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.

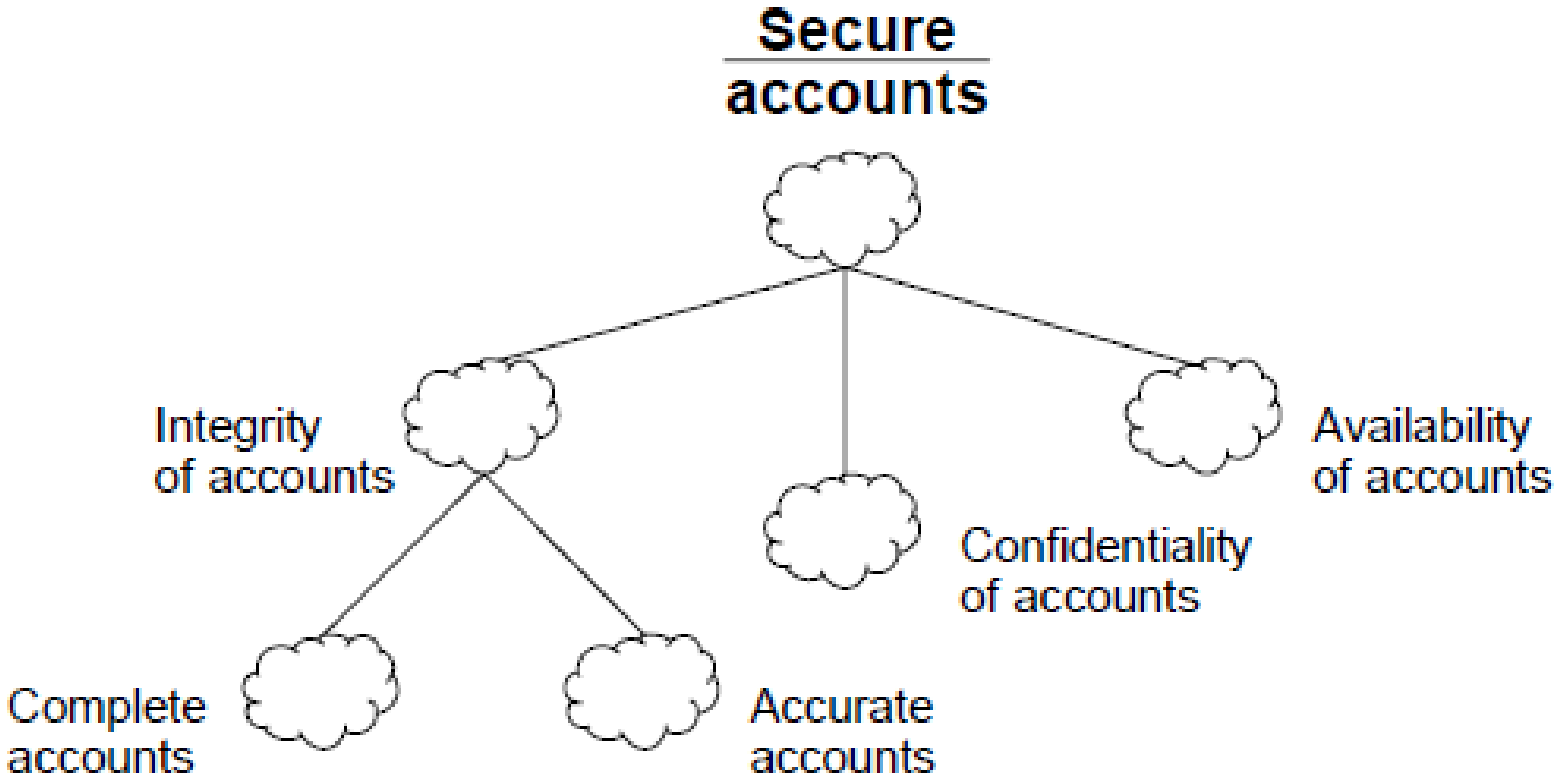
Goals:

- 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)

Framework Activities

- 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 operationalizations and claims.

Types of Softgoals in NFR framework.

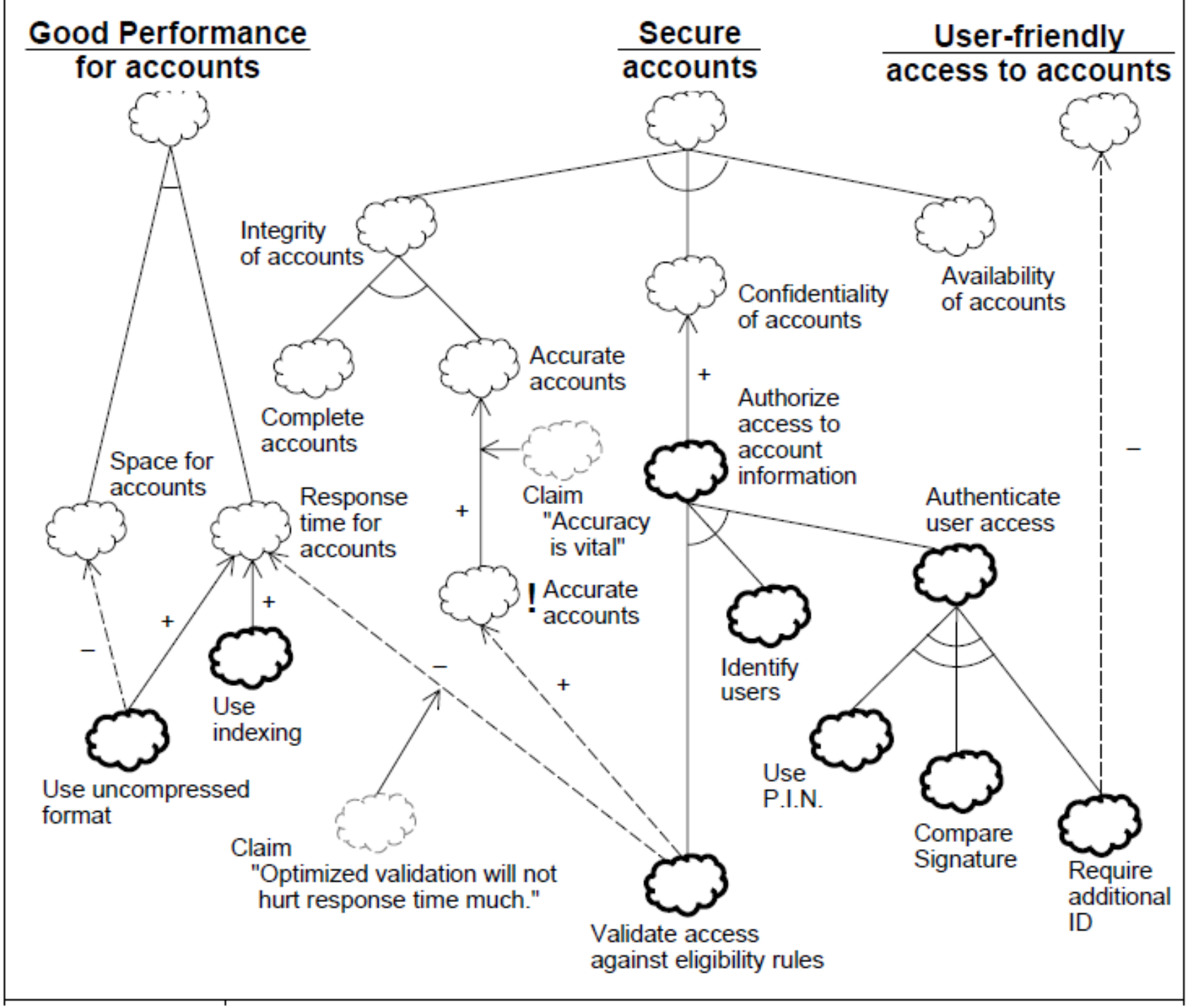
1. NFR softgoals represent non-functional requirements to be considered

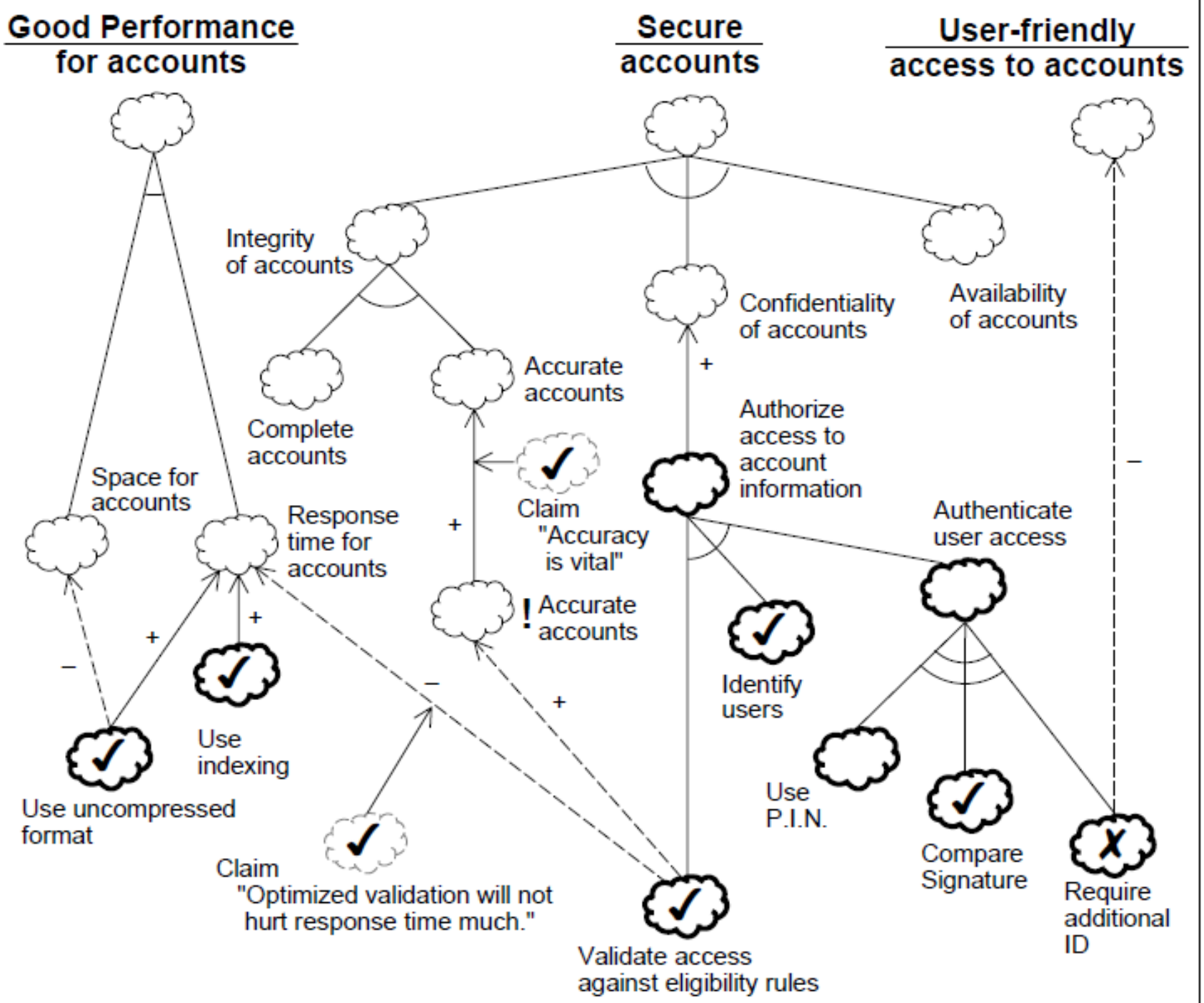
2. Operationalizing softgoals model lower-level (design) techniques for *satisficing* NFR softgoals

3. Claim softgoals 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
Space for accounts
Response time for accounts
Claim "Accuracy is vital"
Authorize access to account information
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
Space for accounts
Response time for accounts
Claim "Accuracy is vital"
Authorize access to account information
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

Summary

- NFR framework
- Framework activities
- Softgoal Interdependency Graph

KAOS (Knowledge Acquisition in Automated Specification)

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.

