, system displays the home page again.

Exception:
At any stage server is disconnected, system displays "Server not connected" message.

Assumption: None

Summary

✓ Steps of use case modeling

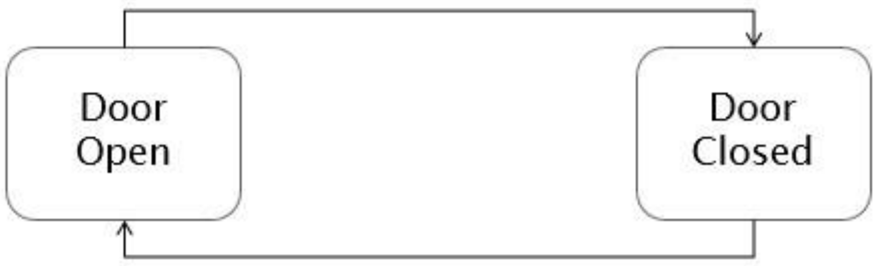
- ✓ Documenting usecase narrative

LECTURE # 8 slide # 1 State machines

Statemachine modeling

- Many information systems deal with business objects that involve a series of possible states.
- Describing a set of complex state changes in natural language creates a high probability of overlooking a permitted state change or including a disallowed change.
- State machines provide a concise, complete, and unambiguous representation of the states of an object or system.
- It relates events and objects.

State machine example



LECTURE # 8 slide # 2

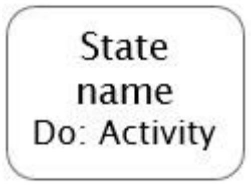
State machines

Muhammad Summair Raza

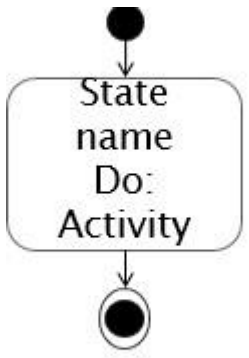
State machine modeling

- States - A state is denoted by a round-cornered rectangle with the name of the state written inside it.

-
-
-

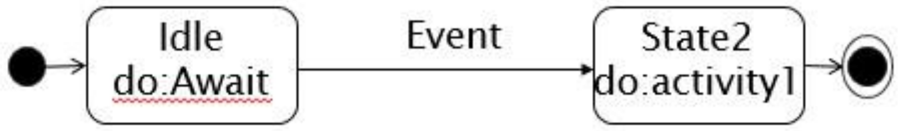


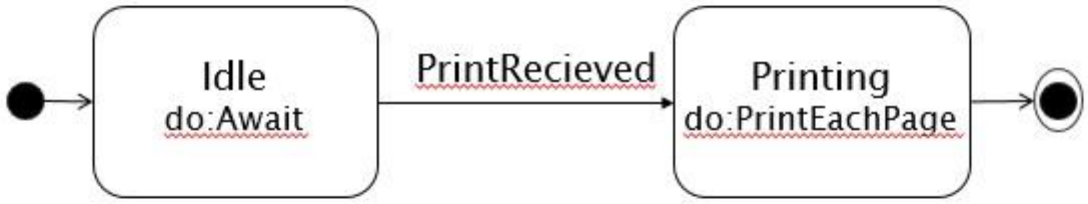
- Initial and Final States - The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.

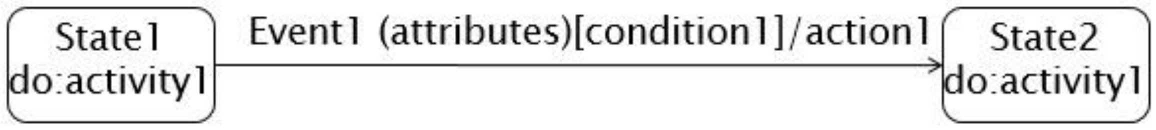


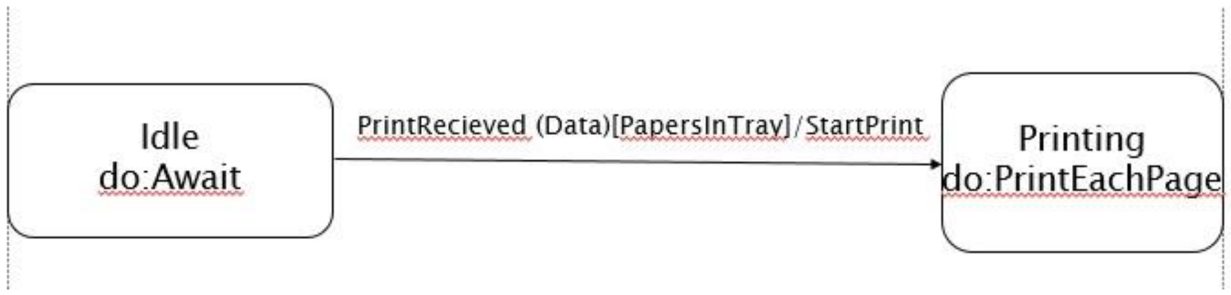
- Transition - Transition from one state to the next is denoted by lines with arrowheads.

- Trigger is the cause of the transition.
  - A signal.
  - An event.
  - A change in some condition.
  - The passage of time.







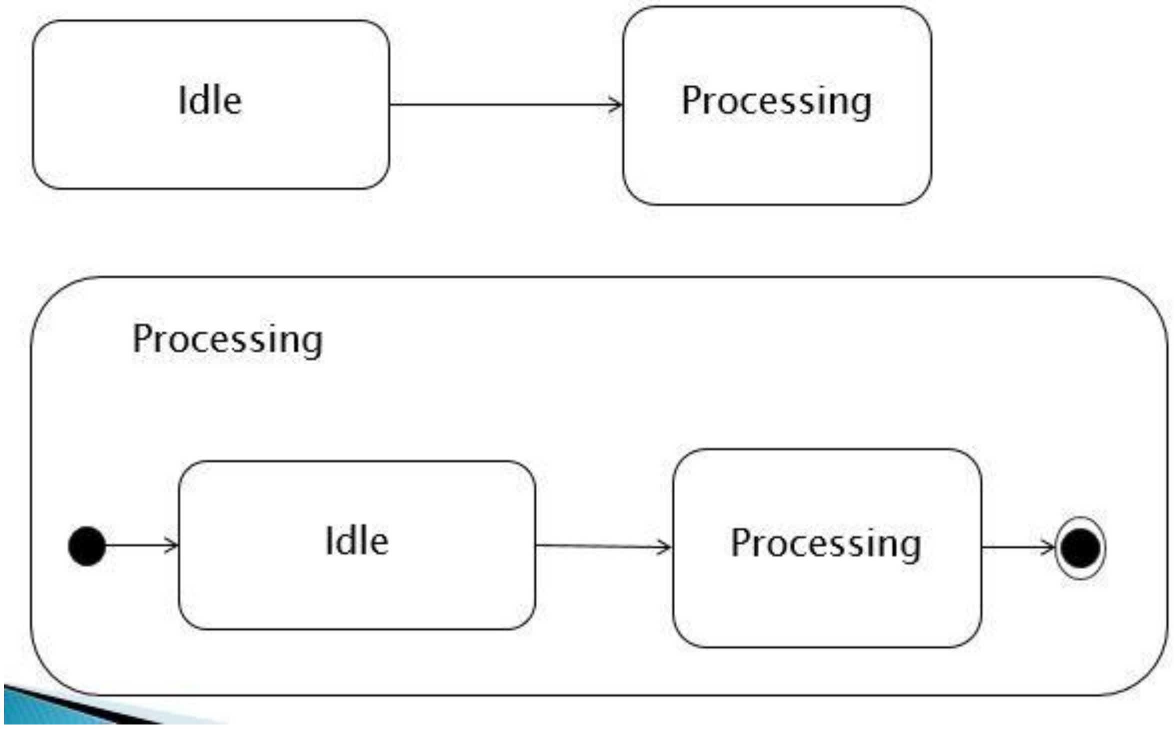


Nested State Diagram

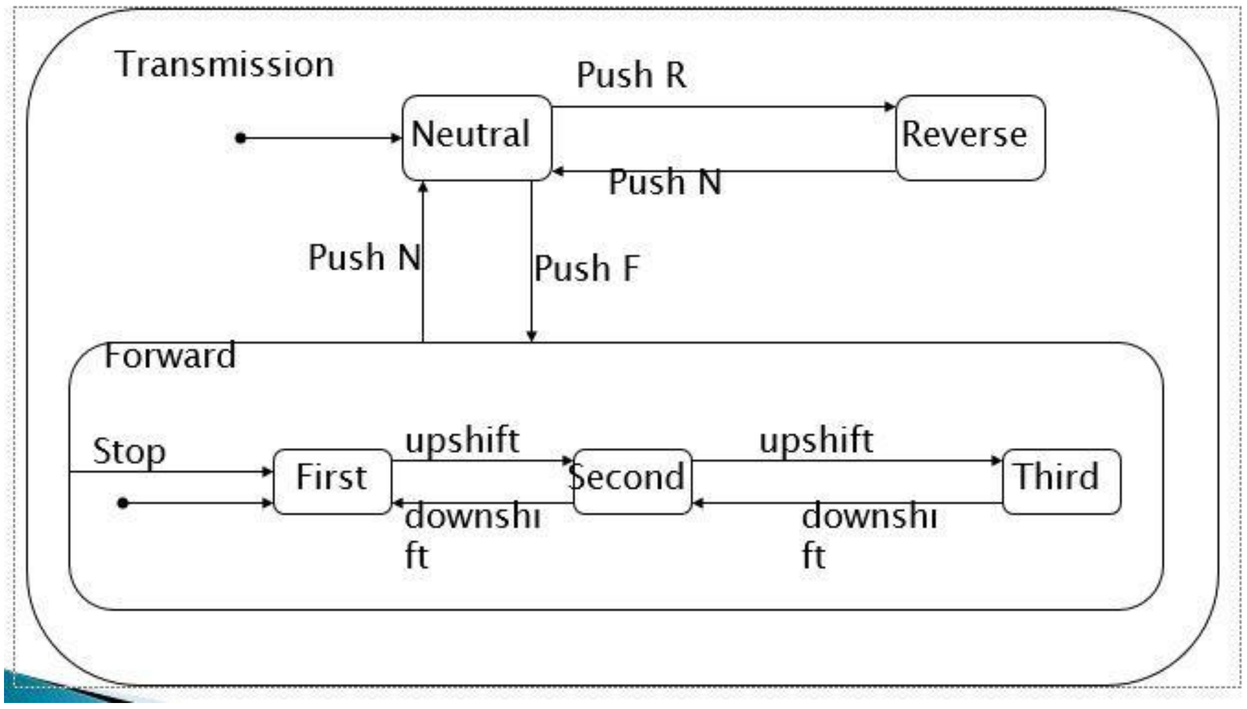
- State diagrams can be structured to permit concise descriptions of complex systems.

-
-
-
- It allows an activity to be described at a high level, then expanded at lower level by adding details.

<u>Nested State Diagram</u>



<u>Generalization of transmission state</u>

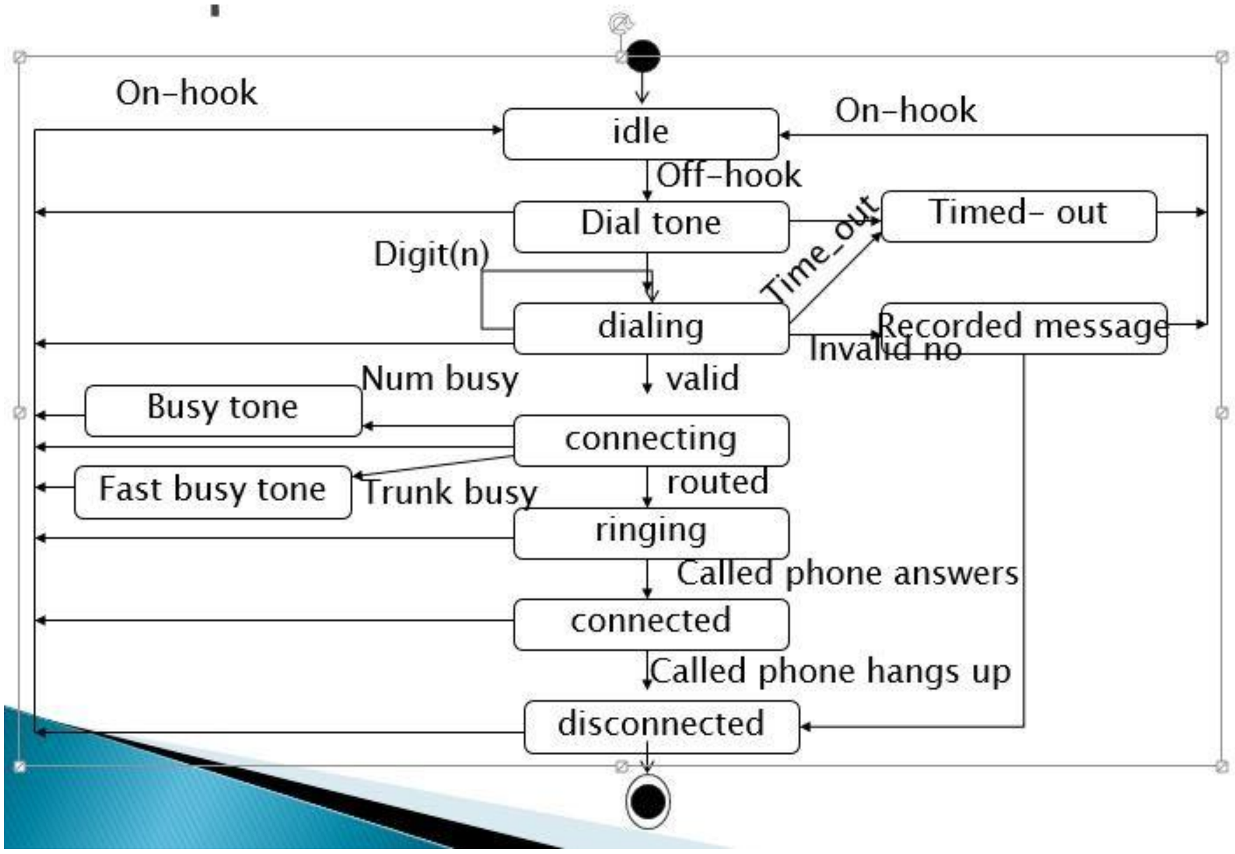


<u>Aggregation and concurrency</u>

- Aggregation means concurrency
- Overall state of aggregate object is combination of states of sub-objects
- Car
  - Ignition
  - Transmission
  - Accelerator
  - Brakes
- <u>Sample state of a car</u>
  - <u>Ignition: ON</u>
  - <u>Transmission: Neutral</u>
  - <u>Accelerator: On</u>
  - <u>Brakes: Off</u>

-
-
-
    - Sample state of a car
    - Ignition: ON
    - Transmission: Reverse
    - Accelerator: off
    - Brakes: On

Sample state machine



Lecture 9 (week # 9)

Goal Oriented RE

Traditional RE

- Traditional RE approaches start from the initial requirements statements (“What”).
- It ignores to focus on “Why” which is objective of GORE
- GORE is concerned with acquisition, modeling and analysis of stakeholder purposes (“goals”) in order to derive functional and non-functional requirements.

Goals:

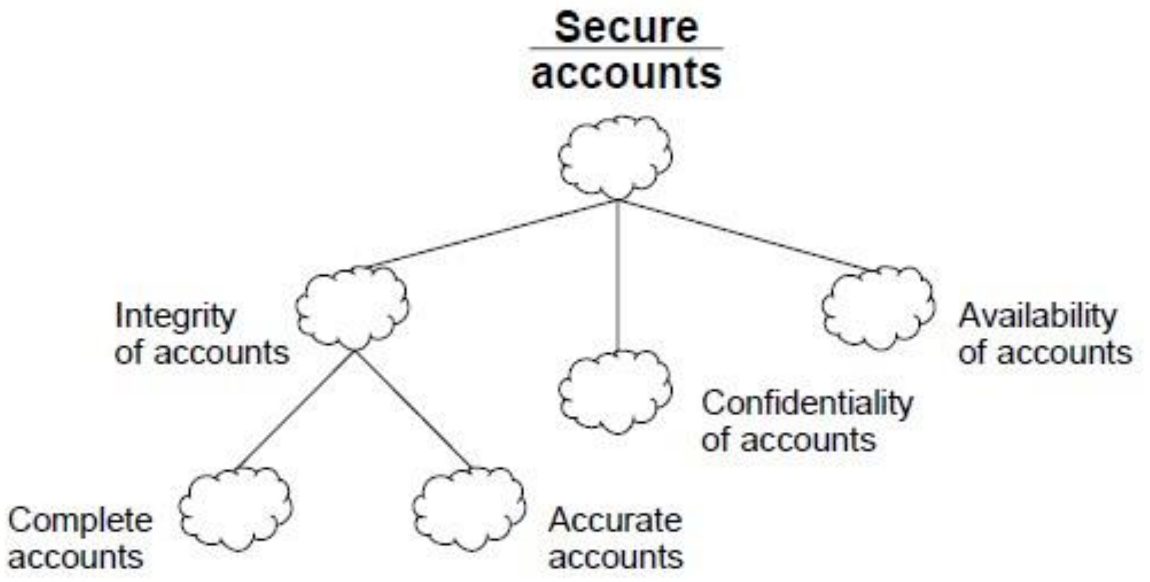
- A goal is an objective the system under consideration should achieve.
    - “Accounts should be secure”
    - Goals can be expressed at different level of abstraction:
    - High level goals
    - Sub-goals
- Goals cover different types of concerns: functional and non functional.

Why GORE?

- Requirements Elicitation
- Exploration of design choices
- Requirements completeness
- Requirements traceability
- Requirements negotiation

A simple goal model

-
-
-



Requirement vs. Goal

- A requirement is a particular way of achieving a goal.
- Goals are at a higher level than requirements
- Goals are more stable than corresponding requirements

Summary

- Limitations of traditional RE
- Why GORE?
- Requirements vs. Goals

NFR framework

Non Functional Requirement (NFR) Framework

- **Key concept is the notion of softgoal.**
- **Softgoals are goals that do not have a clear-cut criterion for their satisfaction; may be partially satisfied.**
- **This construct allows representing goals concerning NFRs of the system as well as ill-defined and high-level objectives of the stakeholders.**

Framework Activities

- **Capturing NFRs for the domain of interest.**

- **Decomposing NFRs**
- **Identifying possible NFR operationalizations (design alternatives for meeting NFRs)**
- **Dealing with ambiguities, tradeoffs, priorities, and interdependencies among NFRs**
- **Supporting decisions with design rationale.**
- **Evaluating impact of decisions.**

Softgoal Interdependency Graph

1. The main modeling tool that the framework provides is the softgoal interdependency graph (SIG).
2. The graphs can graphically represent softgoals, softgoal refinements (AND/OR), softgoal contributions (positive/negative), softgoal operationalization's and claims.

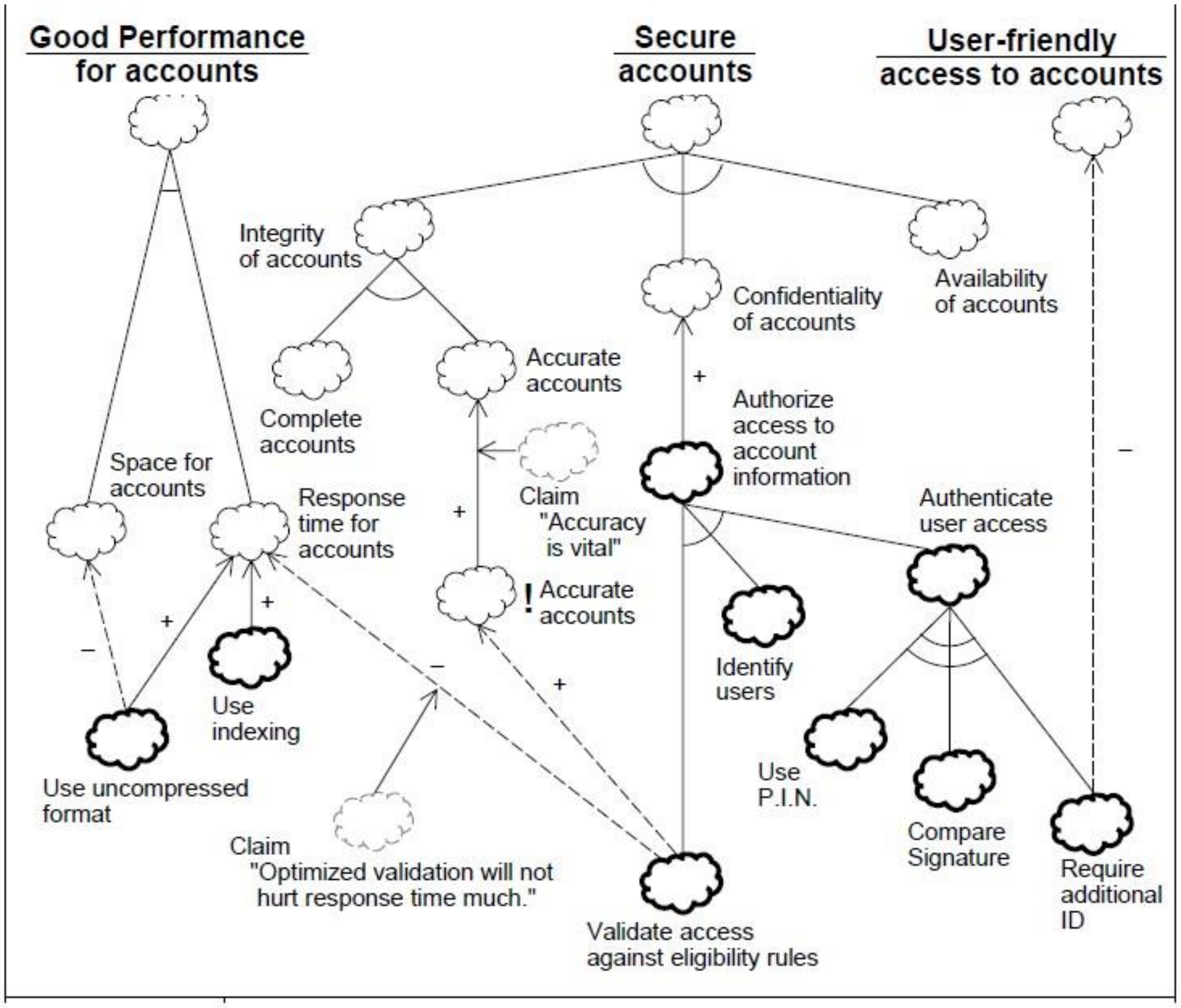
Types of Softgoals in NFR framework.

1. <u>NFR softgoals</u> represent non-functional requirements to be considered
2. <u>Operationalizing softgoals</u> model lower-level (design) techniques for *satisficing* NFR softgoals
3. <u>Claim softgoals</u> allow the analyst to record design rationale for softgoal refinements, softgoal prioritizations, softgoal contributions, etc.

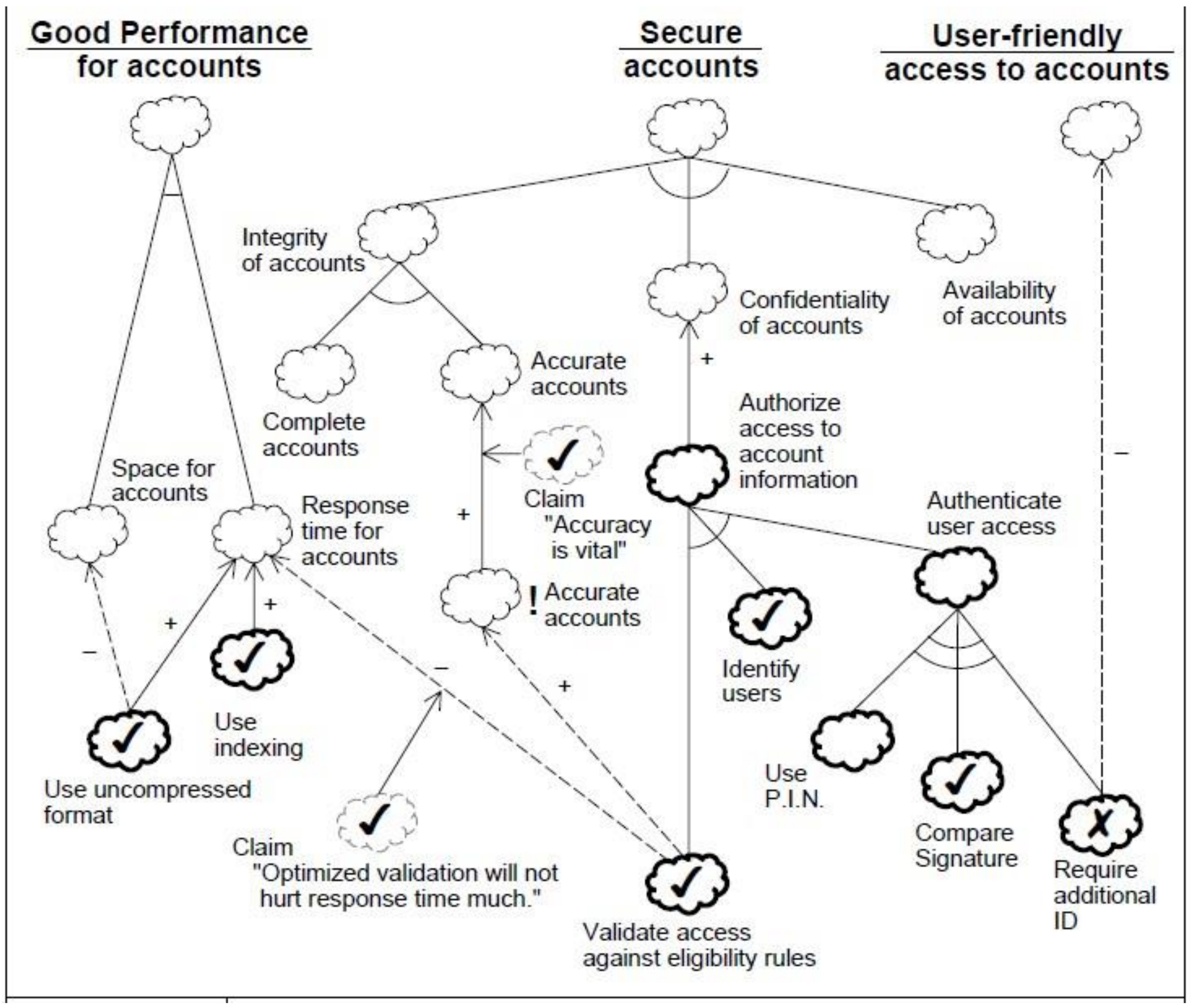
Softgoals

- Softgoals can be refined using AND or OR refinements with obvious semantics.
- Also, softgoal interdependencies can be captured with positive ("+") or negative ("–") contributions.

-
-
-



Good Performance for accounts
Secure accounts
User-friendly access to accounts
Integrity of accounts
Confidentiality of accounts
Availability of accounts
Complete accounts
Accurate accounts
Authorize access to account information
Space for accounts
Response time for accounts
Claim "Accuracy is vital"
Authenticate user access
! Accurate accounts
Identify users
Use indexing
Use uncompressed format
Use P.I.N.
Compare Signature
Require additional ID
Claim "Optimized validation will not hurt response time much."
Validate access against eligibility rules
+
–


Good Performance for accounts
Secure accounts
User-friendly access to accounts
Integrity of accounts
Confidentiality of accounts
Availability of accounts
Complete accounts
Accurate accounts
Authorize access to account information
Space for accounts
Response time for accounts
Claim "Accuracy is vital"
! Accurate accounts
Authenticate user access
Identify users
Use indexing
Use uncompressed format
Use P.I.N.
Compare Signature
Require additional ID
Claim "Optimized validation will not hurt response time much."
Validate access against eligibility rules

Summary

- NFR framework
- Framework activities
- Softgoal Interdependency Graph

Lecture week 10

KAOS (Knowledge Acquisition in Automated Specification)

- Methodology for requirements engineering enabling analysts to
    - build requirements models
    - derive requirements documents from KAOS models.

KAOS Ontology

- Objects are things of interest in the composite system whose instances may evolve from state to state. Objects can be entities, relationships, or events.
- Operations are input-output relations over objects. Operation applications define state transitions.
- An Agent is a kind of object that acts as a processor for operations. Agents are active components that can be humans, devices, software, etc.
- A Goal in KAOS is prescriptive statement of intent about some system whose satisfaction in general requires the cooperation of some of the agents forming that system.
- Goals may refer to services (functional goals) or to quality of services (non-functional goals).
- In KAOS, goals are organized in the usual AND/OR refinement abstraction hierarchies.
- Goal refinement ends when every subgoal is realizable by some individual agent assigned to it. That means the goal must be expressible in terms of conditions that are monitorable and controllable by the agent.

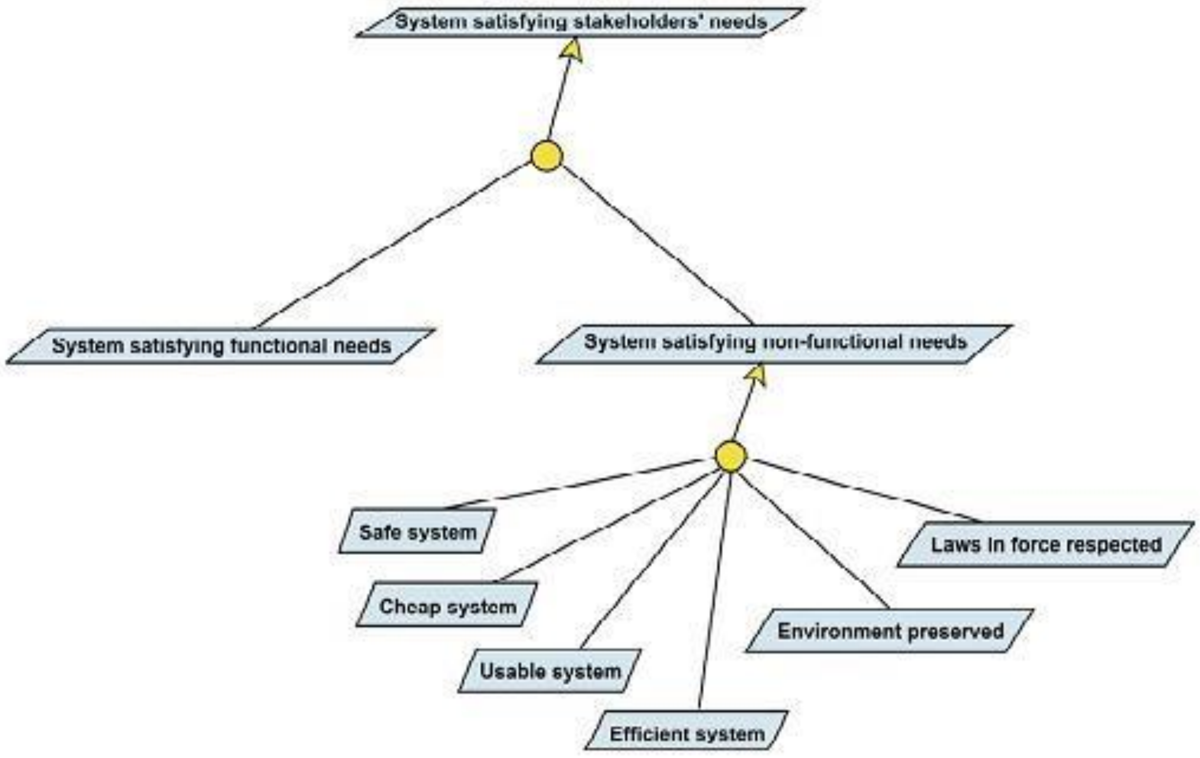
KAOS Models

1. Goal model where goals are represented, and assigned to agents
2. Object model which is a UML model that can be derived from formal specifications of goals since they refer to objects or their properties
3. Operation model which defines various services to be provided by software agents.
4. Responsibility models for various agents