KAOS Ontology

- 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.

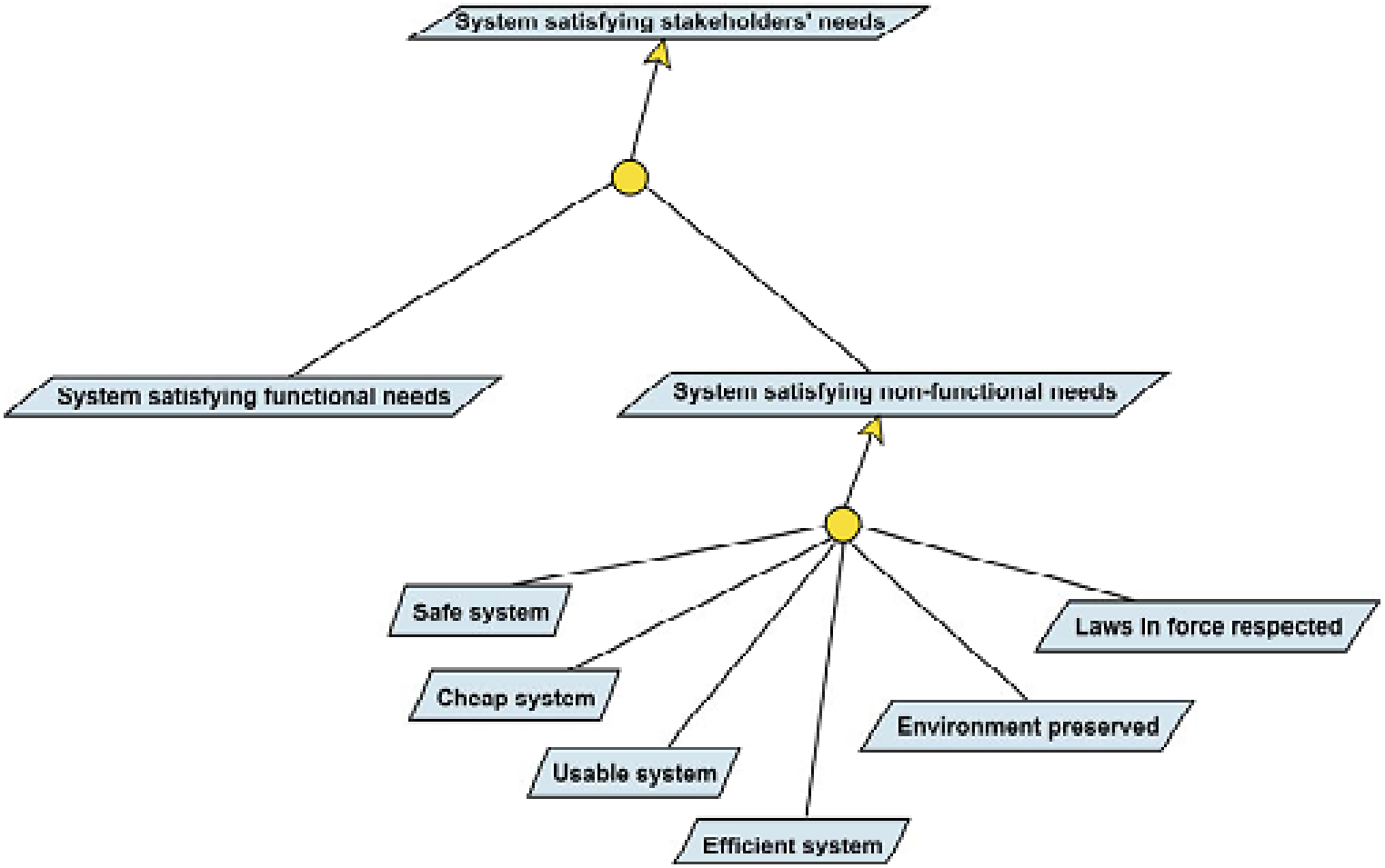
KAOS Models

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

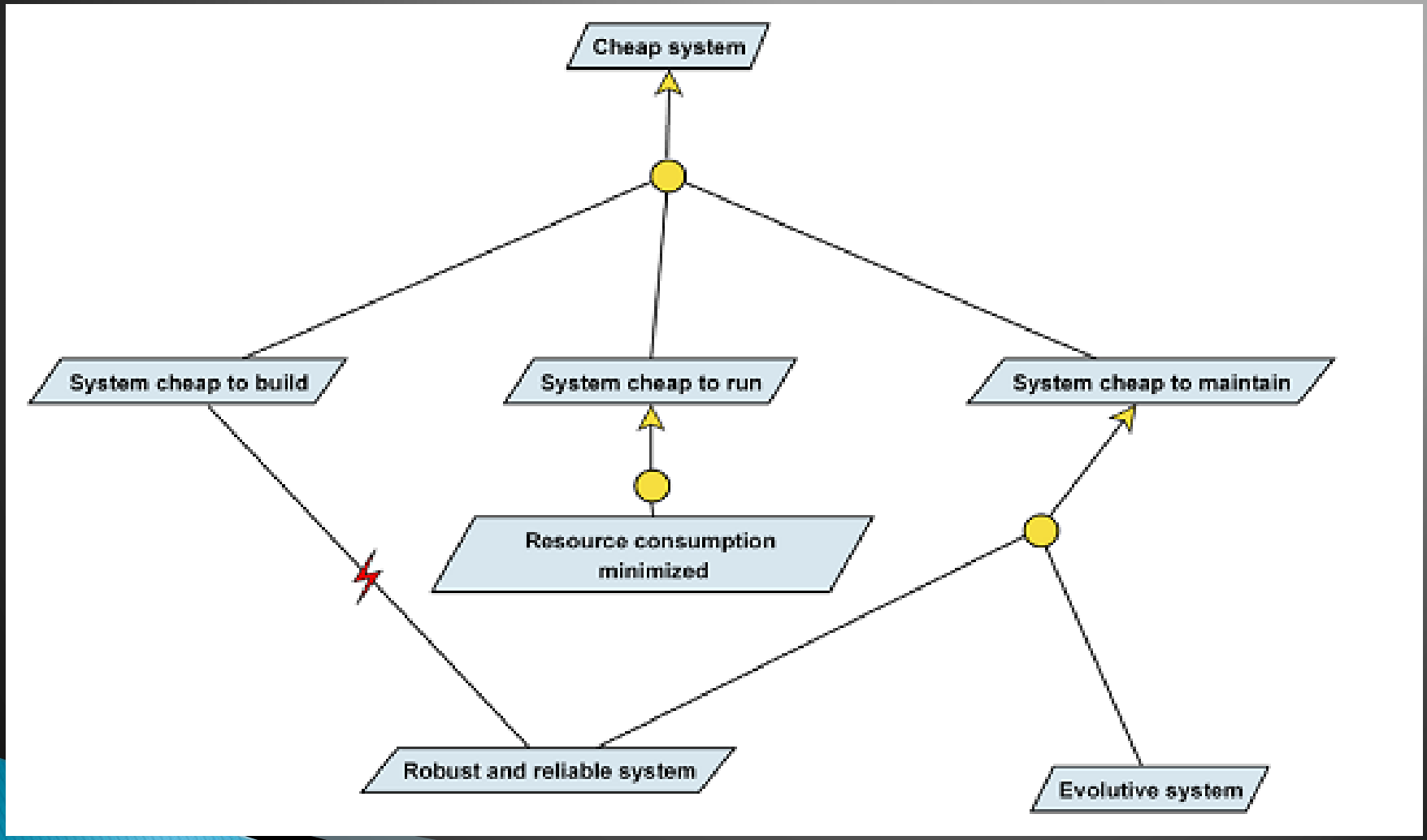
Building the Goal Model

- 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



Example: Generic Conflicting Goals

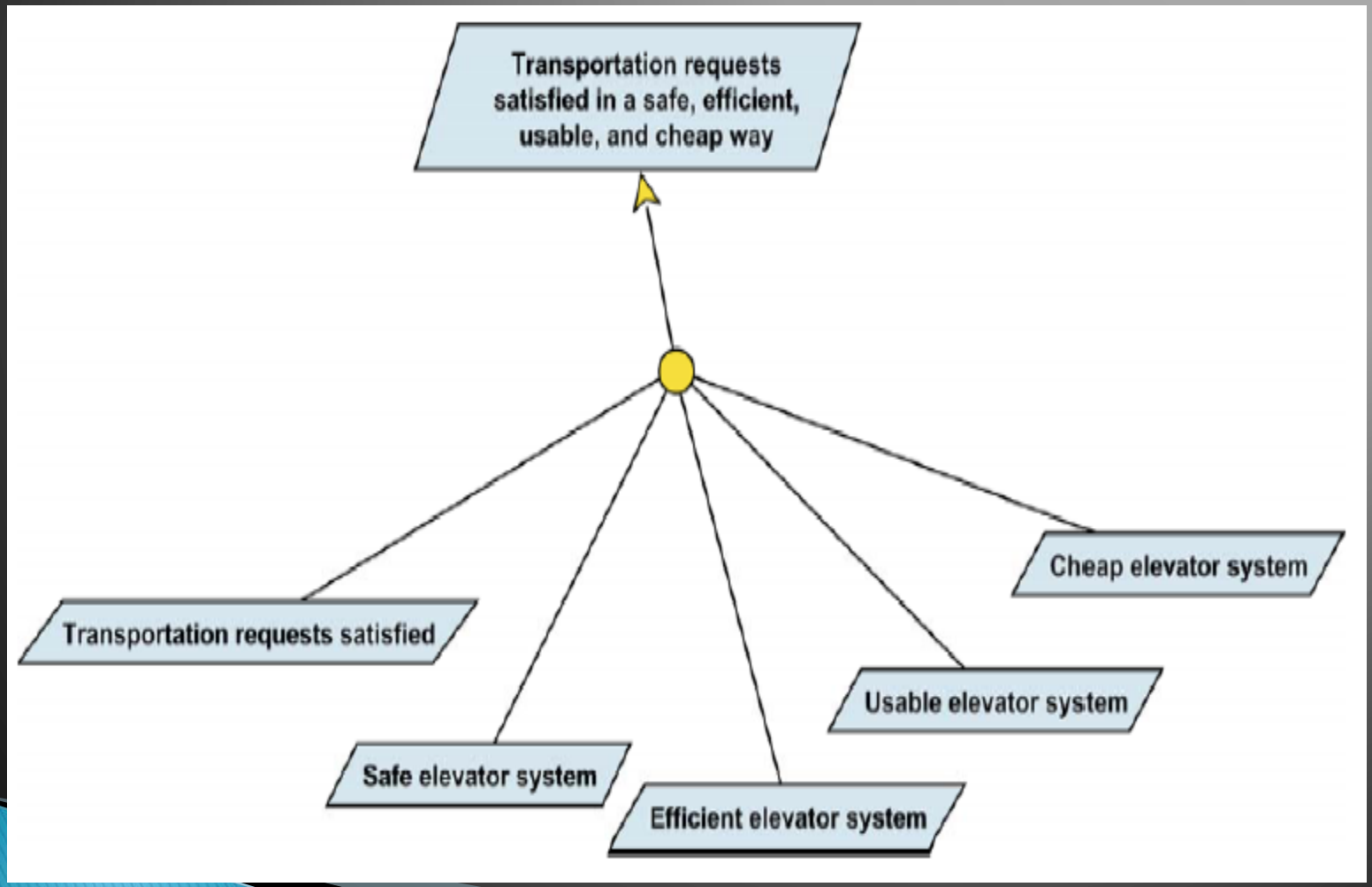


The Elevator Case Study

- 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.