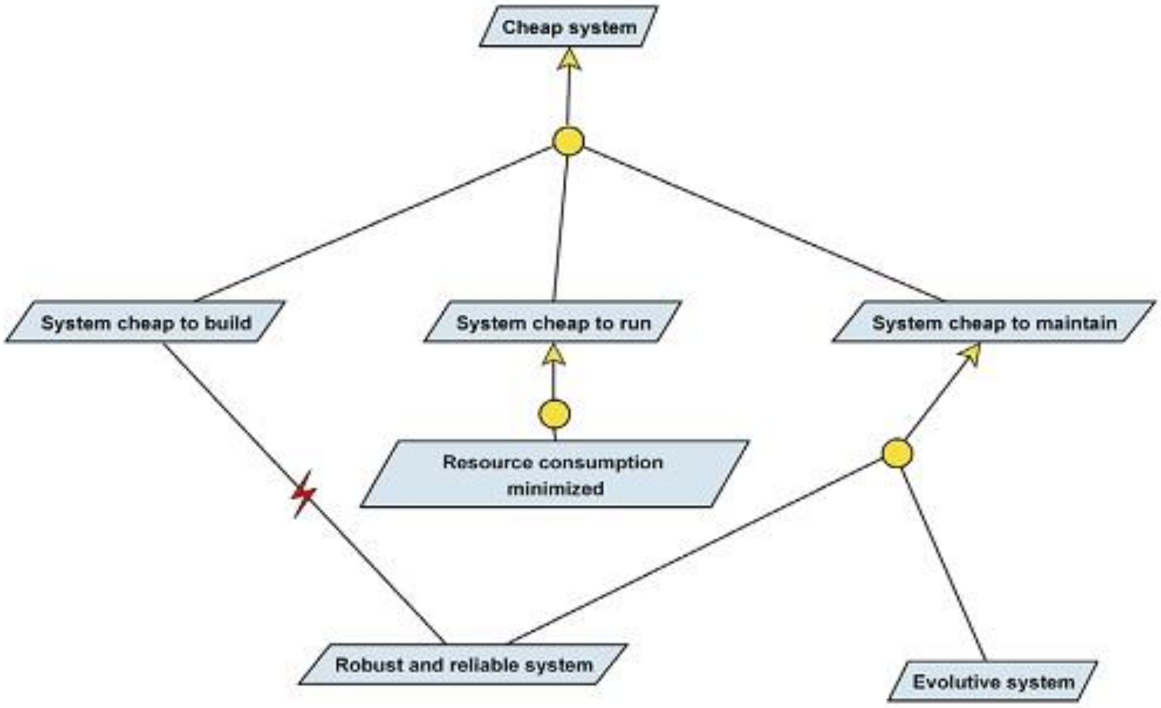
Building the Goal Model

- Good Practices
    - Requirements Patterns, grown over time as organizational experience with GORE matures
    - Milestone driven refinement of goals as a guide for completeness

Example: Generic Goal Pattern


System satisfying stakeholders' needs
System satisfying functional needs
System satisfying non-functional needs
Safe system
Cheap system
Usable system
Efficient system
Environment preserved
Laws in force respected

Example: Generic Conflicting Goals

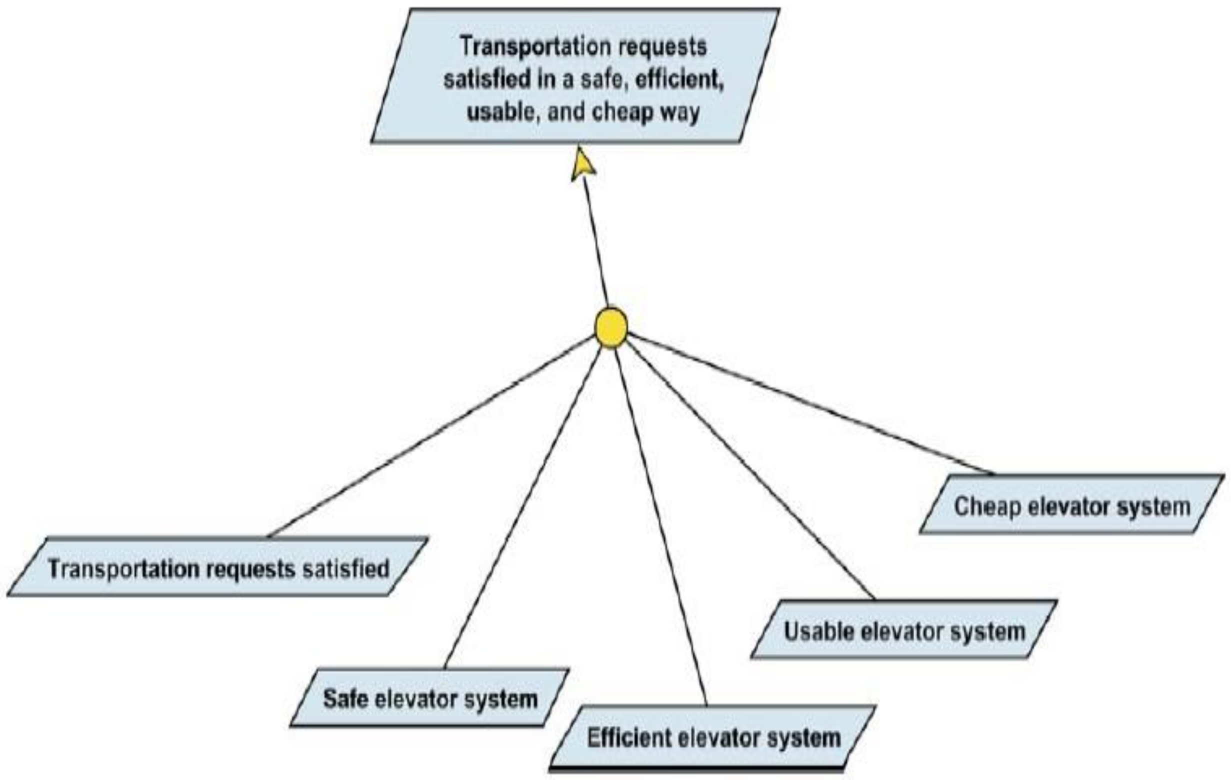


The Elevator Case Study

- Problem Statement:

You’ve just been hired by an elevator design company to improve performance and quality of software development within the company. You’ve directly pointed out a major weakness in the way software is developed : there is currently no formal requirements engineering method in use. As a first challenge, you are asked to build a KAOS model for a new elevator system to be designed.

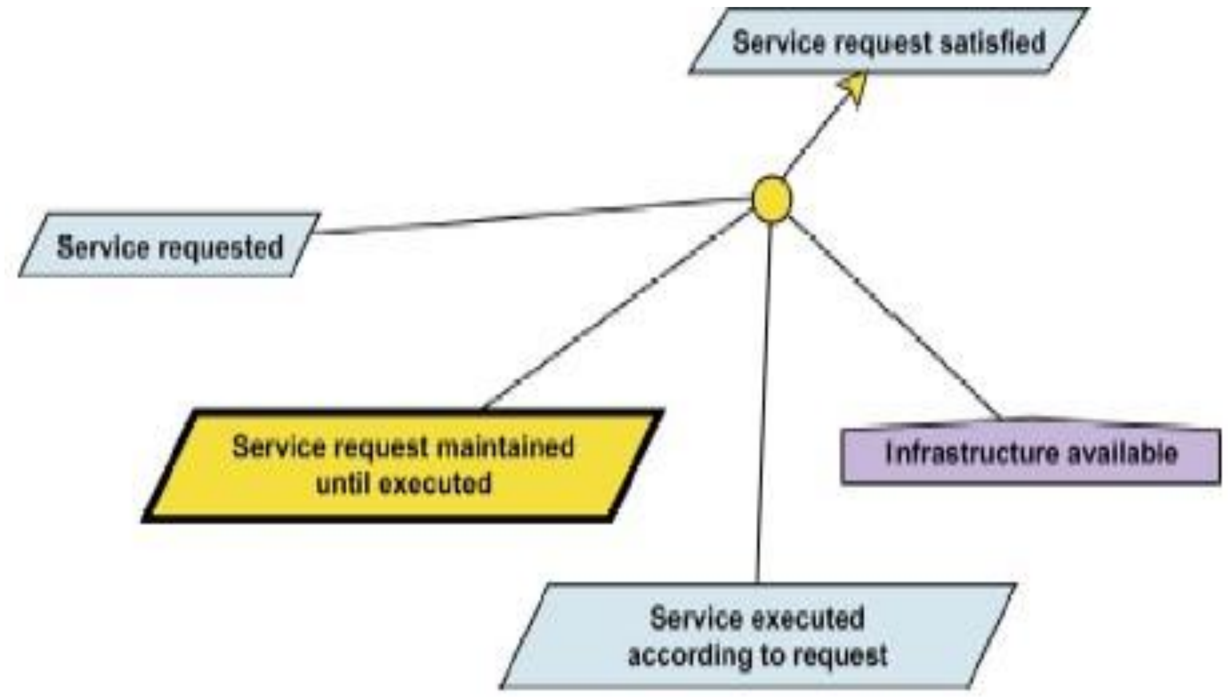
Instantiating the Generic Goal Model


Transportation requests satisfied in a safe, efficient, usable, and cheap way
Transportation requests satisfied
Safe elevator system
Efficient elevator system
Usable elevator system
Cheap elevator system

Goals covered in case study

1. “Transportation requests satisfied” (i.e. functional need)
2. “Safe elevator system” (i.e. non-functional need)

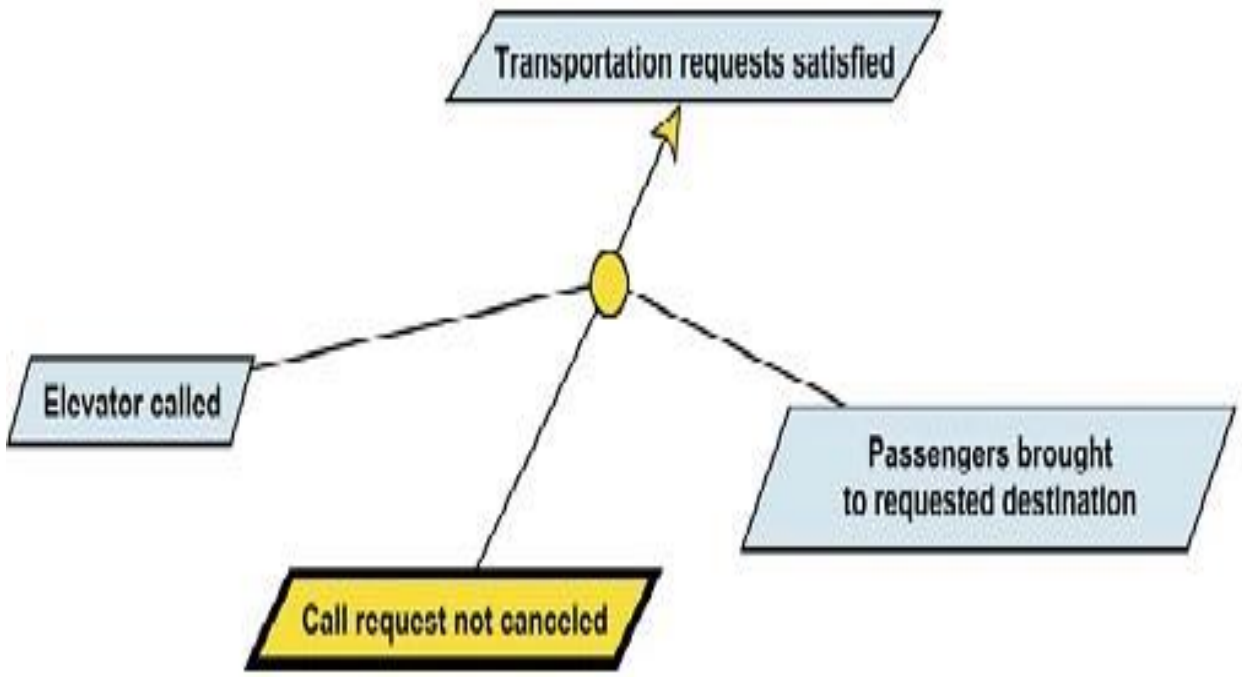
Transportation Request Satisfied: Generic Pattern for Service Requests Satisfaction


Service request satisfied
Service requested
Service request maintained until executed
Infrastructure available
Service executed according to request

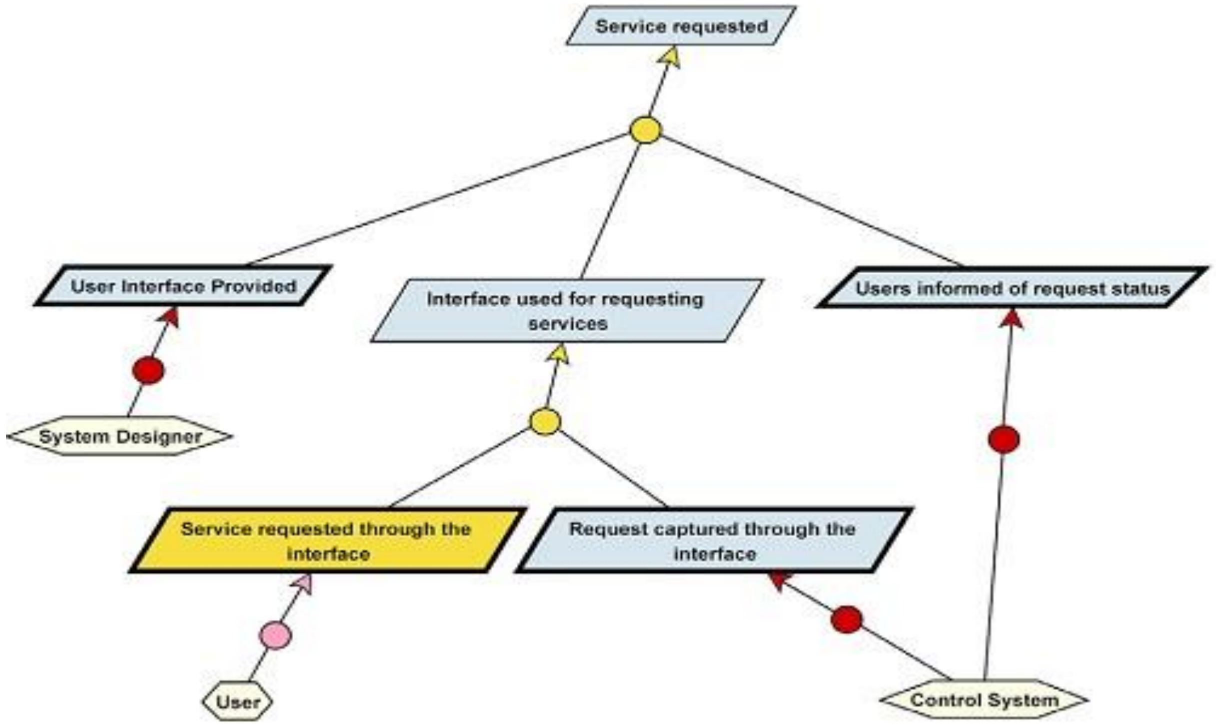
A Note on Terminology

- Look at the previous figure and notice the way goals have been named: a word followed by verb in its passive form. For instance, we have written "Service requested" instead of "Request service" or "The passenger must request the service".
- The reason is to avoid confusion between goals and operations (agent behaviors). Goals basically refer to system states we want to achieve or maintain, cease or avoid. They do not refer to system state transitions.

Instantiating the Service Request Satisfaction Pattern


Transportation requests satisfied
Elevator called
Passengers brought to requested destination
Call request not canceled

Elevator Called: Generic Service Request Pattern

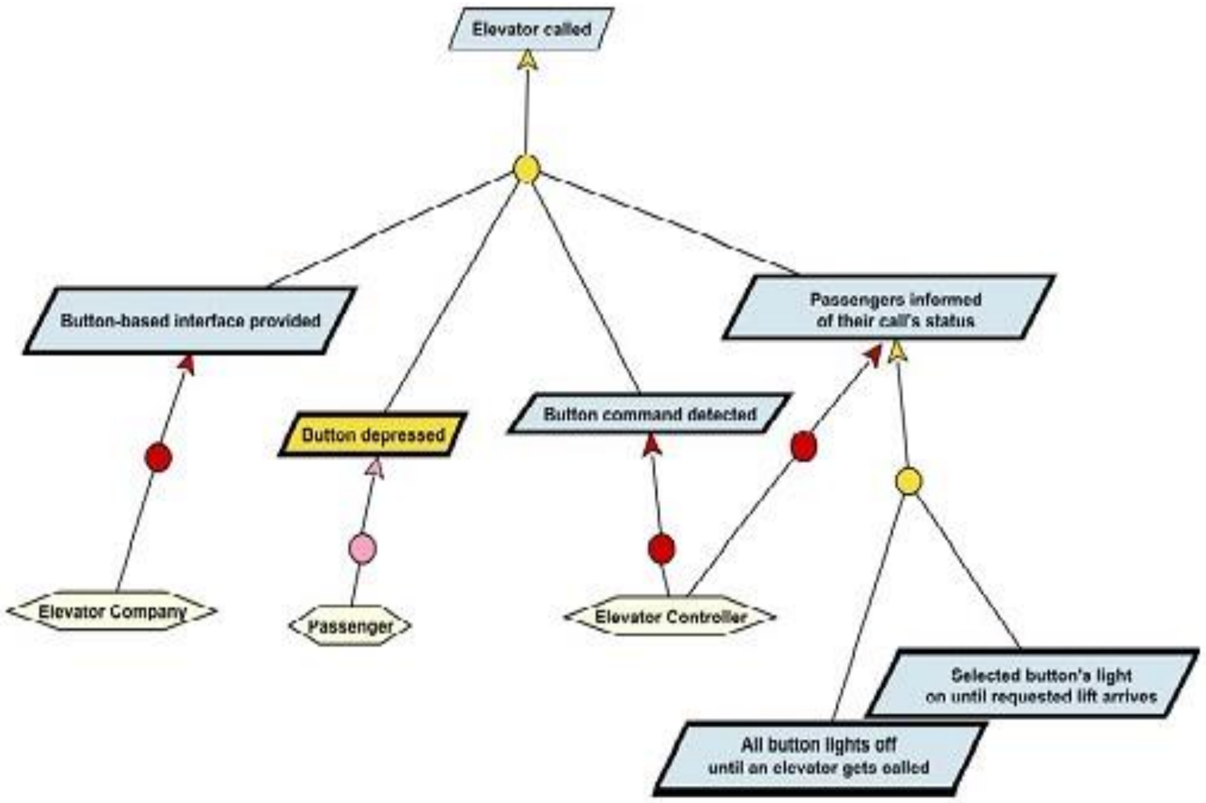

Service requested
User Interface Provided
Interface used for requesting services
Users informed of request status
System Designer
Service requested through the interface
Request captured through the interface
User
Control System

Goal Model Completeness Criteria

- Completeness criterion 1: A goal model is said to be complete with respect to the refinement relationship 'if and only if' every leaf goal is either an expectation, a domain property or a requirement.

- Completeness criterion 2: A goal model is complete with respect to the responsibility relationship ‘if and only if’ every requirement is placed under the responsibility of one and only one agent (either explicitly or implicitly if the requirement refines another one which has been placed under the responsibility of some agent).

Elevator Called: Instantiating the Pattern


Elevator called
Button-based interface provided
Passengers informed of their call's status
Button depressed
Button command detected
Elevator Company
Passenger
Elevator Controller
Selected button's light on until requested lift arrives
All button lights off until an elevator gets called

Summary

- KAOS Goal Model
- Terminologies
- Completeness criteria

KAOS Object Model

- Used to define and document concepts of the application domain that are relevant with respect to the known requirements and provide static constraints on operational system.
- Objects pertaining to the stakeholders’ domain
- Other objects introduced on purpose to express requirements or constraints on the operational system.

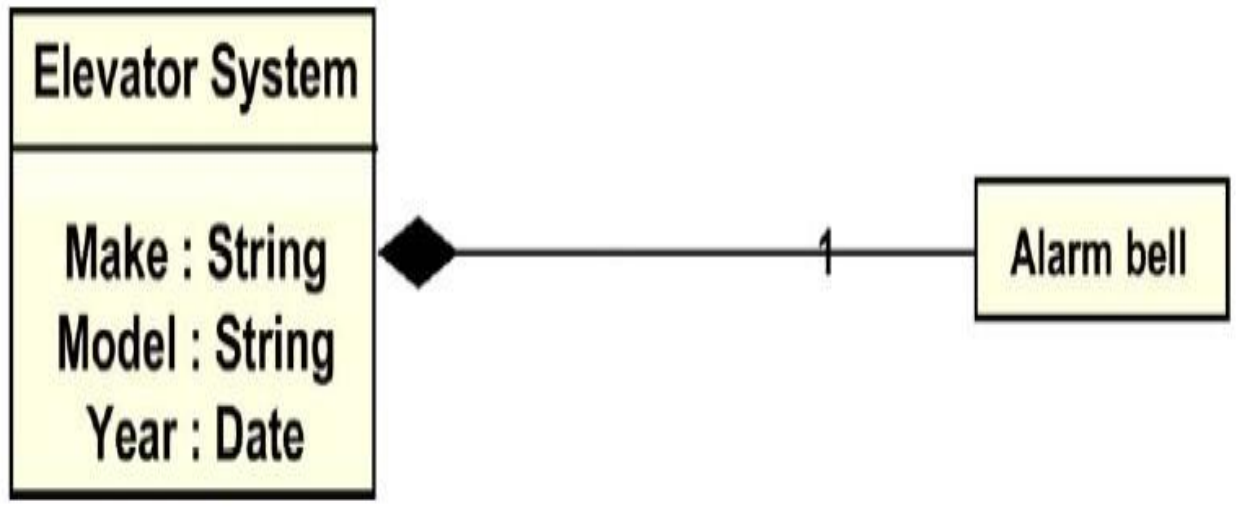
Types of Objects

- Entities:

- Represent independent, passive objects.
- For instance, elevator doors, buttons, etc...
- ‘Independent’ means that their descriptions needn’t refer to other objects of the model.
- They are ‘passive’ means they can’t perform operations.

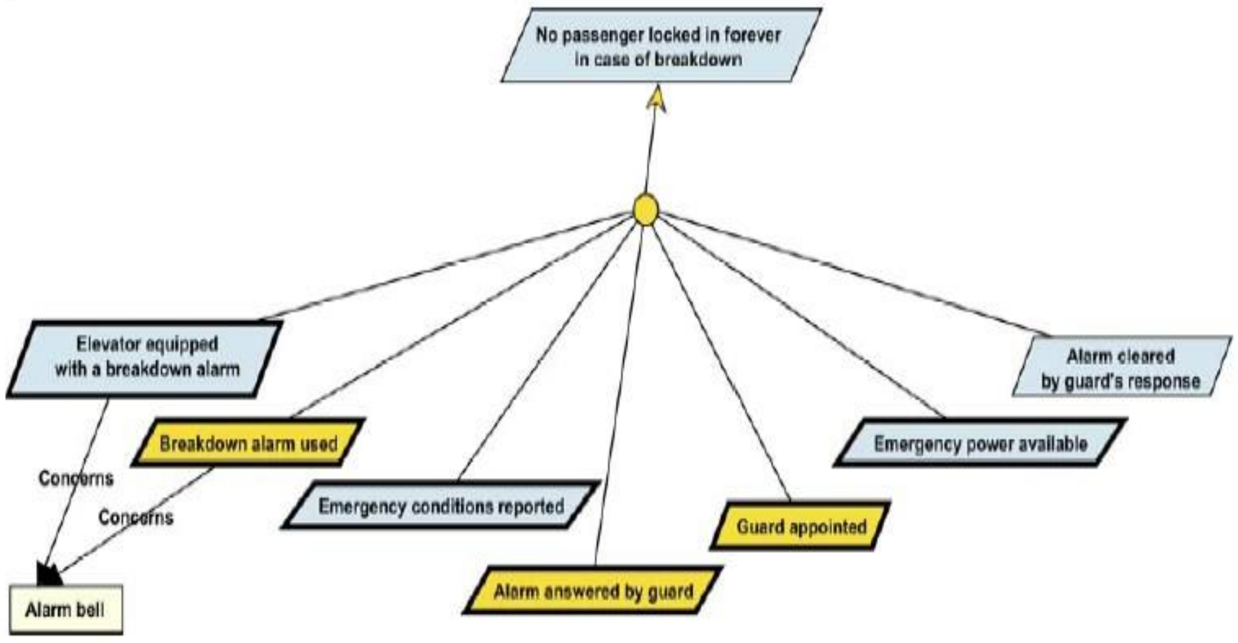
- Agents:
  - represent independent, active objects.
  - For instance, elevator company, passenger, elevator controller etc.
  - They are active meaning they can perform operations.
  - Operations usually imply state transitions on entities (for instance, the “RingAlarm” operation implies the following state transition on the entity “Alarm”: status attribute changed from “Silent” to “Ringing”).
- Associations:
  - dependent, passive objects.
  - ‘Dependent’ because their descriptions refer to other objects.
  - For instance, the “At” association links a Cage to a Floor. An instance of that association (say between Cage ‘c’ and Floor ‘f’) would hold if cage ‘c’ is currently located on floor ‘f’.
  - They are passive so they can’t perform operations. **Object**

Model: Example

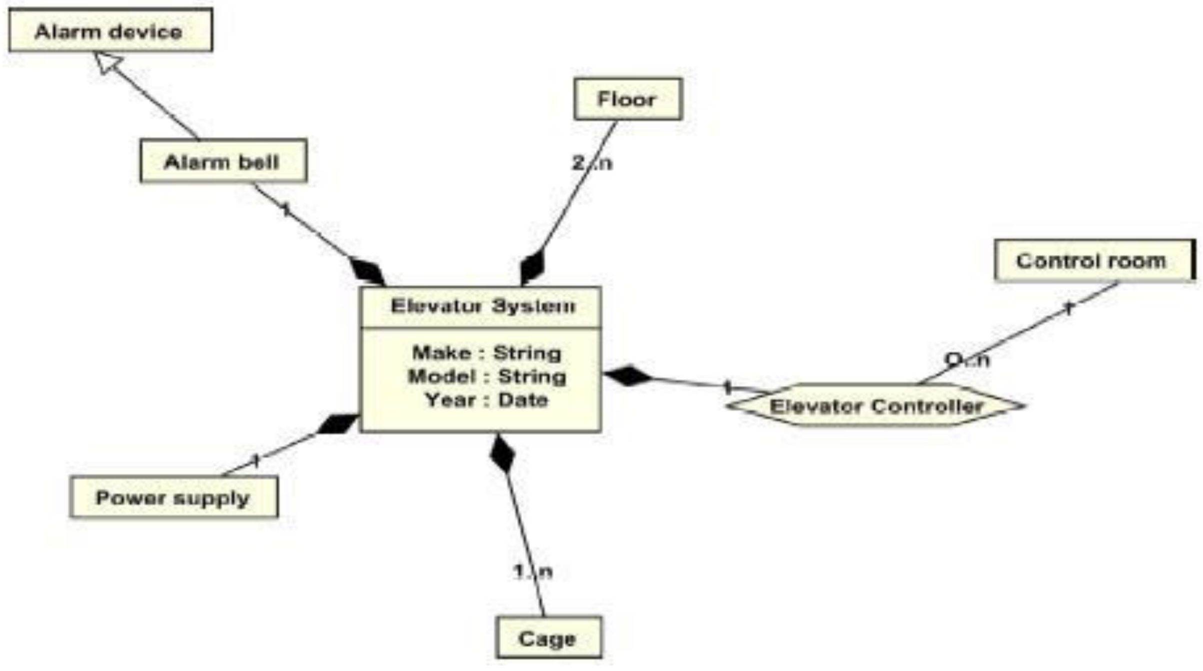


“Concerns” Relationship for Identifying Objects in Goal Model

-


No passenger locked in forever in case of breakdown
Elevator equipped with a breakdown alarm
Breakdown alarm used
Emergency conditions reported
Alarm answered by guard
Guard appointed
Emergency power available
Alarm cleared by guard's response
Concerns
Concerns
Alarm bell

Example: Elevator System



Summary

- KAOS Object Model
- Types of objects
- Example

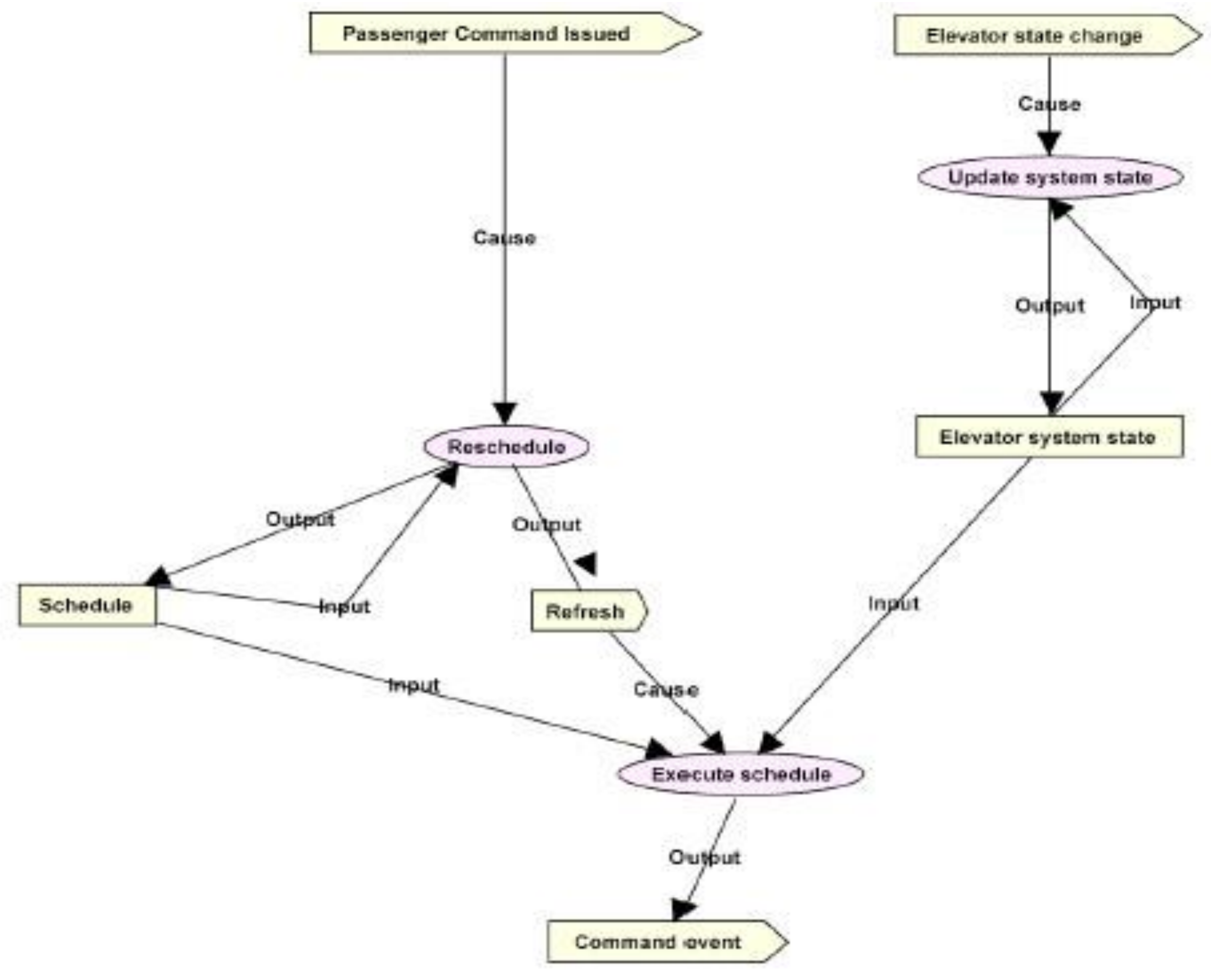
LECTURE # 11

Week # 11

KAOS Operational Model

- **The KAOS operation model describes all the behaviors that agents need to fulfill their requirements.**
- **Behaviors are expressed in terms of operations performed by agents.**
- **Operations work on objects (defined in the object model): they can create objects, trigger object state transitions and activate other operations (by sending an event).**
- **Operations can directly be expressed by stakeholders during the interviews.**
- **Operations can be identified by looking at all the existing requirements.**

Operational Model



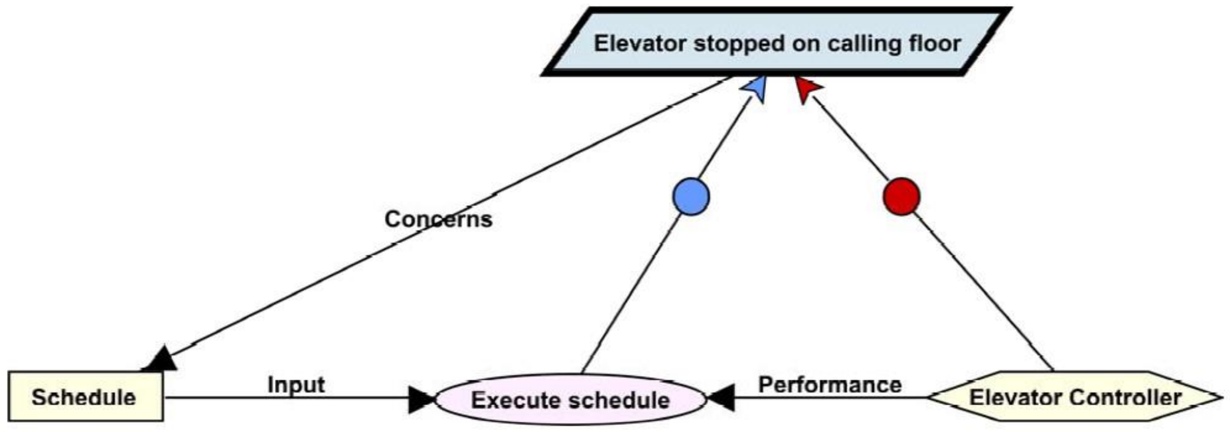
Operational Model

- Operations are represented as ovals.
- Concerned objects are connected to the operations by means of Input and Output links.
- Events are represented as those traffic signs that are used to indicate directions.