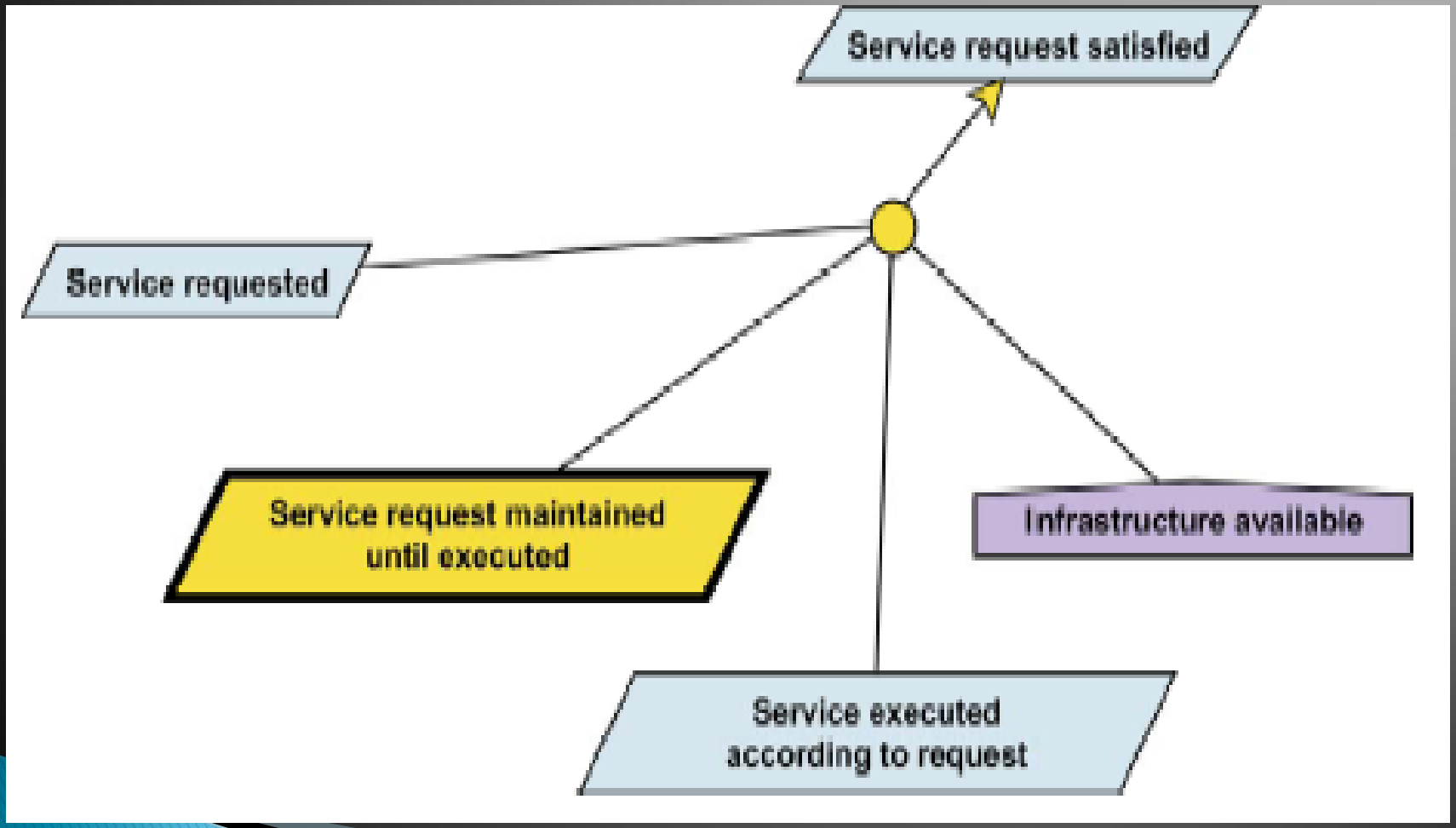
Instantiating the Generic Goal Model



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

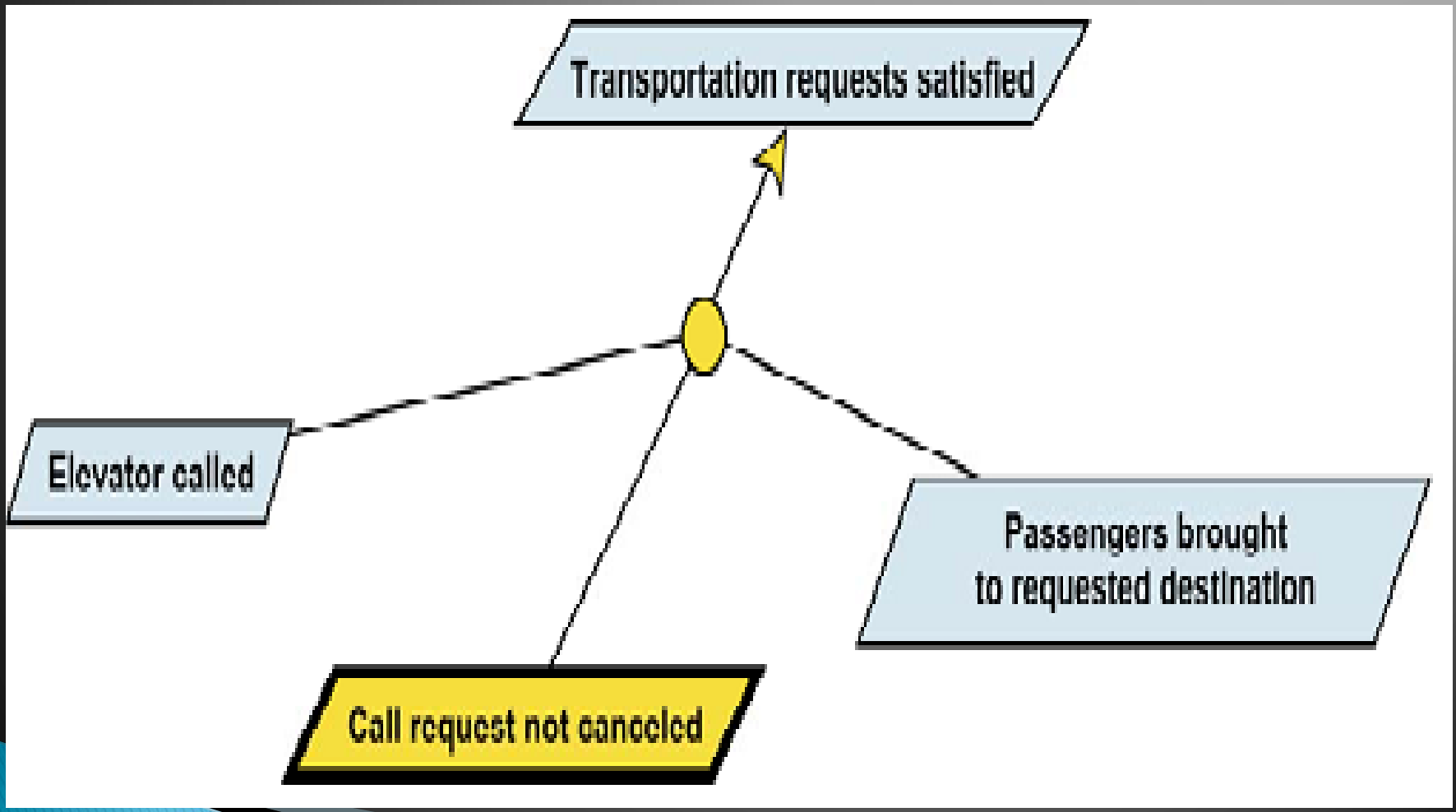


A Note on Terminology

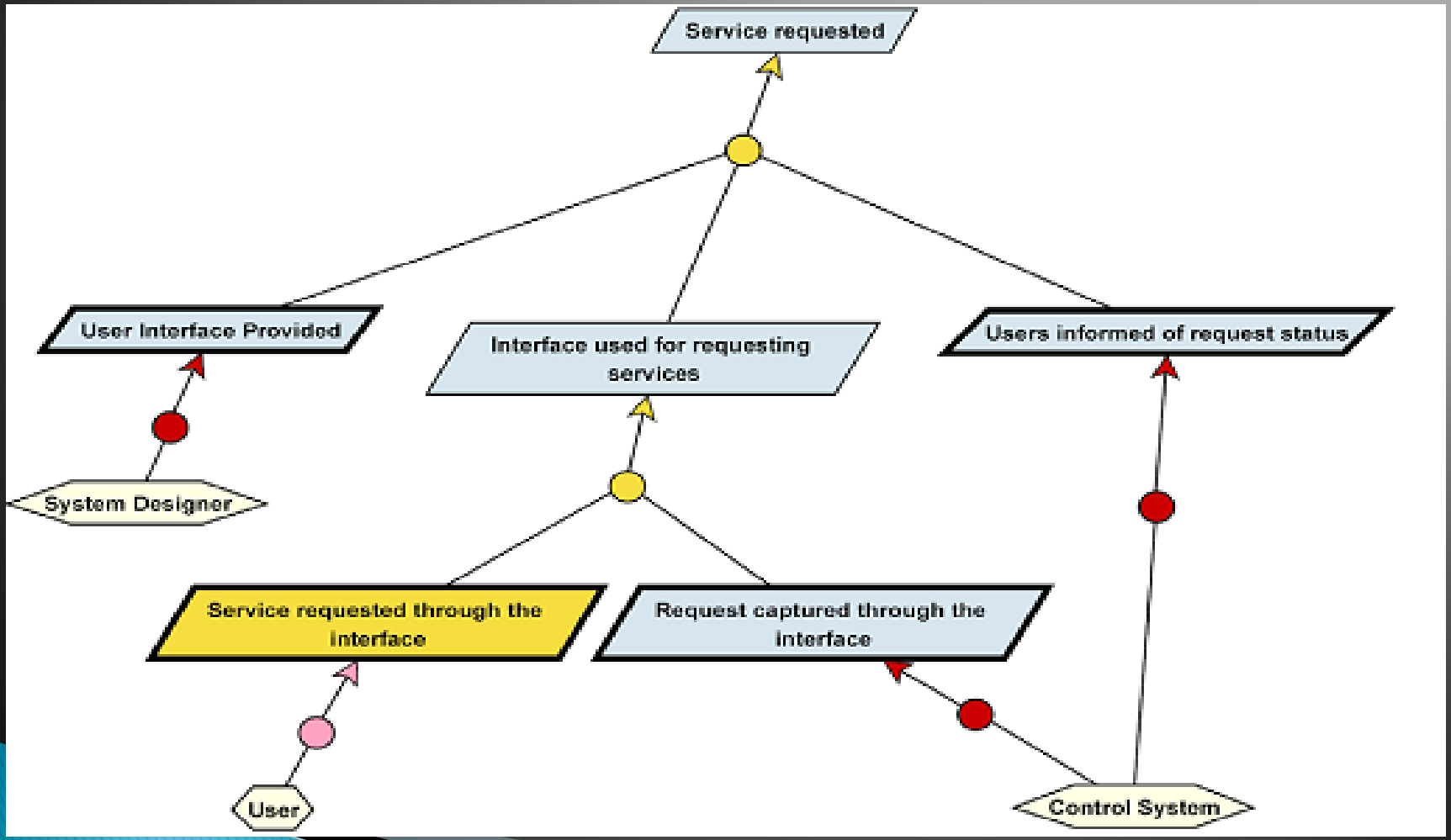
- Look at the previous figure and notice the way goals have been named: a word followed by verb in its passive form. For instance, we have written “Service requested” instead of “Request service” or “The passenger must request the service”.