Completeness criteria

- To be complete, a process diagram must specify:
    - The agents who perform the operations

- **The input and output data for each operation.**
- **when operations are to be executed.**
- **All operations are to be justified by the existence of some requirements (through the use of operationalization links).**



Summary

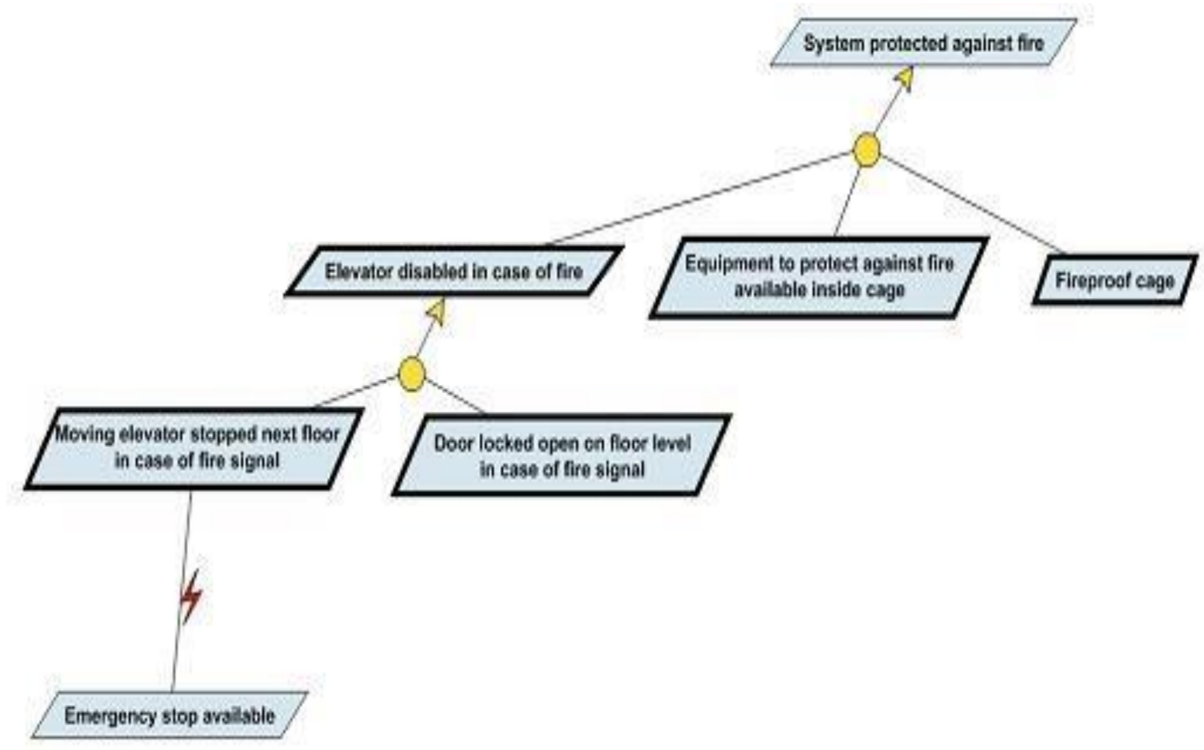
- **KAOS Operational model**
- **Operational model terminology**
- **Example**

KAOS Responsibility Model

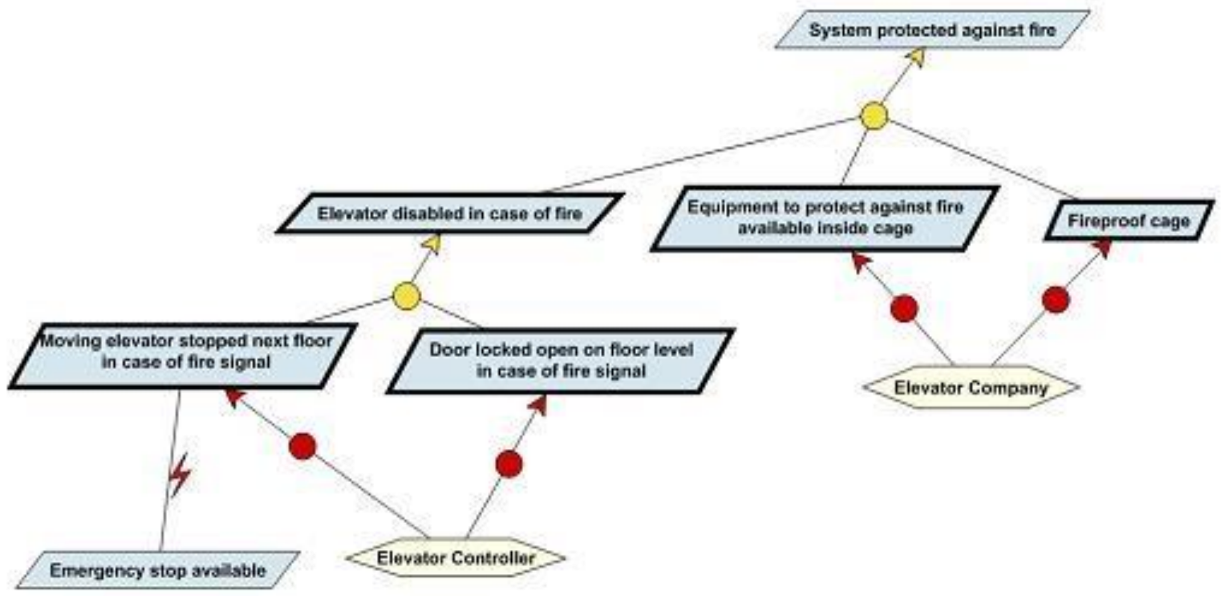
- **The responsibility model contains all the responsibility diagrams.**
- **A responsibility diagram describes for each agent, the requirements and expectations that he's responsible for, or that have been assigned to him.**
- **To build a responsibility diagram, the analyst reviews the different requirements and expectations in the goal model and assigns an agent to each of them.**

- **After all requirements and expectations are assigned a responsible agent, a diagram is generated for each agent, listing all requirements and expectations that he's been assigned.**

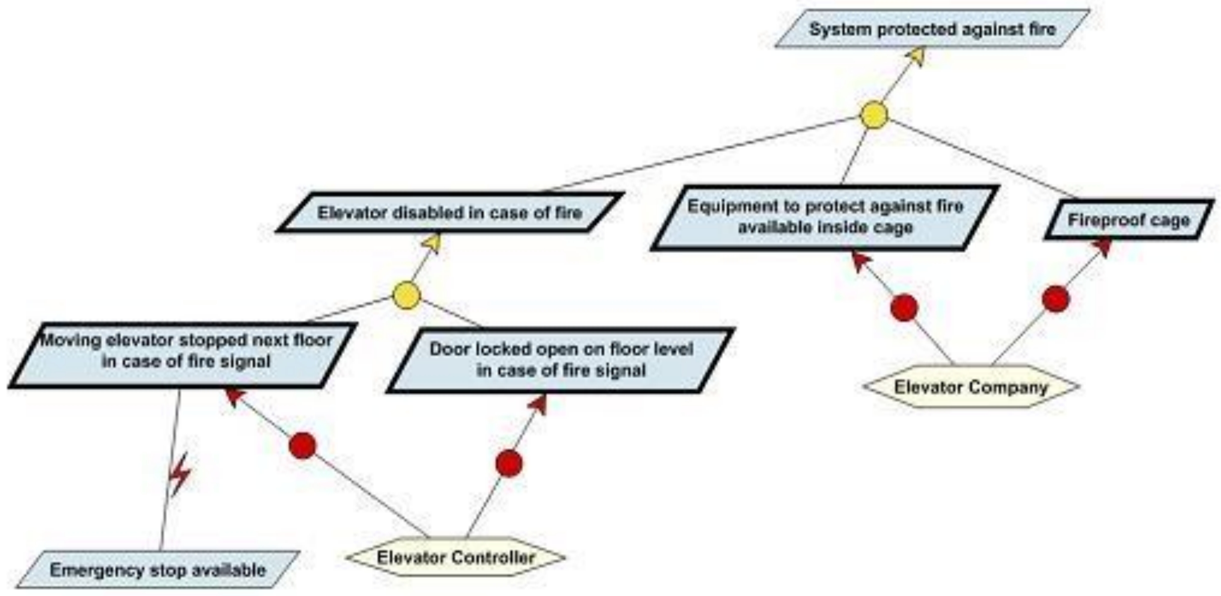
From Goal Diagram to Responsibility Diagram


System protected against fire
Elevator disabled in case of fire
Equipment to protect against fire available inside cage
Fireproof cage
Moving elevator stopped next floor in case of fire signal
Door locked open on floor level in case of fire signal
Emergency stop available

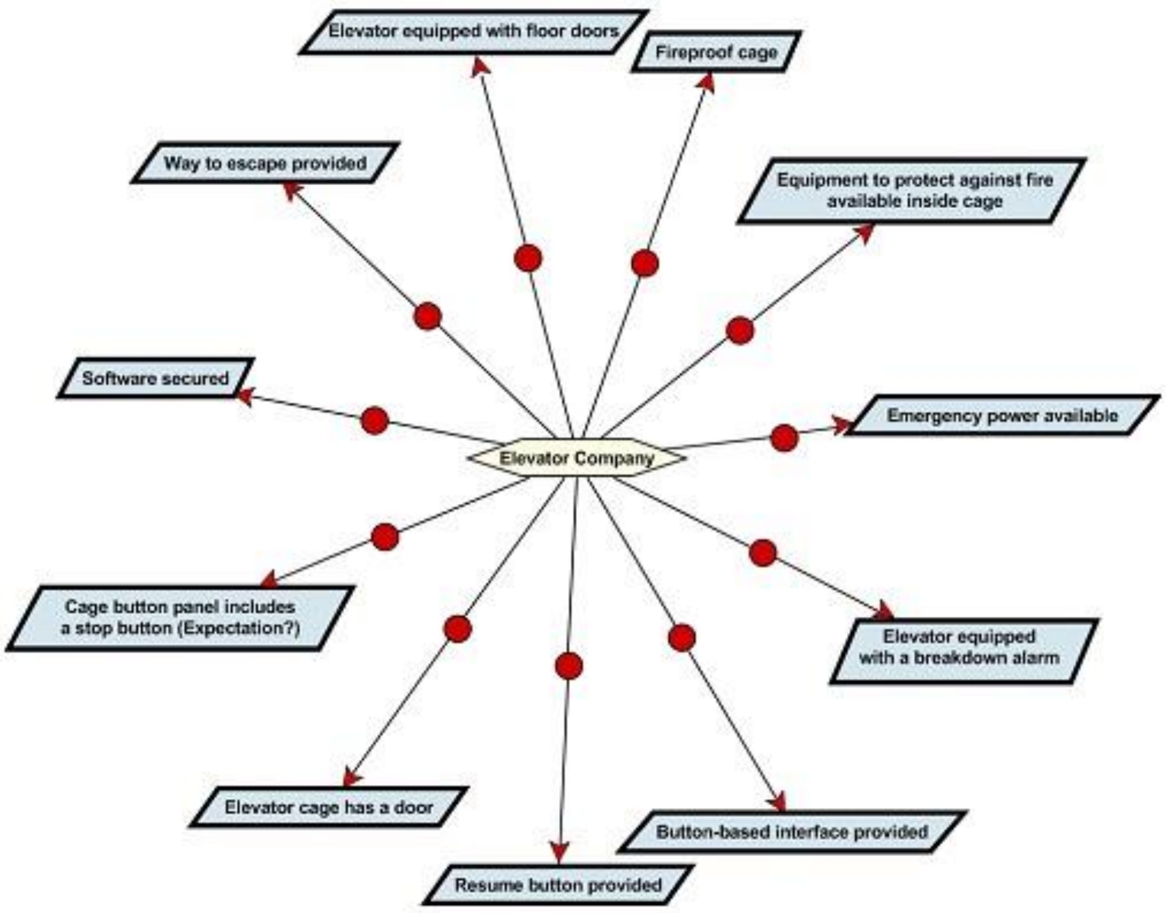
Intermediate Refinement: (Responsibilities Added for all Requirements)


System protected against fire
Elevator disabled in case of fire
Equipment to protect against fire available inside cage
Fireproof cage
Moving elevator stopped next floor in case of fire signal
Door locked open on floor level in case of fire signal
Elevator Company
Emergency stop available
Elevator Controller

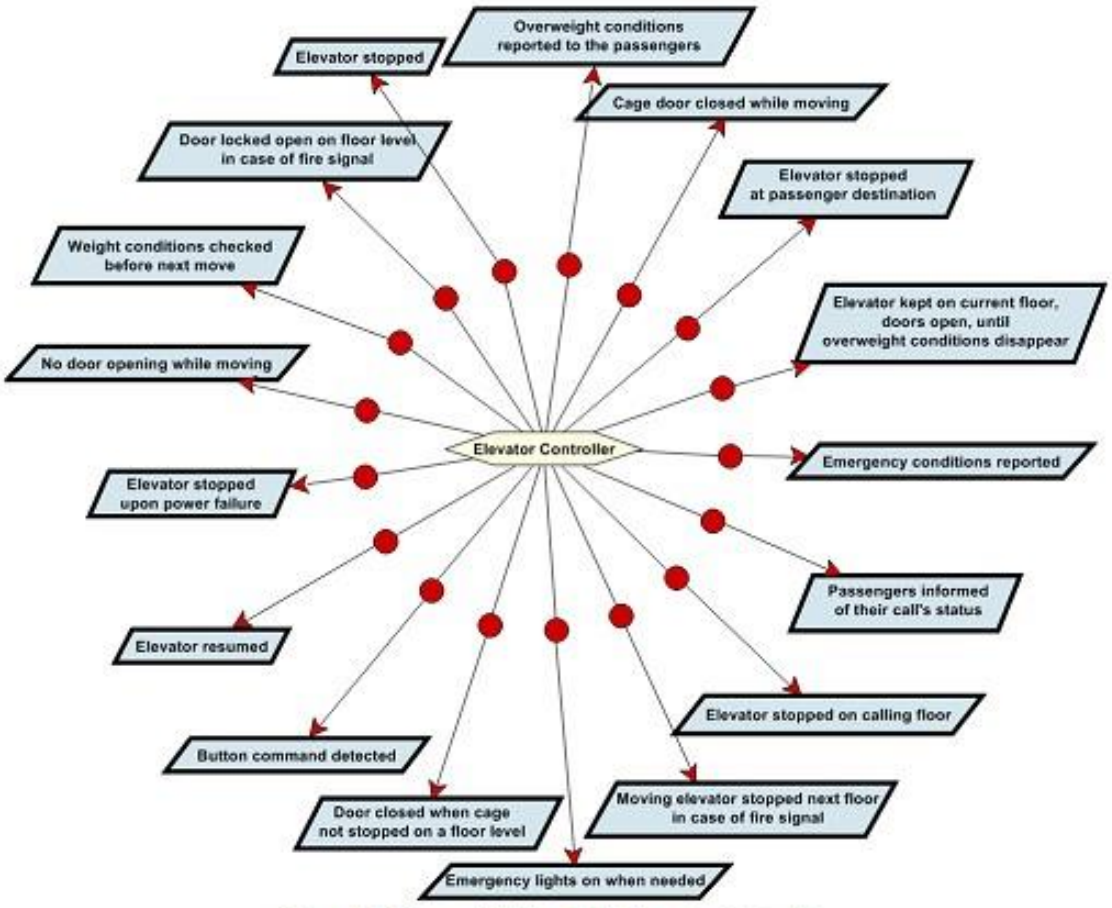
Intermediate Refinement: (Responsibilities Added for all Requirements)


System protected against fire
Elevator disabled in case of fire
Equipment to protect against fire available inside cage
Fireproof cage
Moving elevator stopped next floor in case of fire signal
Door locked open on floor level in case of fire signal
Elevator Company
Emergency stop available
Elevator Controller

Responsibilities for the Elevator Company


Elevator equipped with floor doors
Fireproof cage
Way to escape provided
Equipment to protect against fire available inside cage
Software secured
Emergency power available
Elevator Company
Cage button panel includes a stop button (Expectation?)
Elevator equipped with a breakdown alarm
Elevator cage has a door
Button-based interface provided
Resume button provided

Responsibilities for the Elevator Controller


Elevator stopped
Overweight conditions reported to the passengers
Cage door closed while moving
Door locked open on floor level in case of fire signal
Elevator stopped at passenger destination
Weight conditions checked before next move
Elevator kept on current floor, doors open, until overweight conditions disappear
No door opening while moving
Elevator Controller
Emergency conditions reported
Elevator stopped upon power failure
Passengers informed of their call's status
Elevator resumed
Elevator stopped on calling floor
Button command detected
Door closed when cage not stopped on a floor level
Moving elevator stopped next floor in case of fire signal
Emergency lights on when needed
Figure 22. Responsibilities of the Elevator Controller

Summary

- **KAOS Responsibility model**
- **Responsibility model notations**

Lecture # 12

Week-12 (Requirement Change Management)

Requirement Change Management

❖Requirements get changed during the course of development. It is almost impossible to stop the requirements from changing. Different software development approaches tackle changing requirement in different ways. Unlike Waterfall or document driven approaches of software development, agile methodologies welcome change during the course of software development but at the same time manage the changes in a systematic manner.

Agenda

- ❖**Overview**
- ❖**Reasons for Requirements Changes**
- ❖**Agile Manifesto**
- ❖**Different Form of Agile, and their working**
- ❖**Rational Unified Process**
- ❖**Requirement Change Management in Agile**
- ❖**Tools**

Overview

- ❖Importance/Significance
  - ❖ **Reasons for Requirements Changes**
- ❖Lack of domain knowledge at start
- ❖Inconsistent requirements
- ❖Change in customer prioritization
- ❖Change in platform or environment
- ❖Change due to expensiveness or difficult to implement

- ❖The changes in organization

Agile Manifesto

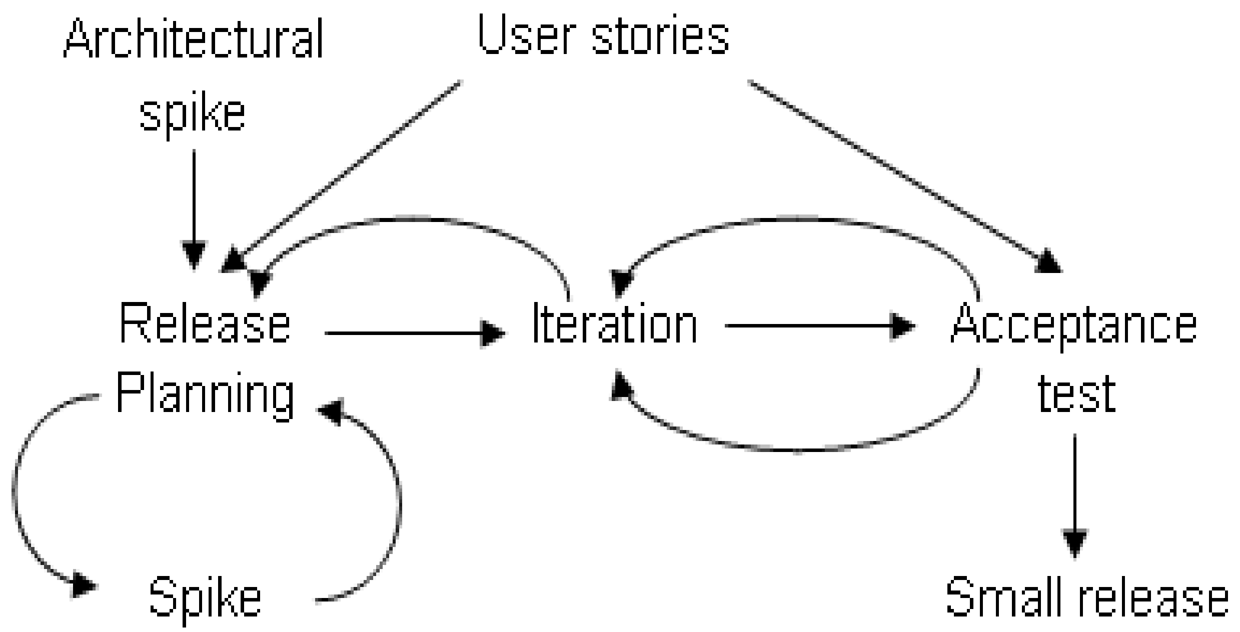
- ❖Agile focus and prefer more on individuals capabilities and expertise and their interaction rather than tools, techniques and processes.
- ❖Agile concentrate to have working application rather than formality of having large documentation.
- ❖Agile highly contemplate to have a very close relationship with customer in terms of continuous and constant conversation rather than contract negotiation.
- ❖Agile assures to adopt and respond frequently over change request rather than following a plan.

Extreme Programming (XP)

- ❖Extreme Programming is the most famous agile technique. XP use story cards for elicitation. A user story is the description that provides business value to the customer
- ❖These rules are described below:
- ❖The Planning Game
- ❖Small Releases
- ❖Metaphor
- ❖Simple Design
- ❖Tests

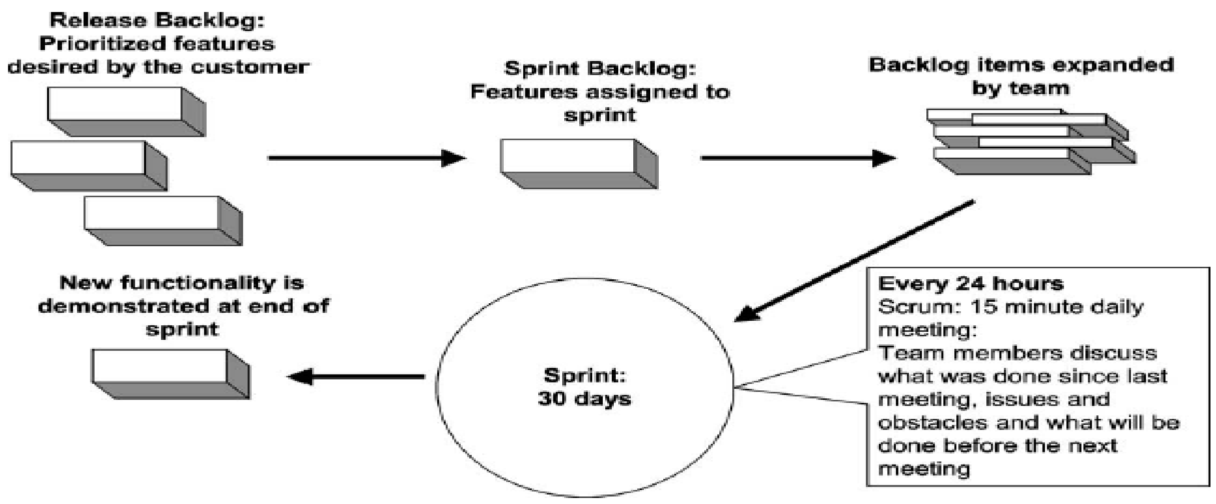
- ❖Refactoring
- ❖Pair Programming
- ❖Collective Ownership
- ❖Continuous Integration
- ❖40-hour Week
- ❖On-site customer
- ❖Coding Standard

Extreme Programming (XP)



Scrum

- ❖Scrum is another popular agile technique used to develop and manage software. The following figure explains the activities performed in Scrum.


Release Backlog: Prioritized features desired by the customer
Sprint Backlog: Features assigned to sprint
Backlog items expanded by team
Every 24 hours
Scrum: 15 minute daily meeting:
Team members discuss what was done since last meeting, issues and obstacles and what will be done before the next meeting
Sprint: 30 days
New functionality is demonstrated at end of sprint

Rational Unified Process

❖RUP---- Document Driven approach

- **RUP Phases, RUP Disciplines**
- Extensive planning, Codified Process, Heavy Documentation , Big Design up Front
- **Strengths:**
  - Straightforward, methodical and structured nature, Predictability, stability and high assurance
- **Weaknesses**
  - Slow adaptation to rapidly changing business requirements
  - A tendency to be over budget
  - A tendency to behind schedule
  - Failed to provide dramatic progress in productivity, simplicity and reliability

- ❖ **XP and Scrum-- Agile Software Development Approaches**
  - ▪ Iterative and Incremental development, Customer collaboration, Frequent Delivery, Light and fast development, Light documentation
  - ▪ **Strengths:**
    - Short development cycle, Higher customer satisfaction, Low bug rates, Quick adaptation to rapidly changing requirements
    - Highest Priority Work ,Constant Feedback, Control over Cost and Schedule
  - ▪ **Weaknesses**
    - Significant document reduction, Heavy dependence on individual knowledge, Not suitable for critical safety systems, Not suitable for large scale systems, Frequent change effect cost and schedule, Managed prioritization, Organizational structure

Agile (XP and Scrum) requirements change management process

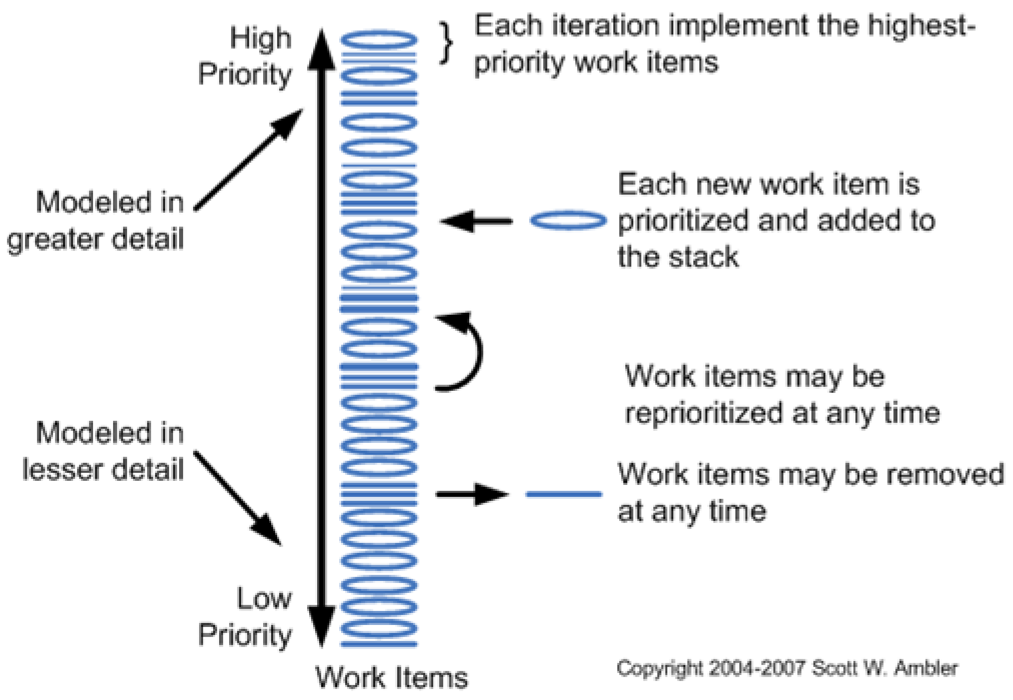
- ❖ The agile change management process handles changes in the beginning of each iteration of the development cycle. It may be a sprint in Scrum or iteration in XP and so on. The key stakeholders in change management process are the managers, developing team and off course customers or product owner.

- ❖ As agile’s development period is considerably short for a particular iteration therefore it is understood that all the requirements can not be implemented in one go therefore there is a pile or stack of requirements and the relevant stakeholders have to decide which requirements to implement in the one iteration. Therefore the prioritization is also a continuous process in agile development and the requirement stack is constantly updated as a result of update.

Change Management Process

- ❖This aspect of Agile shows a different picture from that of traditional development where requirements are collected once and changes made in that requirement set are rare. Here agile is welcoming the change even after every iteration. Therefore agile is gaining wide acceptance in today’s development where we also have high speed development environments.

Change Management Process


High Priority
Each iteration implement the highest-priority work items
Modeled in greater detail
Each new work item is prioritized and added to the stack
Work items may be reprioritized at any time
Modeled in lesser detail
Work items may be removed at any time
Low Priority
Work Items
Copyright 2004-2007 Scott W. Ambler

Lifecycle of change management in agile

- ❖**Start:** In the start of each iteration, the team takes the highest priority requirement that can be completed in the specified iteration period. This requirement which is now going to be implemented is well understood with the help of customer and other related stakeholders and documents etc. Necessary planning, documentation or modeling can also be done at this stage so as to achieve the goal within the specified time and budget.
- ❖**Middle**: During development they may take help from customer as well as from other relevant stakeholders to have better understanding of the requirement. The

aim is to build the software that best meets the requirement.

- ❖ **End:** The working product developed can be deployed and it is preferable to deploy so as to take the feedback from the end-users. The acceptance can also be run on the developed software so that the necessary quality can also be ensured.

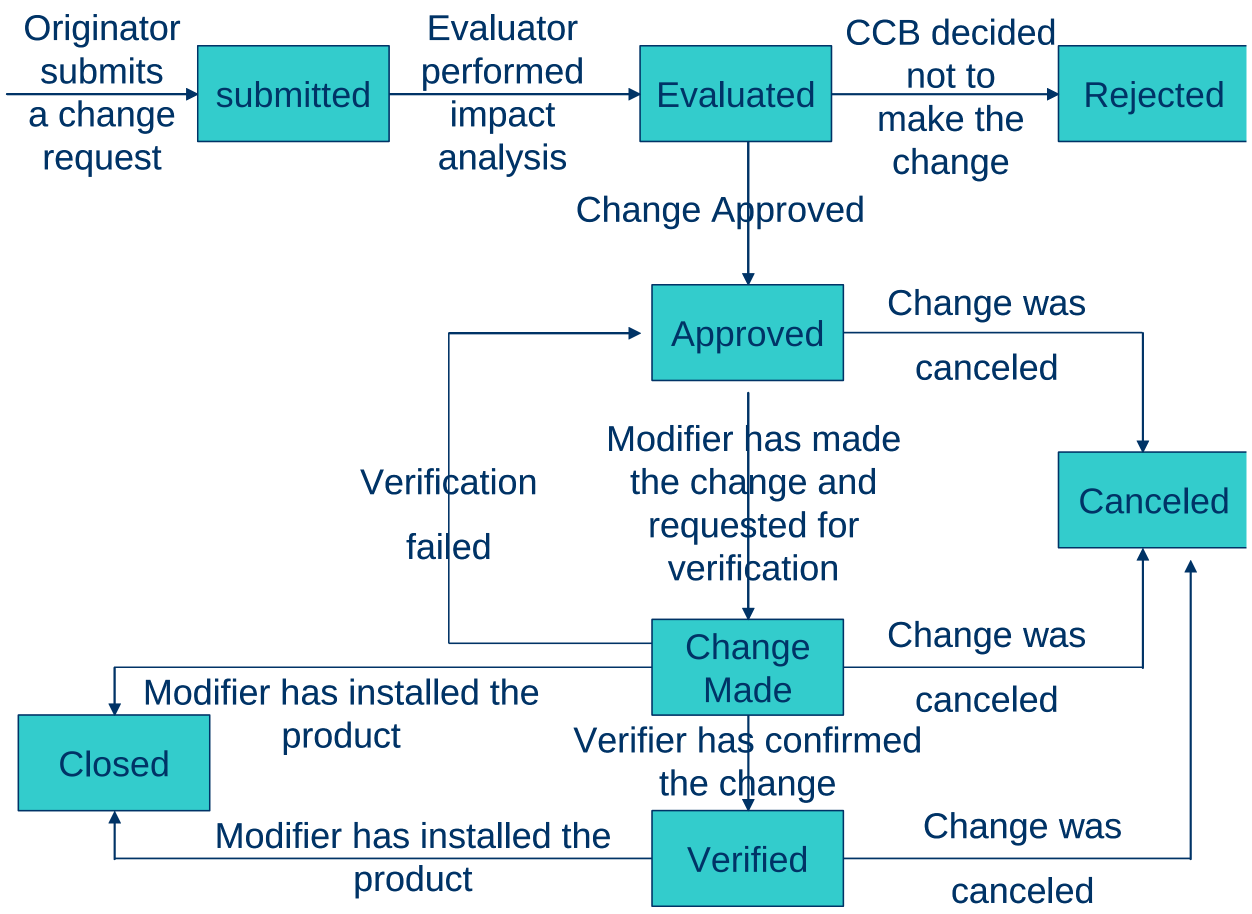
What Kind of Tool Do We Need?

- ❖ Word processor (Microsoft Word with templates…)
- ❖ Spreadsheet (Microsoft Excel…)
- ❖ Industrial-strength, commercial RM tools
  - IBM/Telelogic DOORS, IBM Requisite Pro, Borland CaliberRM…
- ❖ Internal tools
  - GenSpec (Hydro-Quebec)…
- ❖ Open source RM tools
  - OSRMT: http://sourceforge.net/projects/osrmt
- ❖ Bug tracking tools (free or not)
  - Bugzilla…
- ❖ Collaboration tools (free or not)
  - TWiki…

Lecture # 13

Week-13 (Requirement evaluation and prioritization)

Requirement Change Management Process


Originator submits a change request
submitted
Evaluator performed impact analysis
Evaluated
CCB decided not to make the change
Rejected
Change Approved
Approved
Change was canceled
Canceled
Verification failed
Modifier has made the change and requested for verification
Change Made
Change was canceled
Modifier has installed the product
Closed
Verifier has confirmed the change
Modifier has installed the product
Verified
Change was canceled

Comparison between Agile and Conventional Philosophy

❖ Both Agile and Document Driven (companies following traditional approaches) companies face similar issues with respect to requirements management but they use to handle it in different ways. Some of the differences between these two approaches that have been identified as a result of a study are explained below:

❖ **Changing requirements(88%--13%) :**

- ❖ **Reason for Requirement Change(knowledge Deepening):**

Comparison:

- ❖ **Requirements Gathering Process(complete Specification/incrementally):**
- ❖ **Means of Communication (document/Onsite customer)**:
- ❖ **Contracts with Clients(Strict/flexible):**
- ❖ **Attitudes towards Change(Difficult task/Welcome)**:
- ❖ **Relationship with Customer(satisfactory/Close):**

Freezing the Requirements:

- ❖ Embracing requirement changes in Agile does not mean that requirement could be changed at any stage of the software development process. What it really means is that unlike traditional approaches like waterfall model where you do not have the privilege to request requirement changes after the start of development cycle, here in agile the customer or the product owner have an opportunity to add, modify or remove any requirement from the requirement stack.
- ❖ In fact Agile has laid down a process in every technique to accept change in an organized way for

next iteration instead of forcing to implement the new requirement in the current release.