- 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



Elevator Called: Generic Service Request Pattern

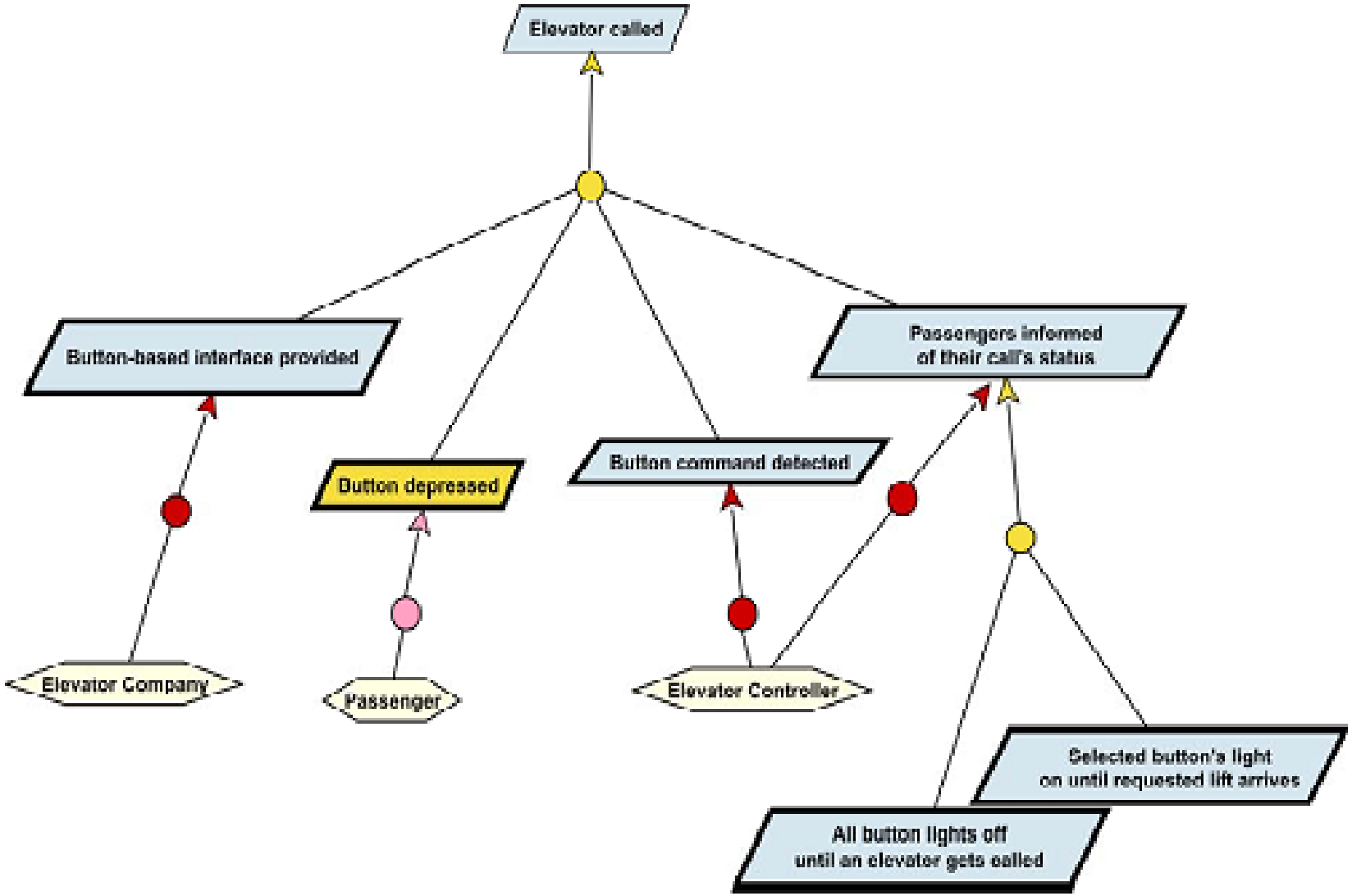


Goal Model Completeness Criteria

- 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



Summary

- KAOS Goal Model
- Terminologies
- Completeness criteria

KAOS Object Model

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.

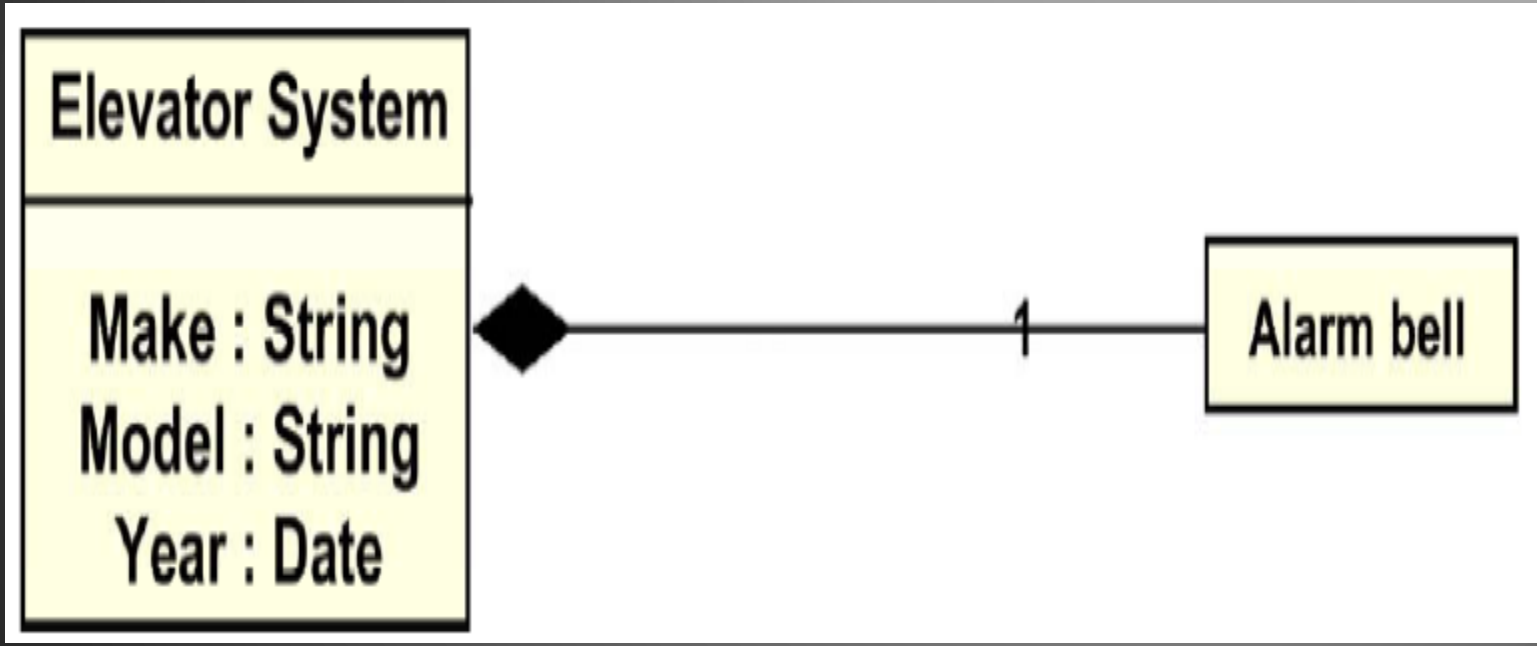
Types of Objects

- 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”).

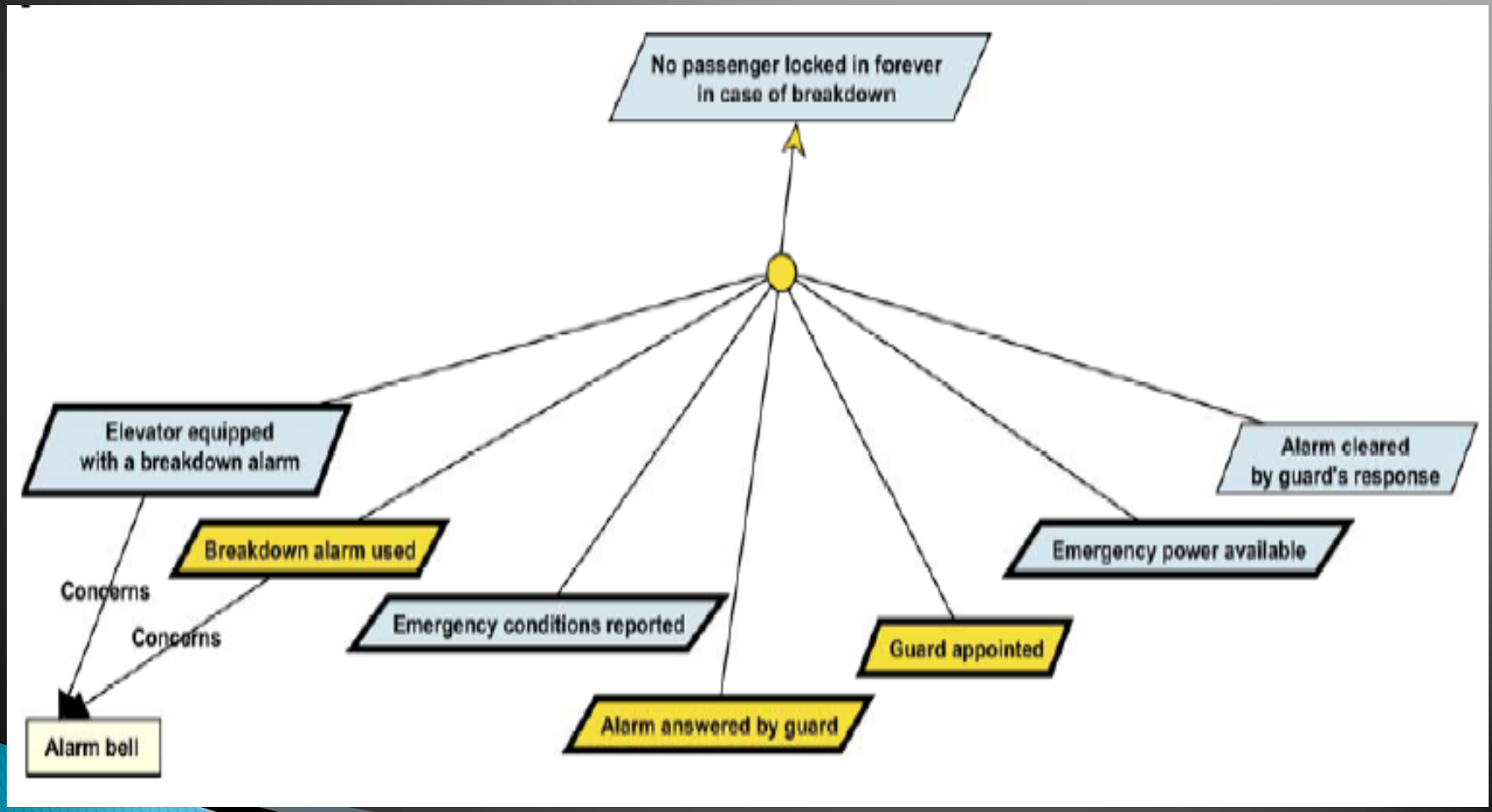
Types of Objects

- 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.

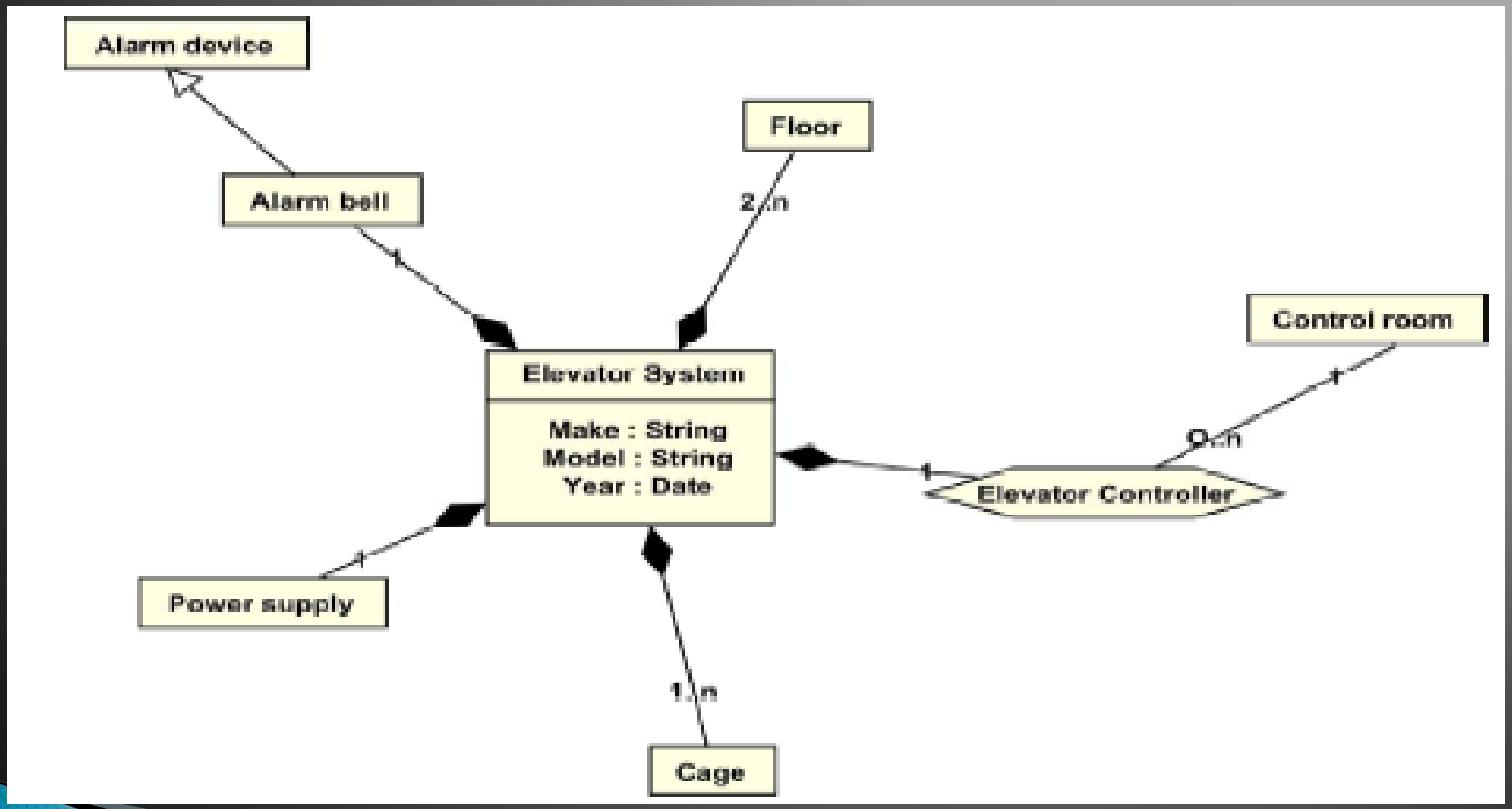
Object Model: Example



“Concerns” Relationship for Identifying Objects in Goal Model



Example: Elevator System



Summary

- KAOS Object Model
- Types of objects
- Example

KAOS Operational Model

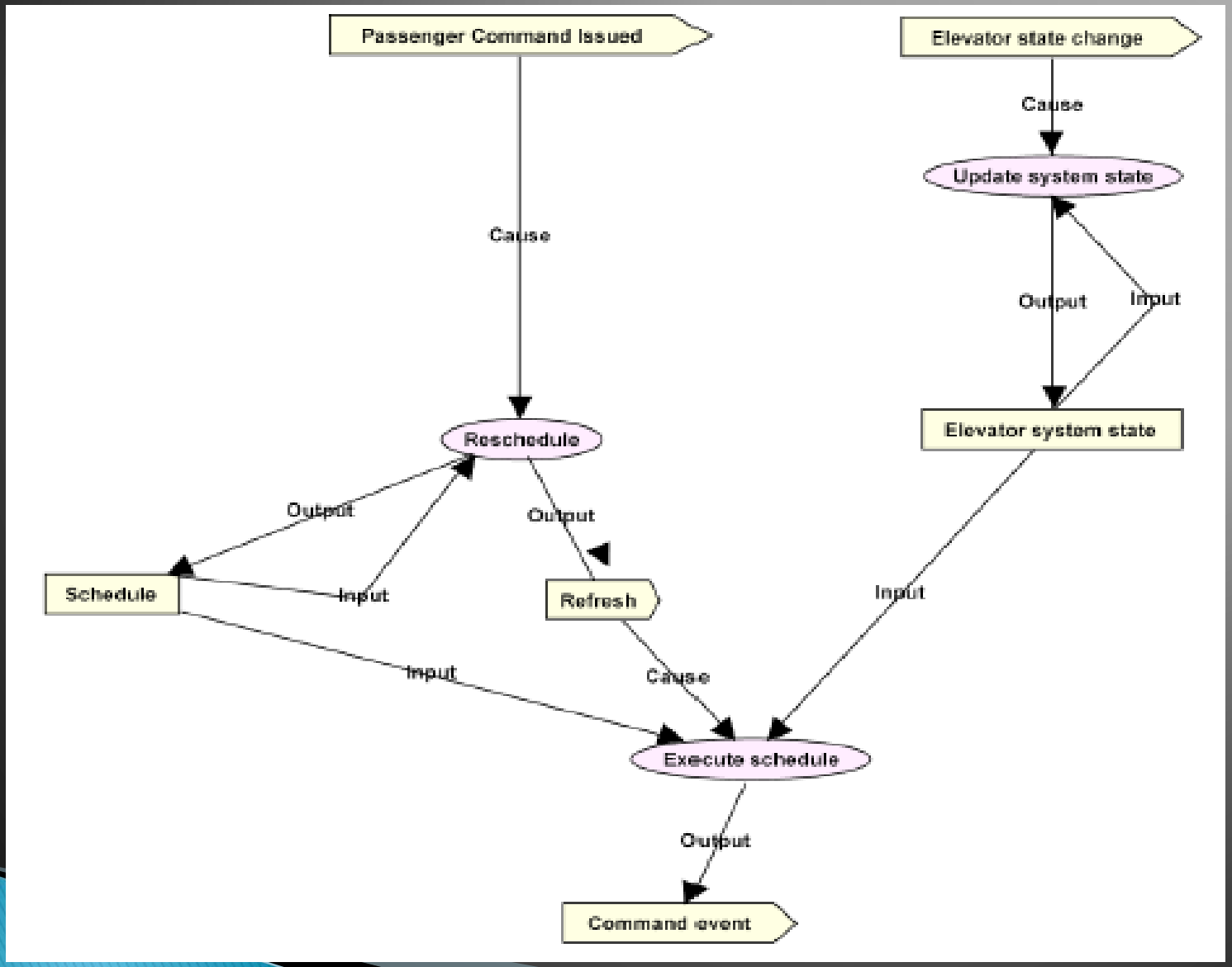
KAOS Operation Model

- 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).

KAOS Operation Model

- Operations can directly be expressed by stakeholders during the interviews.
- Operations can be identified by looking at all the existing requirements.

Operational Model



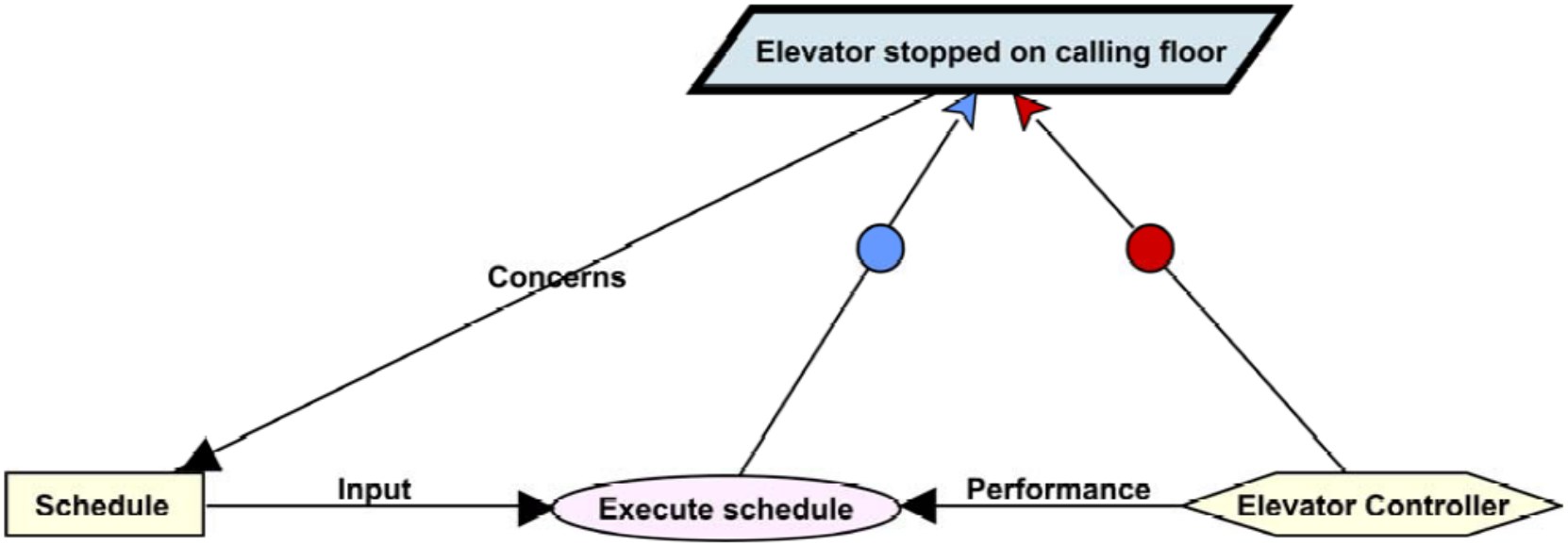
Operational Model

- 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).