- ❖XP and OpenUP allows accepting change to a certain extent during the development iteration but recommends suggesting the requirement for next iteration.

Requirements Prioritization:

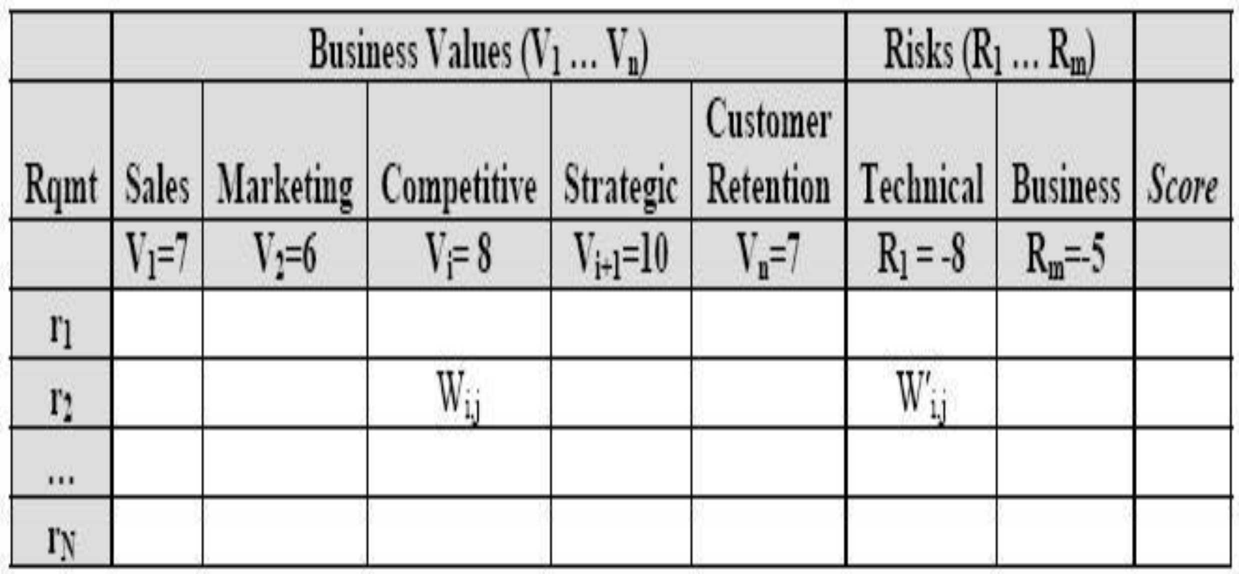
- ❖Nature of Agile development lifecycle demands
- ❖Stakeholders status in the beginning
- ❖feedback after every development iteration
- ❖agile development framework allows the stakeholders to re-prioritize the requirements
- ❖factors like market uncertainty, technical uncertainty, project duration and project budget that demands that the requirements should be analyzed and re-prioritized

Value Oriented Prioritization:

- ❖Many studies have been carried out to propose good and scientific methods of prioritization
- ❖One of the techniques for requirement prioritization was named as "Value Oriented Prioritization". This technique suggests that the company or the stakeholders should identify the business value areas like Sales, Marketing, Strategic, Customer Retention

etc. Then a positive numeric value should be associated with each of these business value areas.

- ❖After that, values are also assigned to each requirement after negotiation and consultation with all the relevant stakeholders.
- ❖Along with identifying the business value areas, the associated risks should also be identified and a certain negative value should also be assigned to each risk as shown in fig.

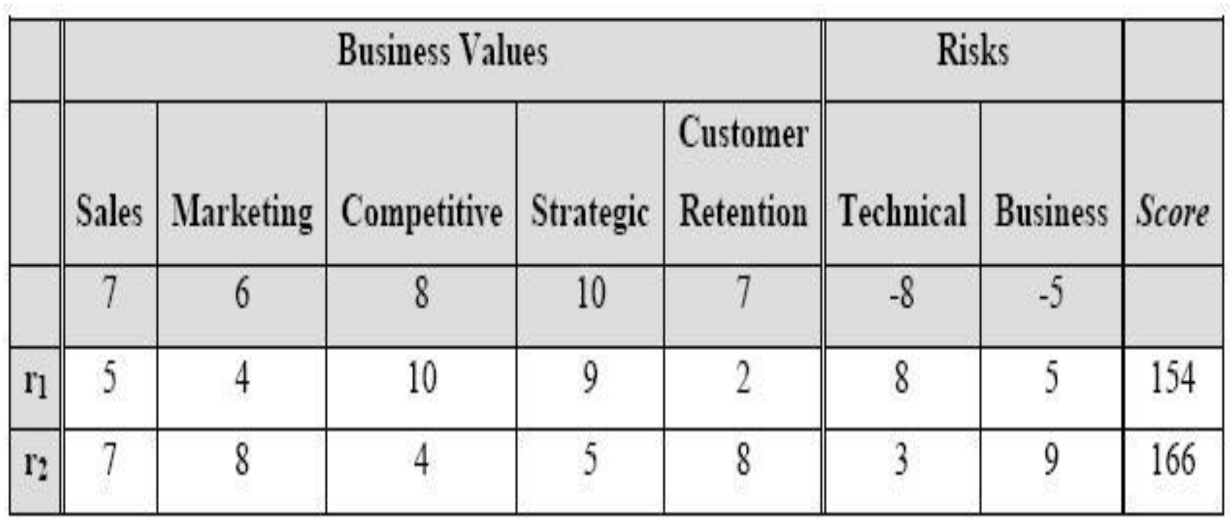


	Business Values ($V_1 \ldots V_n$)					Risks ($R_1 \ldots R_m$)		
Rqmt	Sales	Marketing	Competitive	Strategic	Customer Retention	Technical	Business	*Score*
	V_1=7	V_2=6	V_i= 8	V_{i+1}=10	V_n=7	R_1 = -8	R_m=-5	
r_1								
r_2			$W_{i,j}$			$W'_{i,j}$		
...								
r_N								

Value Oriented Prioritization

- ❖Then all the requirements are listed and given a numeric value against business value areas as well as to the risk associated with that business value. In this way the weight of each requirement can be calculated against each business value area and similarly the weight of risks can also be identified. The final weight of a requirement can be calculated by subtracting the sum of weights of the risks from the sum of weights of business values against each requirement.

Example



	Business Values					Risks		
	Sales	Marketing	Competitive	Strategic	Customer Retention	Technical	Business	*Score*
	7	6	8	10	7	-8	-5	
r1	5	4	10	9	2	8	5	154
r2	7	8	4	5	8	3	9	166

Evaluation

- ❖Requirements Evaluation from End user’s perspective
- ❖Usability
- ❖Usability Evaluation
- ❖Usability Evaluation Methods
- ❖The process of systematically collecting *data* that *informs* us about what it is like for a particular or group of *users* to use a *product* for a particular *task* in a certain type of *environment.*

Why, what, where, and when to evaluate:

- Why: to check that users can use the product and that they like it.
- What: a conceptual model, early prototypes of a new system and later, more complete prototypes.
- Where: in natural and laboratory settings.
- When: throughout design; finished products can be evaluated to collect information to inform new products.

Usability

Usability has been defined by the International Standards Organization (ISO) as “the extent to which the product can be used by specified users to achieve specified goals with

- effectiveness
- efficiency
- satisfaction

Usability Evaluation

- **Metrics used to measure Usability:**
- 1. Time to complete a task

  3. Fraction of task completed

  4. Fraction of task completed in a given time

  5. Number of errors

  6. Time spent on errors

Usability Evaluation Methods

- Testing

- ❖Inspection
- ❖Inquiry

Lecture # 14

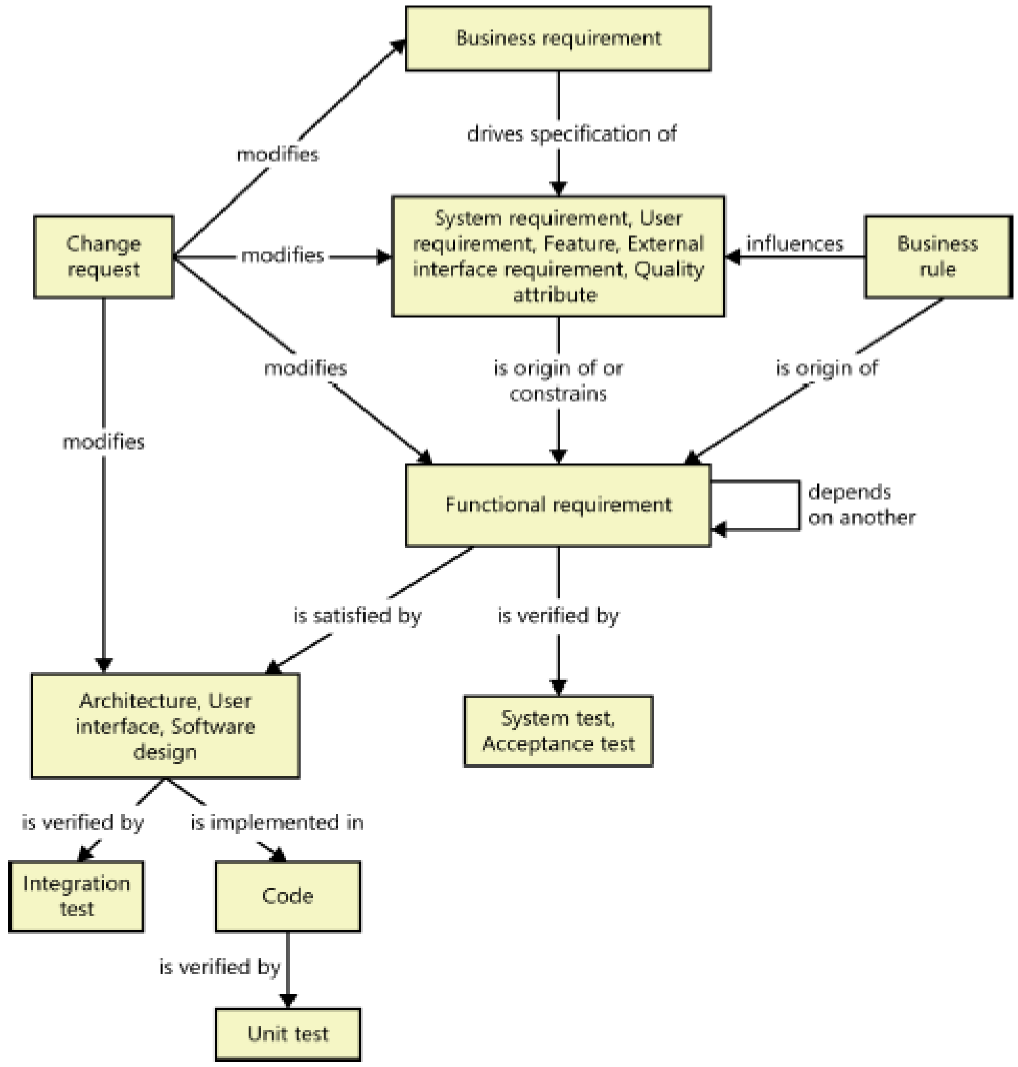
Week-14 (Requirements Traceability)

Introduction

- **What is traceability?**
  - **"the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another." [IEEE-610]**
- **Requirements Traceability:**
  - **The requirements traceability is the ability to describe and follow the life of a requirement, in both a forward and backward direction.**

Requirement Traceability (RT)

- **Why requirement Traceability**
  - **Finding missing requirements**
  - **Finding unnecessary requirements**
  - **Certification and compliance**
  - **Change impact analysis**
  - **Maintenance and**
  - **Project tracking etc.**



Classification of Requirement Traceability

- **Backward-from traceability**

Links requirements to their sources i.e documents or people

- **Forward-from traceability**

Links requirements to design and implementation components

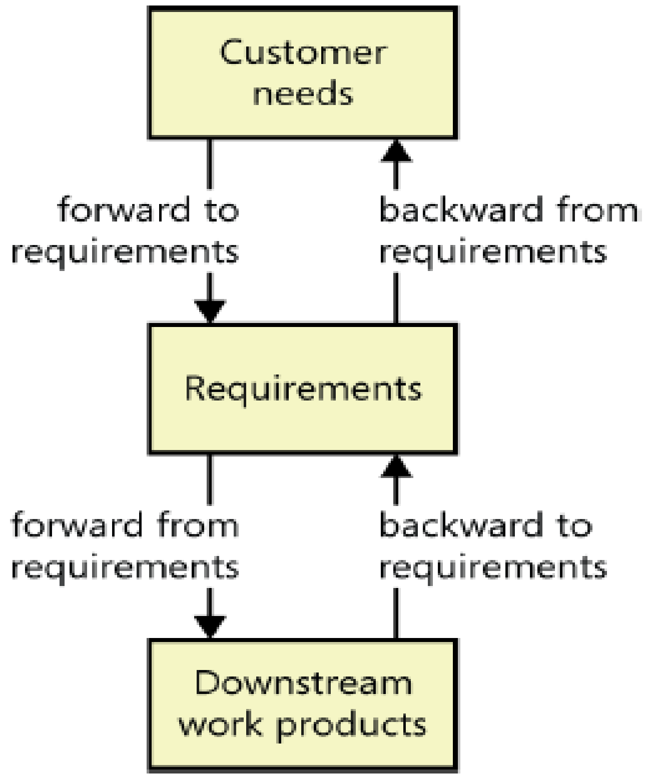
- **Backward-to traceability**

Links design and implementation components back to requirements

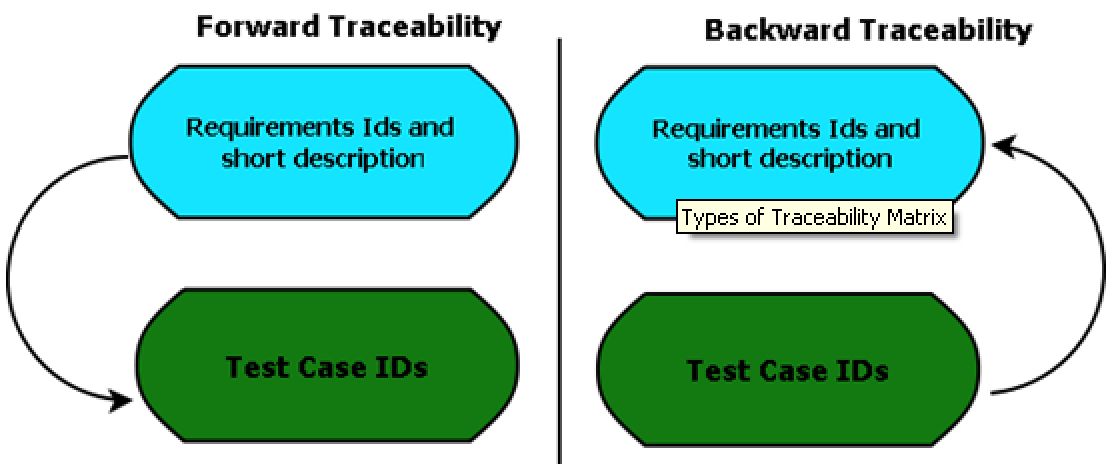
- **Forward-to traceability**

Links requirements back to their sources

Classification of RT:



Classification of RT:


Forward Traceability
Requirements Ids and short description
Test Case IDs
Backward Traceability
Requirements Ids and short description
Types of Traceability Matrix
Test Case IDs

Categories of requirement traceability

- **Requirements-sources traceability**

Links the requirement and their sources which specified the requirement

- **Requirements-rationale traceability**

Links the requirement with a description of why that requirement has been specified

- **Requirements-requirements traceability**

Links requirements with other requirements which are, in some way, dependent on them

- **Requirements-architecture traceability**

Links requirements with the sub-systems where these requirements are implemented

- **Requirements-design traceability**

Links requirements with specific hardware or software components in the system, which are used to implement the requirement

- Requirements-interface traceability

Links requirements with the interfaces of external systems, which are used in the provision of the requirements

How is tracing performed?