Completeness criteria



Summary

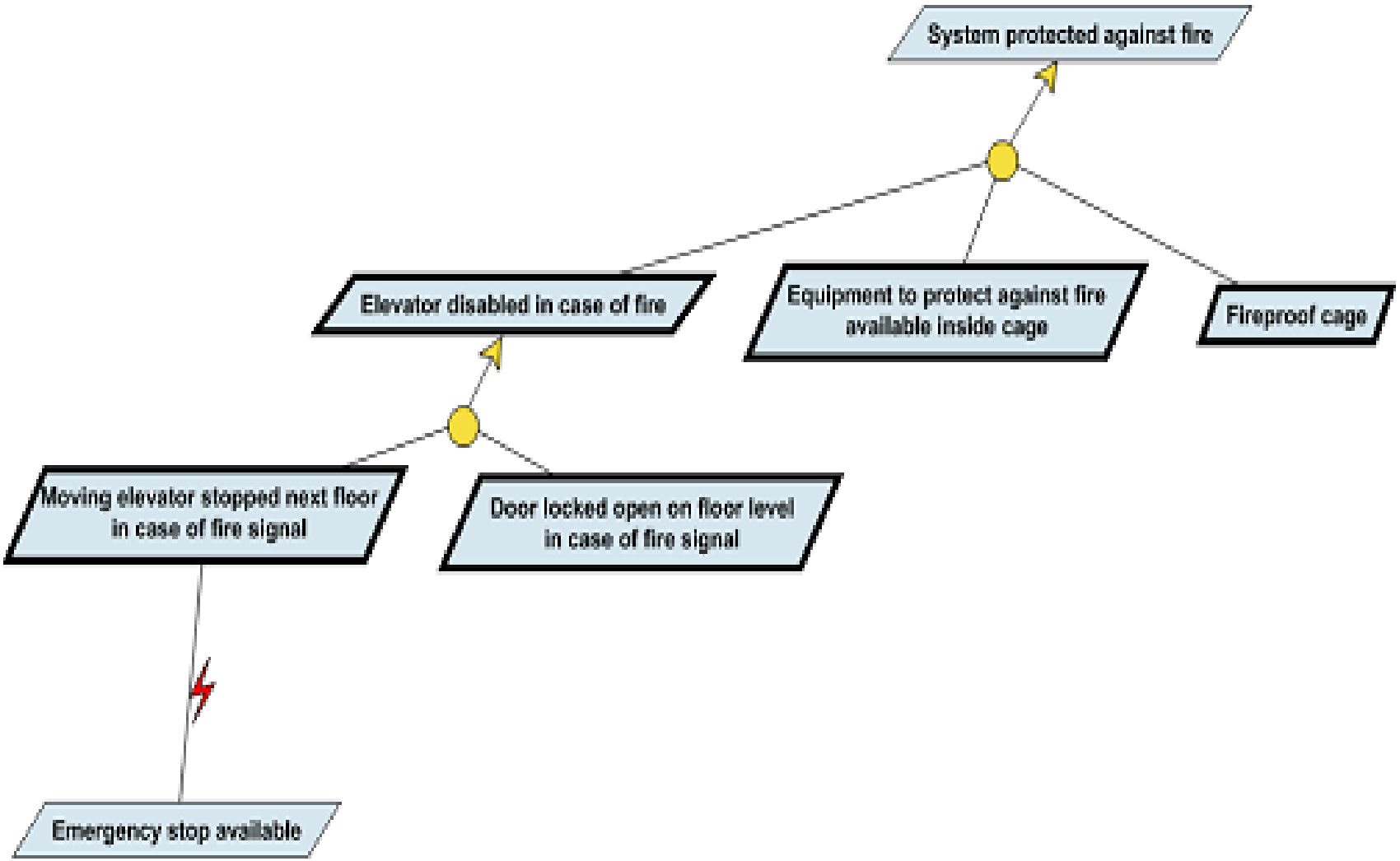
- KAOS Operational model
- Operational model terminology
- Example

KAOS Responsibility Model

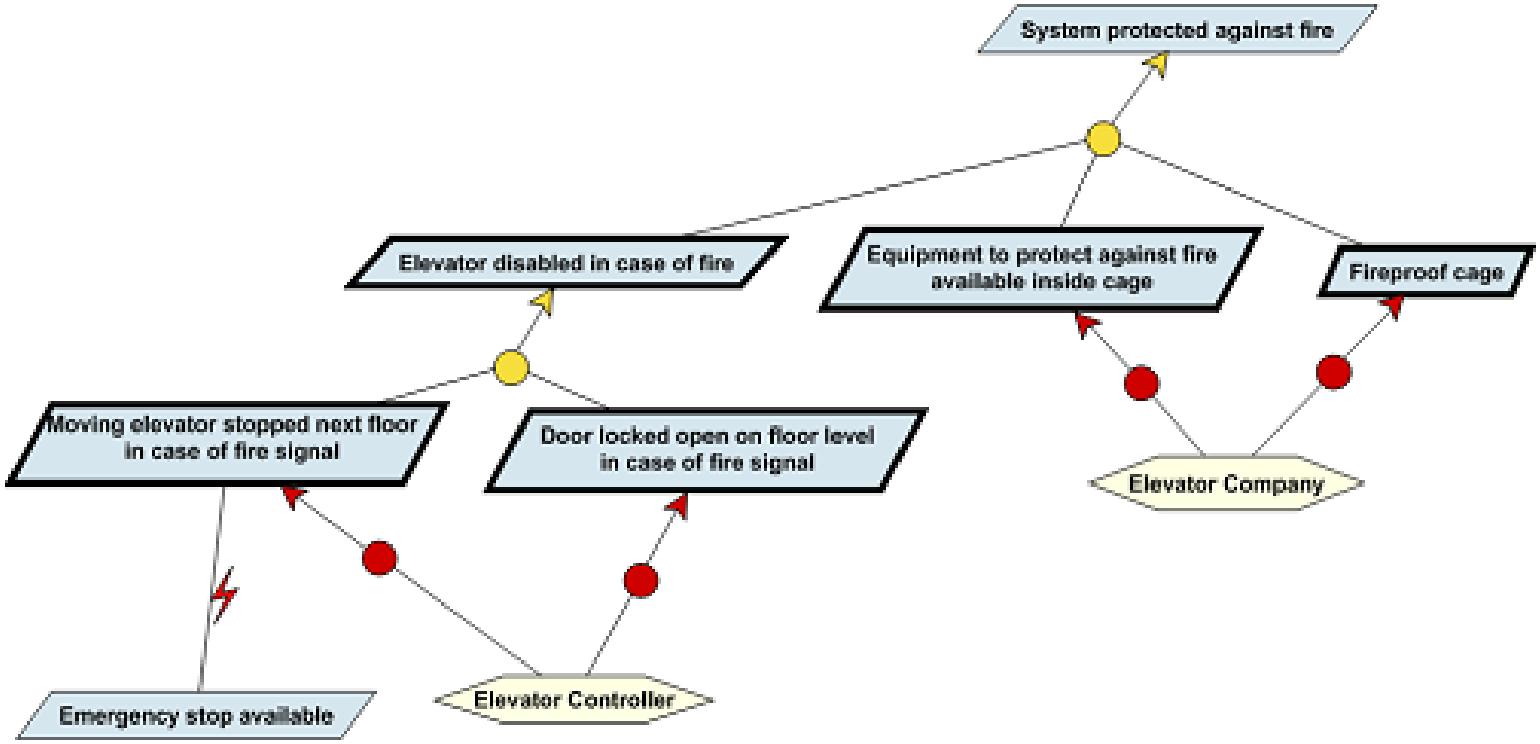
KAOS Responsibility Model

- 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.

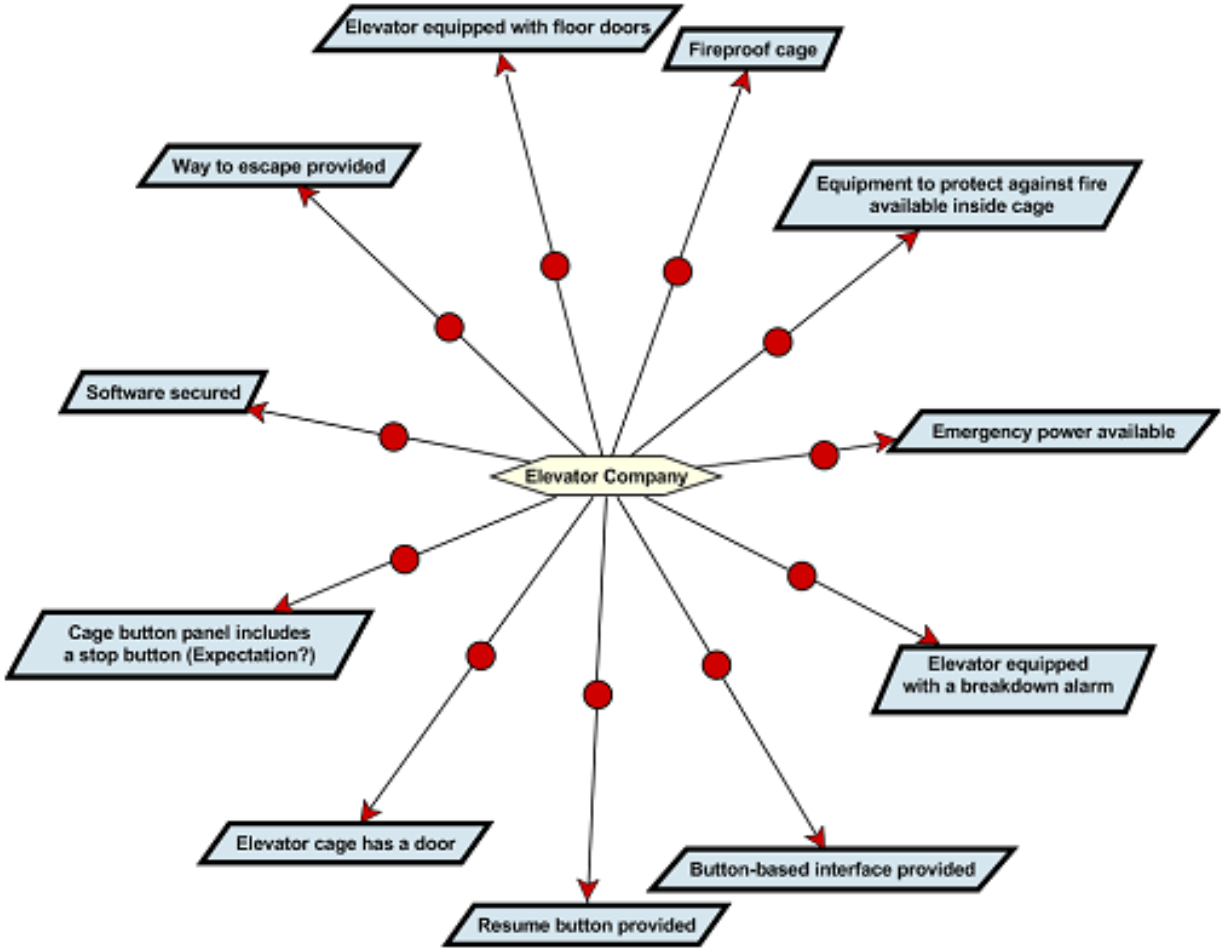
From Goal Diagram to Responsibility Diagram



Intermediate Refinement: (Responsibilities Added for all Requirements)



Responsibilities for the Elevator Company



Responsibilities for the Elevator Controller

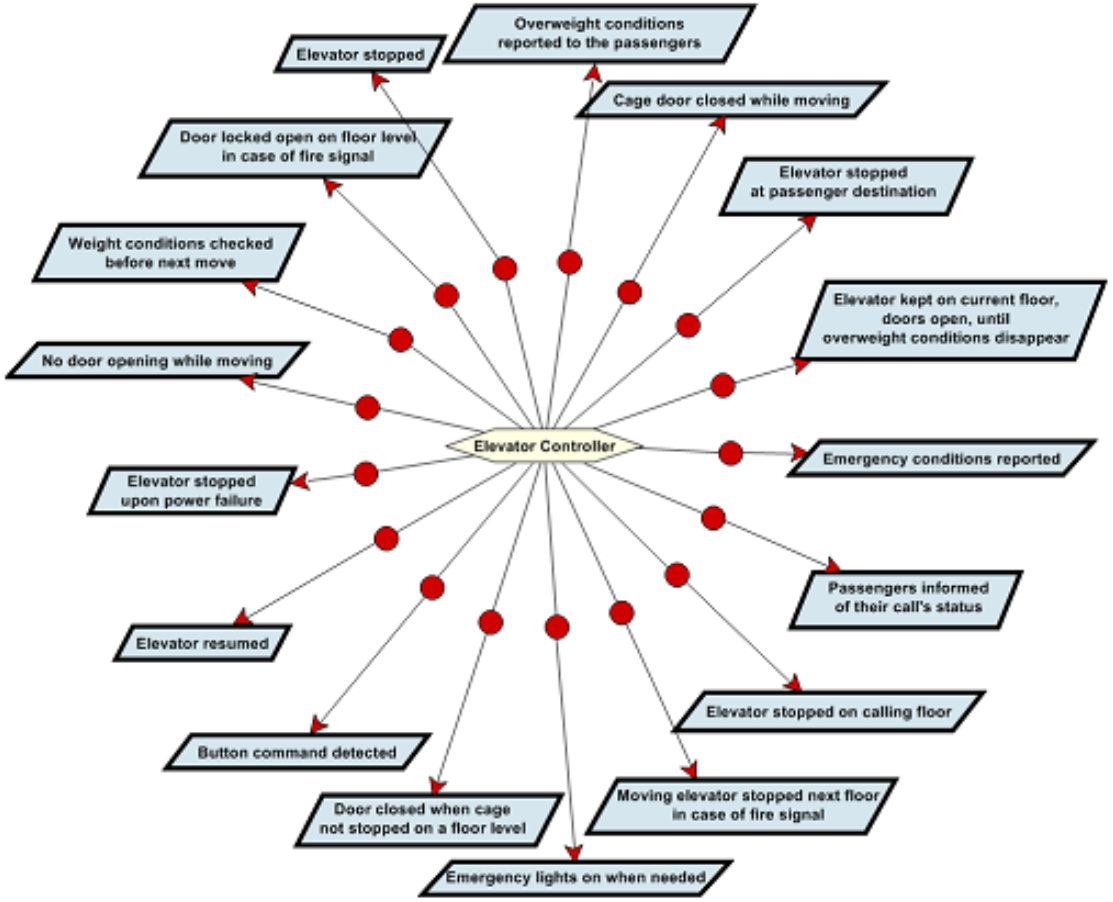


Figure 22. Responsibilities of the Elevator Controller

Summary

- KAOS Responsibility model
- Responsibility model notations

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 thier working
- Rational Unified Process
- Requirment 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

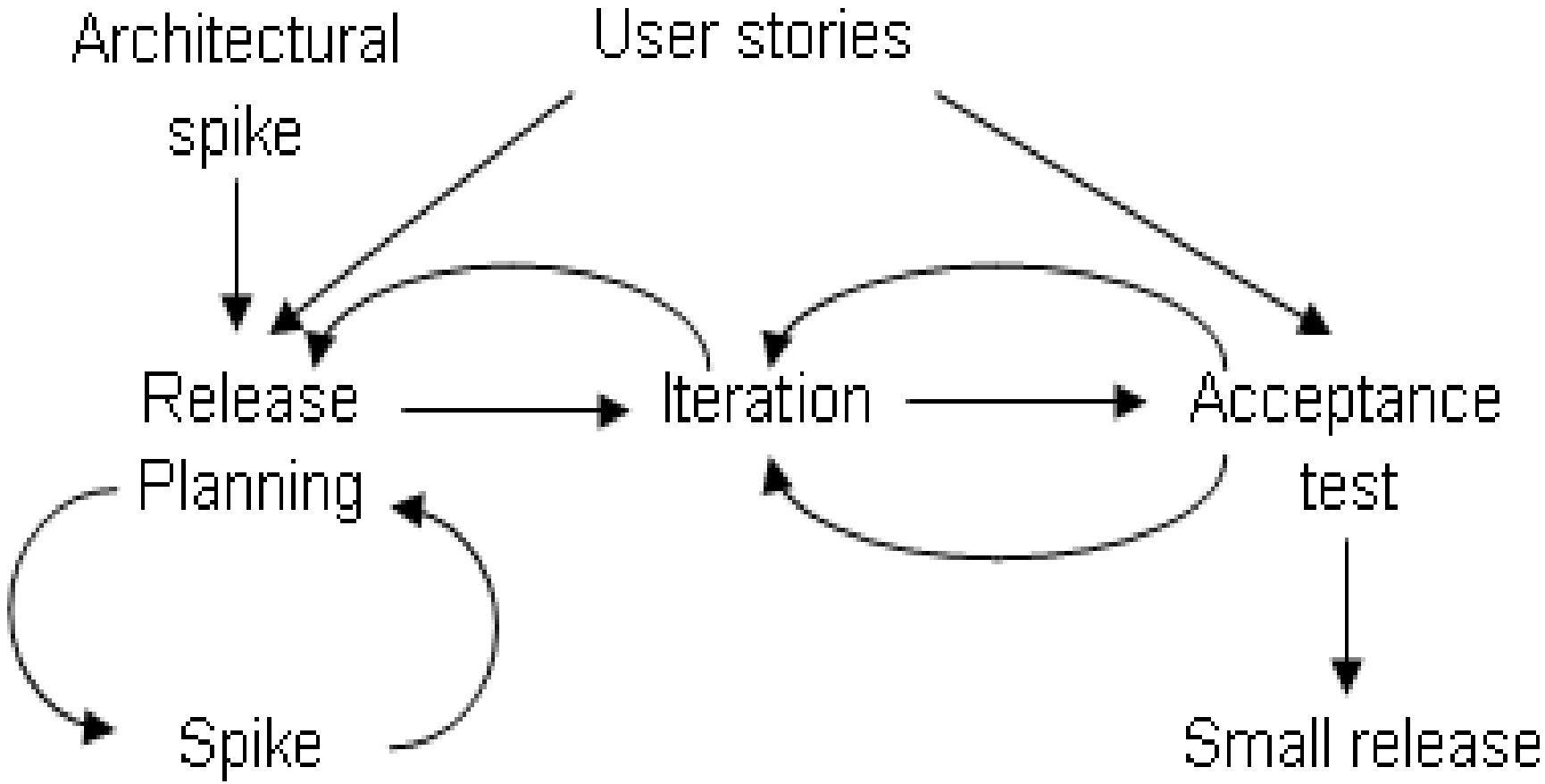
Agile Manifesto

- 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)

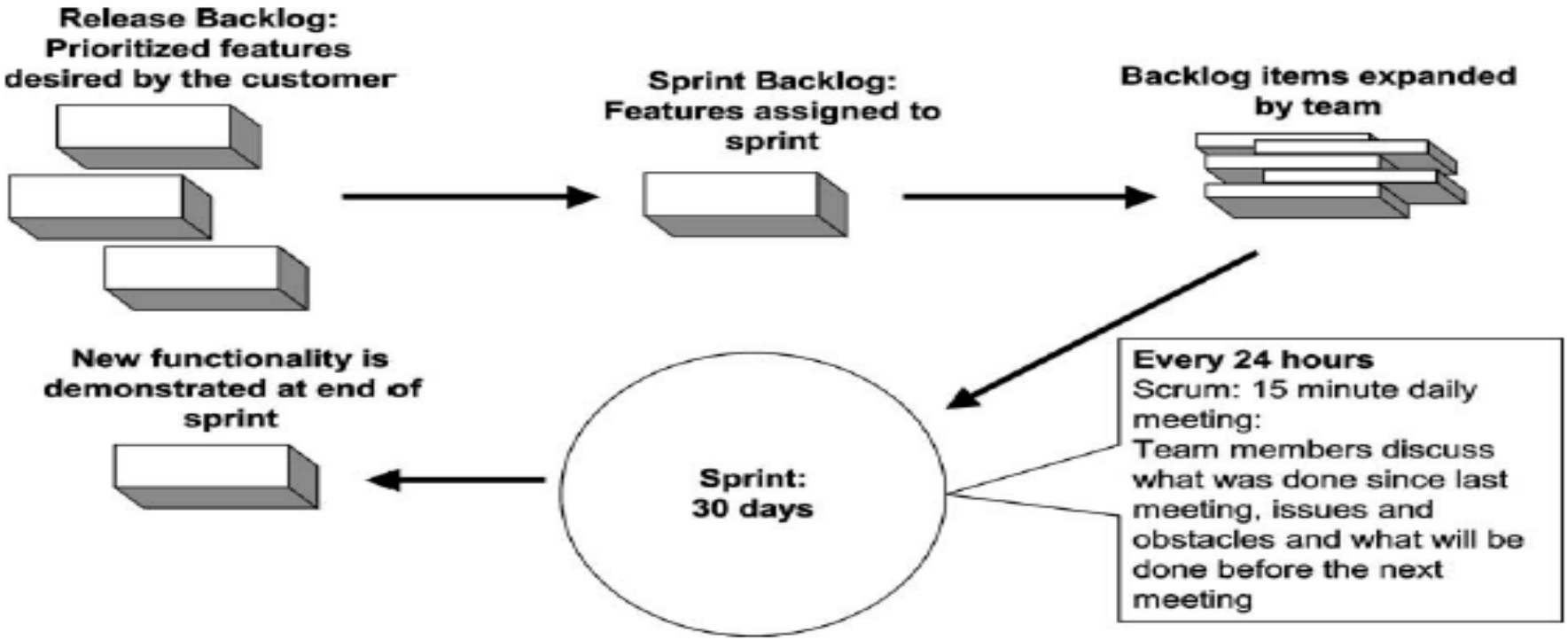
- Extreme Programming is the most famous agile technique. XP use story cards for elicitation. A user story is the description that provides business value to the customer
- These rules are described below:
- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Tests
- Refactoring
- Pair Programming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



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 process

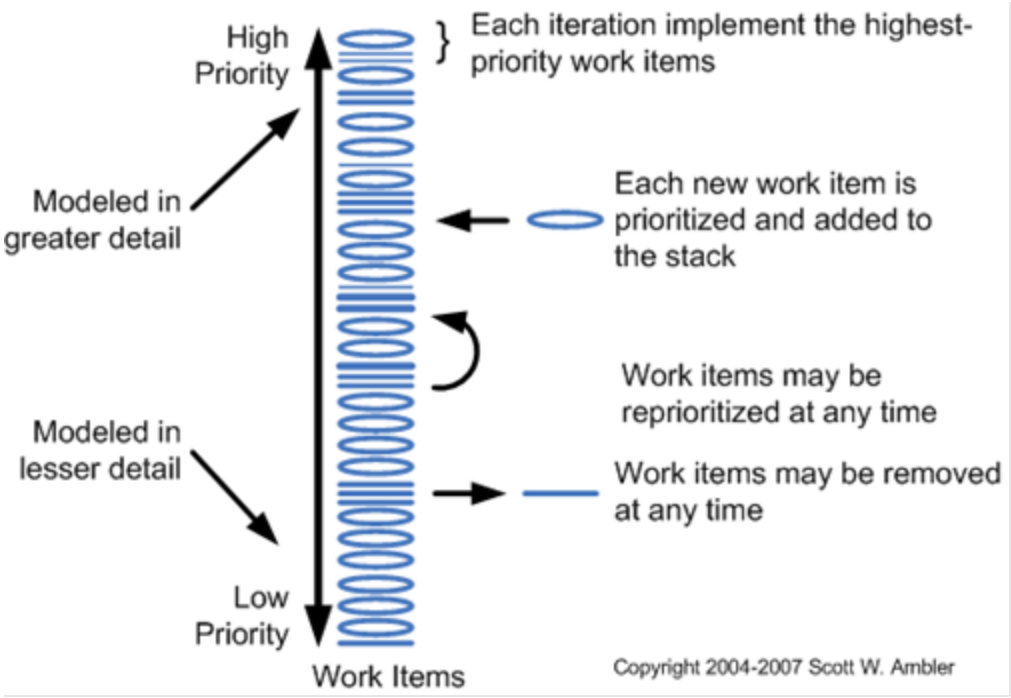
- The agile change management process handles changes in the beginning of each iteration of the development cycle. It may be a sprint in Scrum or iteration in XP and so on. The key stakeholders in change management process are the managers, developing team and off course customers or product owner.