- Each element is given a unique identifier
  - Element – requirement, design attribute, test, etc
- Linkages done manually and managed by a CASE tool
- Traceability tables are made
  - Matrix

Traceability matrix

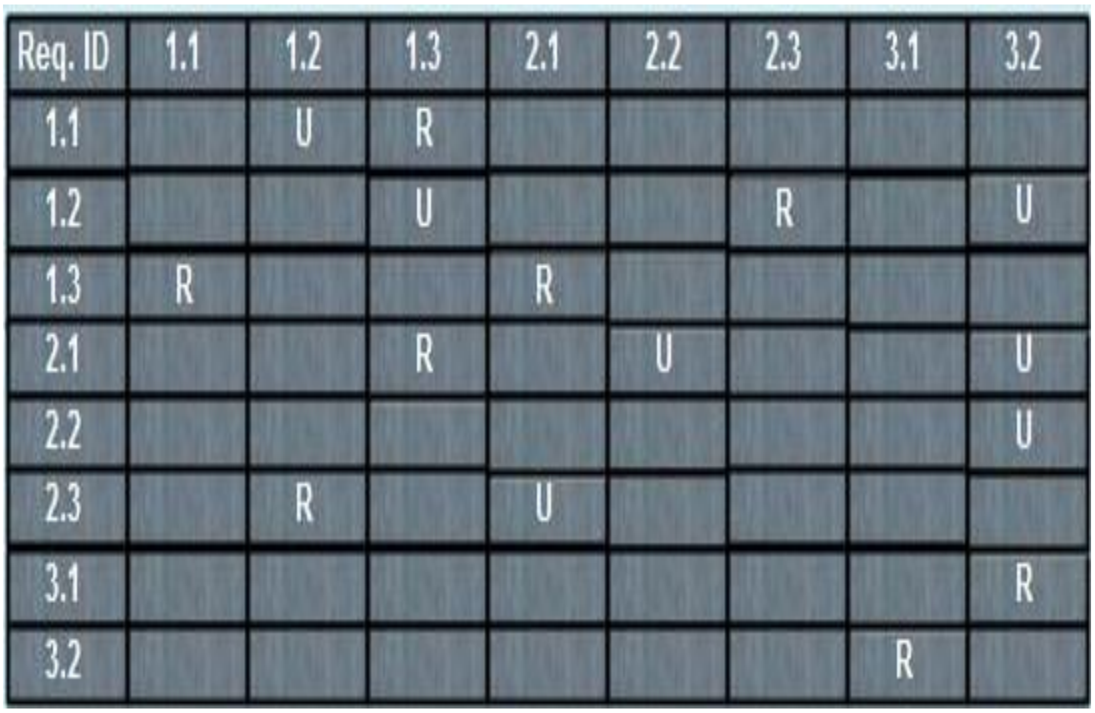
- A traceability matrix is a document that co-relates any two-baseline documents that require a many-to-many relationship to check the completeness of the relationship.

- **It is used to track the requirements and to check the current project requirements are met.**

Types of Traceability Matrix

- **Types of Traceability Matrix**
  - **Forward traceability**
  - **Backward or reverse traceability**
  - **Bi-directional traceability (Forward+Backward)**
- **Advantage of Requirement Traceability Matrix**
  - **It confirms 100% test coverage**
  - **It highlights any requirements missing or document inconsistencies**
  - **It shows the overall defects or execution status with a focus on business requirements**

Requirement traceability matrix



Req. ID	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

Requirement traceability Tools

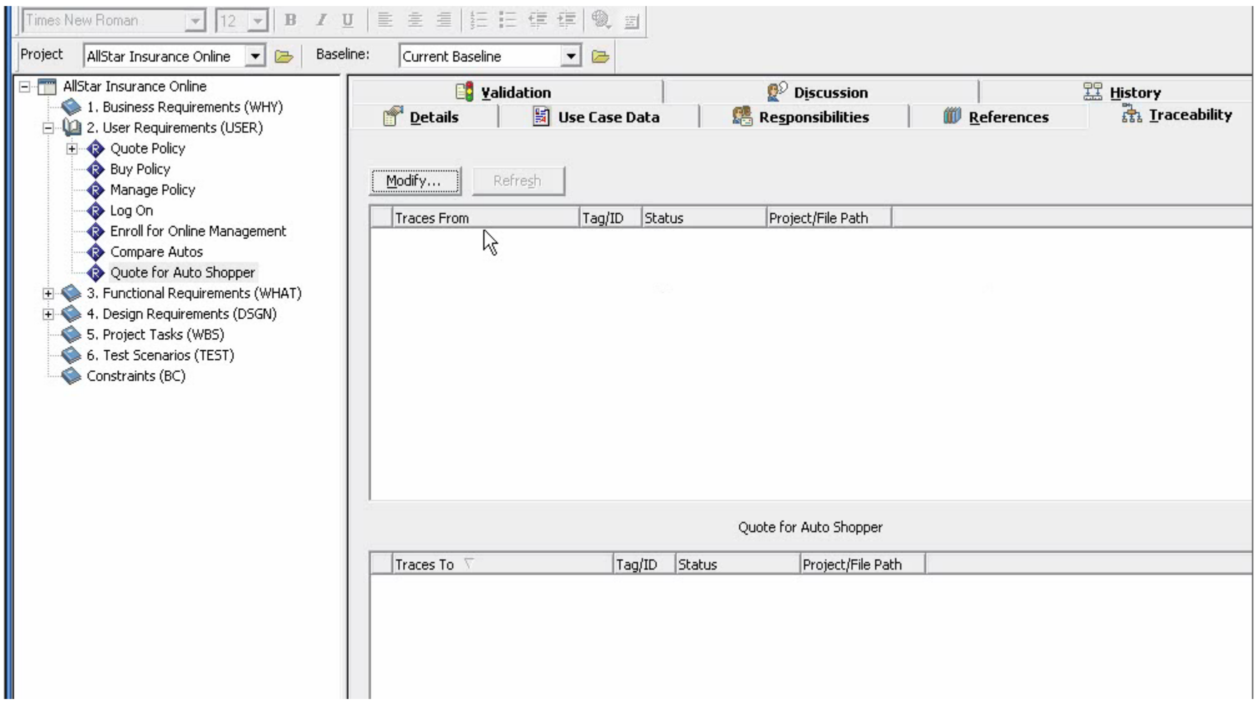
- **CASE Tools**
- **Characteristics**
  - **Hypertext linking**
  - **Unique identifiers**
  - **Syntactical similarity coefficients**
- **Problems**
  - **Hypertext linking and syntactical similarity does not consider context**
  - **Unique identifiers do not show requirement information**

- Choosing architecture view and classification schemas will always be manual

Caliber-RM

- Caliber-RM
  - Centralized repository
  - Requirements traceability across the lifecycle
  - Impact analysis

Caliber-RM

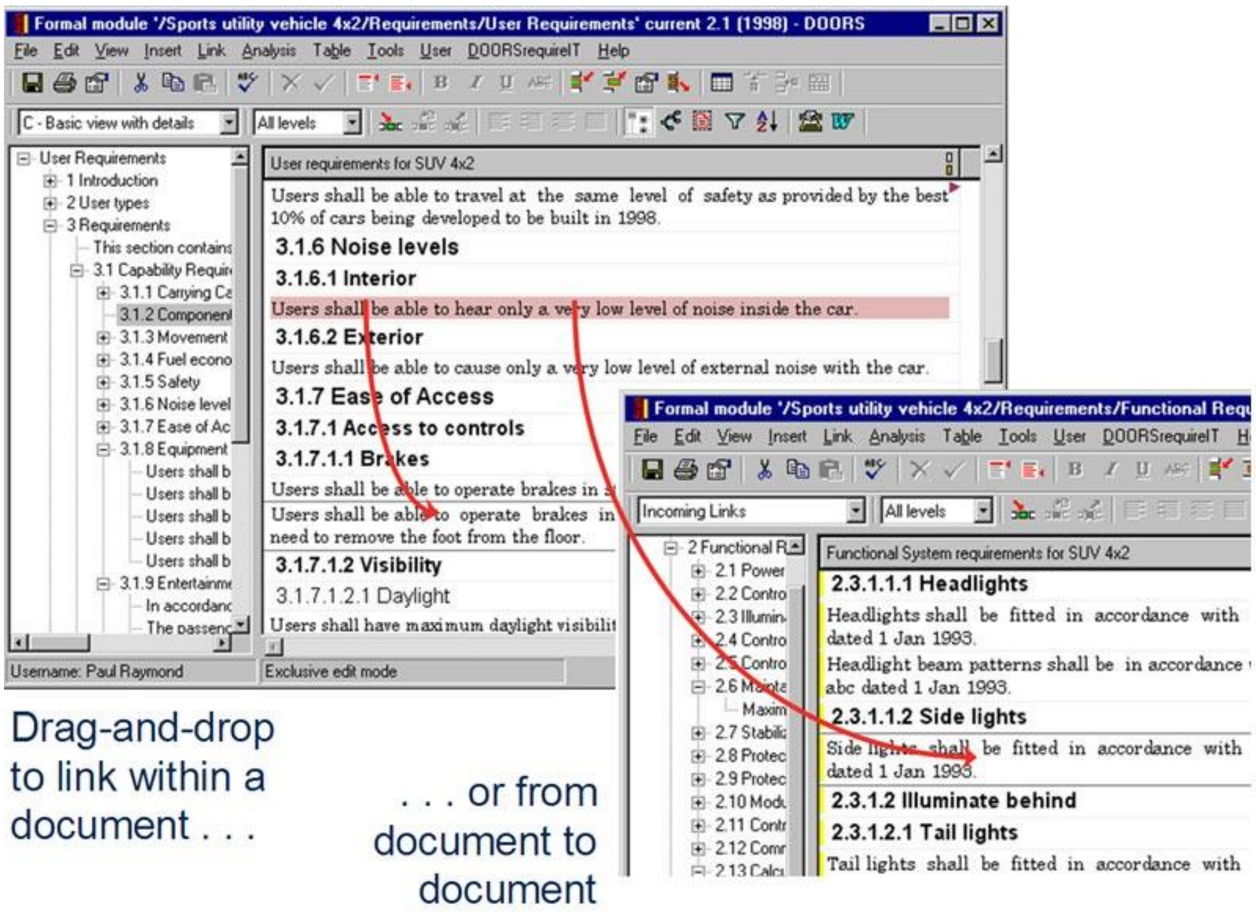


Rational Dynamic Object Oriented Requirements System(DOORS)

- DOORS

- **Telelogic**
- **“capture, link, trace, and manage”**
- **For large applications**

DOORS


Formal module '/Sports utility vehicle 4x2/Requirements/User Requirements' current 2.1 (1998) - DOORS
File Edit View Insert Link Analysis Table Tools User DOORSrequireIT Help
C - Basic view with details
All levels

User Requirements
1 Introduction
2 User types
3 Requirements
This section contains
3.1 Capability Requir
3.1.1 Carrying Ca
3.1.2 Component
3.1.3 Movement
3.1.4 Fuel econo
3.1.5 Safety
3.1.6 Noise level
3.1.7 Ease of Ac
3.1.8 Equipment
Users shall b
Users shall b
Users shall b
Users shall b
Users shall b
3.1.9 Entertainme
In accordanc
The passeng

User requirements for SUV 4x2

Users shall be able to travel at the same level of safety as provided by the best 10% of cars being developed to be built in 1998.

3.1.6 Noise levels

3.1.6.1 Interior

Users shall be able to hear only a very low level of noise inside the car.

3.1.6.2 Exterior

Users shall be able to cause only a very low level of external noise with the car.

3.1.7 Ease of Access

3.1.7.1 Access to controls

3.1.7.1.1 Brakes

Users shall be able to operate brakes in

Users shall be able to operate brakes in need to remove the foot from the floor.

3.1.7.1.2 Visibility

3.1.7.1.2.1 Daylight

Users shall have maximum daylight visibilit

Username: Paul Raymond
Exclusive edit mode

Formal module '/Sports utility vehicle 4x2/Requirements/Functional Requ
File Edit View Insert Link Analysis Table Tools User DOORSrequireIT H
Incoming Links
All levels

2 Functional R
2.1 Power
2.2 Contro
2.3 Illumin
2.4 Contro
2.5 Contro
2.6 Mainte
Maxim
2.7 Stabiliz
2.8 Protec
2.9 Protec
2.10 Modu
2.11 Contr
2.12 Comr
2.13 Calcu

Functional System requirements for SUV 4x2

2.3.1.1.1 Headlights

Headlights shall be fitted in accordance with dated 1 Jan 1993.

Headlight beam patterns shall be in accordance abc dated 1 Jan 1993.

2.3.1.1.2 Side lights

Side lights shall be fitted in accordance with dated 1 Jan 1993.

2.3.1.2 Illuminate behind

2.3.1.2.1 Tail lights

Tail lights shall be fitted in accordance with

Drag-and-drop to link within a document . . .

. . . or from document to document

References

- **Software Requirements Third Edition by Karl Wiegers and Joy Beatty**
- **James D. Palmer**

- **Requirements Engineering: Processes and Techniques' by G. Kotonya and I. Sommerville, John Wiley & Sons, 1998**

Lecture # 15

Week-15 (Requirement Document, Requirement verification & Validation)

Introduction:

- **What is requirement document?**
  - **Software requirements specifications or SRS**
  - **The requirements document is a formal document used to communicate the requirements to customers, engineers and managers**
- **Requirements includes:**
  - **The services and functions which the system should provide**
  - **The constraints under which the system must operate**
  - **Overall properties of the system**

Requirement document

- **Users of the requirements**
  - **System customers**
  - **Managers**
  - **System engineers**
  - **System test engineers**
  - **System maintenance engineers**

- How to Organize an SRS?
- Clients/developers may have there own way of organizing an SRS
- US Department of Defense
- NASA
- IEEE/ANSI 830-1993 Standard
- IEEE Std. 830-1998

- **Characteristics of good SRS**
  - **Correct**
  - **Unambiguous**
  - **Complete**
  - **Consistent**
  - **Verifiable**
  - **Modifiable**
  - **Traceable**

IEEE/ANSI Standard 830-1993

- **Parts of IEEE/ANSI Standard 830-1993**
- **Introduction**
- **General description**
- **Specific requirements**
- **Appendices**
- **Index**

Parts of IEEE/ANSI Standard 830-1993

- **Introduction:**
- **1.1 Purpose of the requirements document**
- **1.2 Scope of the product**
- **1.3 Definitions, acronyms, and abbreviations**
- **1.4 References**
- **1.5 Overview of the remainder of the document**

General description

- **2.1 Product perspective**
- **2.2 Product functions**
- **2.3 User characteristics**
- **2.4 General constraints**
- **2.5 Assumptions and dependencies**

Specific Requirements

- **Covering functional, non-functional, and interface requirements.**
- **These should document external interfaces, functionality, performance requirements, logical database requirements, design constraints, system attributes, and quality characteristics**

IEEE/ANSI Standard 830-1993

<u>General discussion on IEEE Standard</u>

- **It is good starting point for organizing requirements documents**
- **First two sections are introductory chapters about background and describe the system in general terms**

- The third section is the main part of the documents and the standard recognizes that this section varies considerably depending on the type of the system

References

- IEEE Recommended Practice for Software Requirements Specifications
- 'Requirements Engineering: Processes and Techniques' by G. Kotonya and I. Sommerville, John Wiley & Sons, 1998

Verification and Validation

- Verification:
- Process in which we check a product against its specifications.
- White box, black box testing
- Validation:
- Process in which we check expectations of the users who will be using it.
- Inspection, Formal Technical Review

Defects

- software without any defects
- No, it is almost impossible to develop software without having any defect. Software and defects go side-by-side during software development. It is impossible to build a product in first instance without presence of any defects and these two cannot be separated.

Black box testing

- In this type of testing, a component or system is treated as a black box and it is tested for the required behavior. This type of testing is not concerned with how the inputs are transformed into outputs. As the system's internal implementation details are not visible to the tester. He gives inputs using

an interface that the system provides and tests the output. If the outputs match with the expected results, system is fine otherwise a defect is found.

Structural testing (white box)

- **As opposed to black box testing, in structural or white box testing we look inside the system and evaluate what it consists of and how is it implemented. The inner of a system consists of design, structure of code and its documentation etc. Therefore, in white box testing we analyze these internal structures of the program and devise test cases that can test these structures.**

Defect Removal Process

- Steps you take to check the presence of the defects
- I will run the scenario as described in the bug report and try to reproduce the defect.
- If the defect is reproduced in the development environment, I will identify the root cause, fix it and send the patch to the testing team along with a bug resolution report.

Cycloramic complexity

- E = Number of Edges
- N = Number of Nodes
- $V(G) = E - N + 2$