- 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



lifecycle of change management in agile

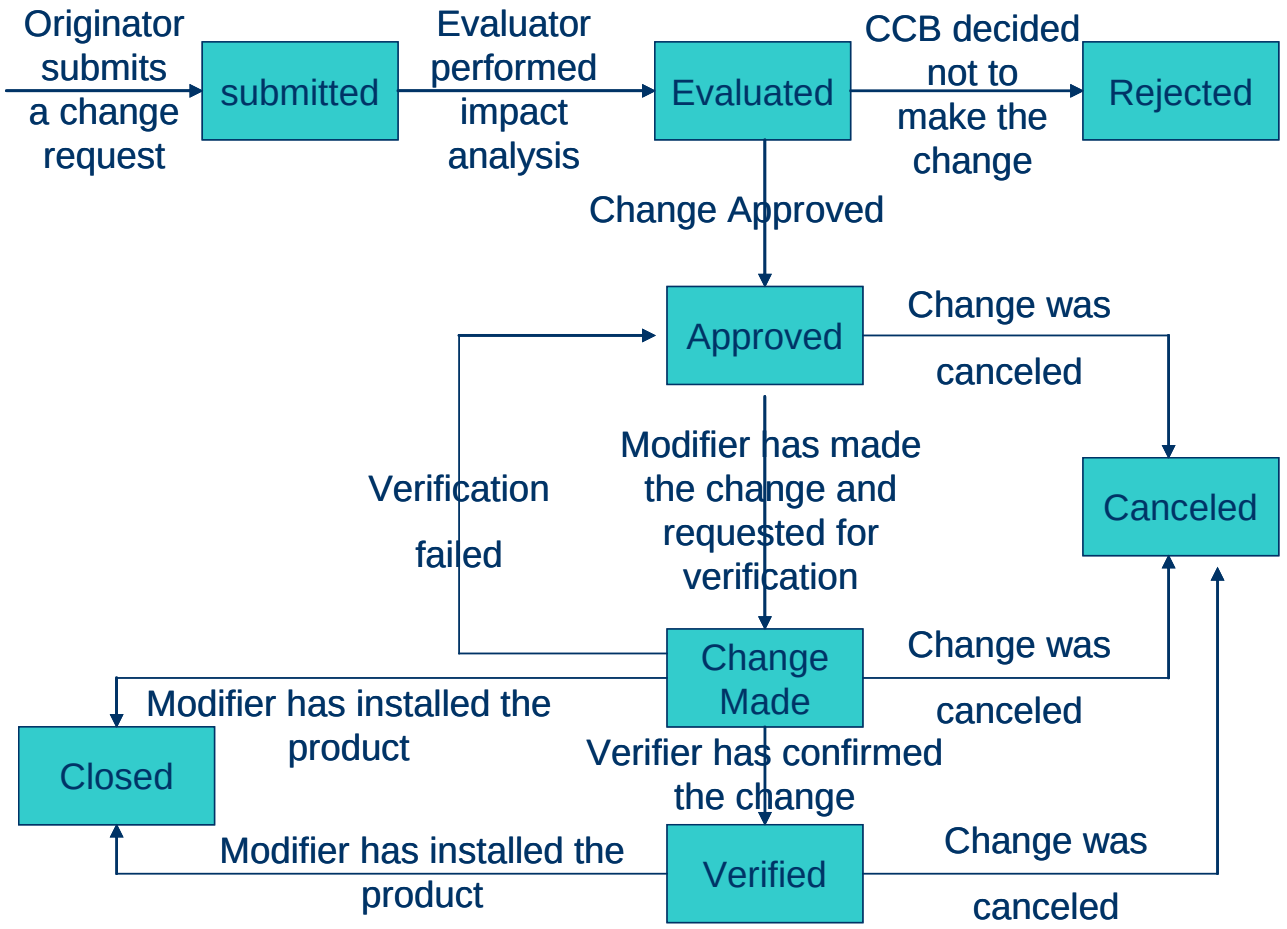
- **Start:** In the start of each iteration, the team takes the highest priority requirement that can be completed in the specified iteration period. This requirement which is now going to be implemented is well understood with the help of customer and other related stakeholders and documents etc. Necessary 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…

Requirement Change Management Process



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.

Freezing the Requirements

- 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.

Value Oriented Prioritization

- 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.

Value Oriented Prioritization

	Business Values (V_1 ... V_n)					Risks (R_1 ... 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	
r_1	5	4	10	9	2	8	5	154
r_2	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

Software Requirement and Specification

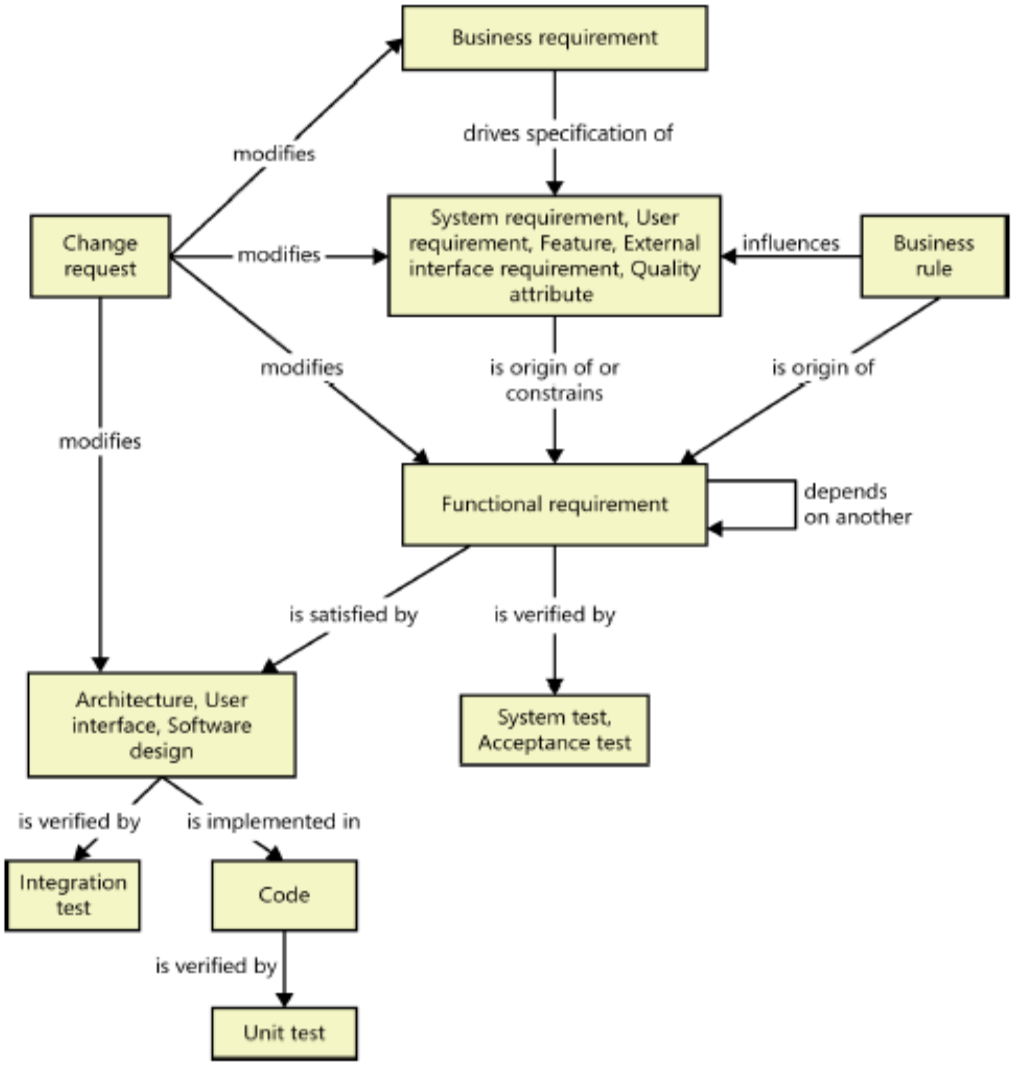
Introduction

- 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.


Business requirement
drives specification of
modifies
Change request
modifies
System requirement, User requirement, Feature, External interface requirement, Quality attribute
influences
Business rule
modifies
is origin of or constrains
is origin of
modifies
Functional requirement
depends on another
is satisfied by
is verified by
Architecture, User interface, Software design
System test, Acceptance test
is verified by
is implemented in
Integration test
Code
is verified by
Unit test

Classification of Requirement Tracibility

- Backward-from traceability

Links requirements to their sources i.e documents or people

Links requirements to design and implementation components

- Forward-from traceability

Classification of Requirement Traceability

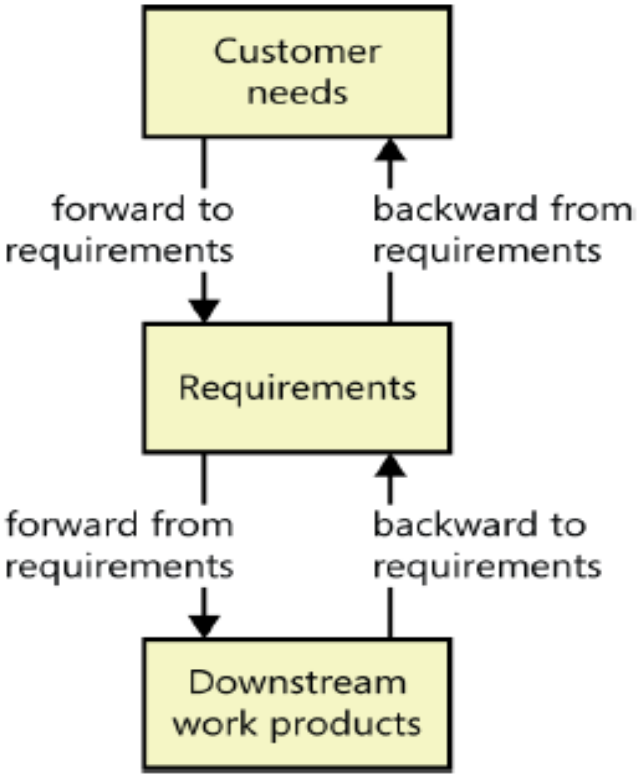
- Backward-to traceability

Links design and implementation components back to requirements

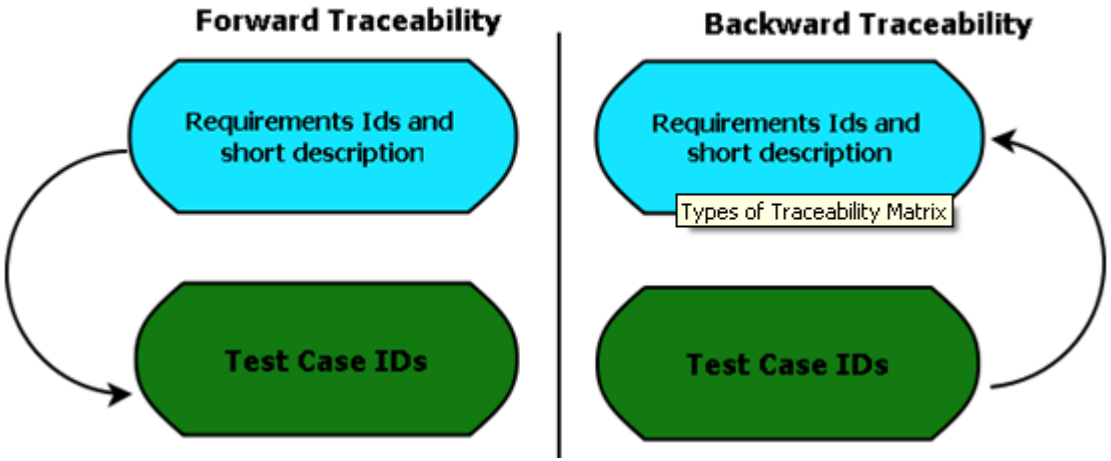
- Forward-to traceability

Links requirements back to their sources

Classification of RT



Classification of RT



Categories of requirement traceability

- Requirements-sources traceability

Links the requirement and their sources which specified the requirement

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

- Requirements-rationale traceability

- Requirements-requirements traceability

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

Categories [illegible]ent traceability

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

- Requirements-architecture traceabi

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

- Requirements-design traceability

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

- Requirements-interface traceability

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)

Traceability Matrix

- 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

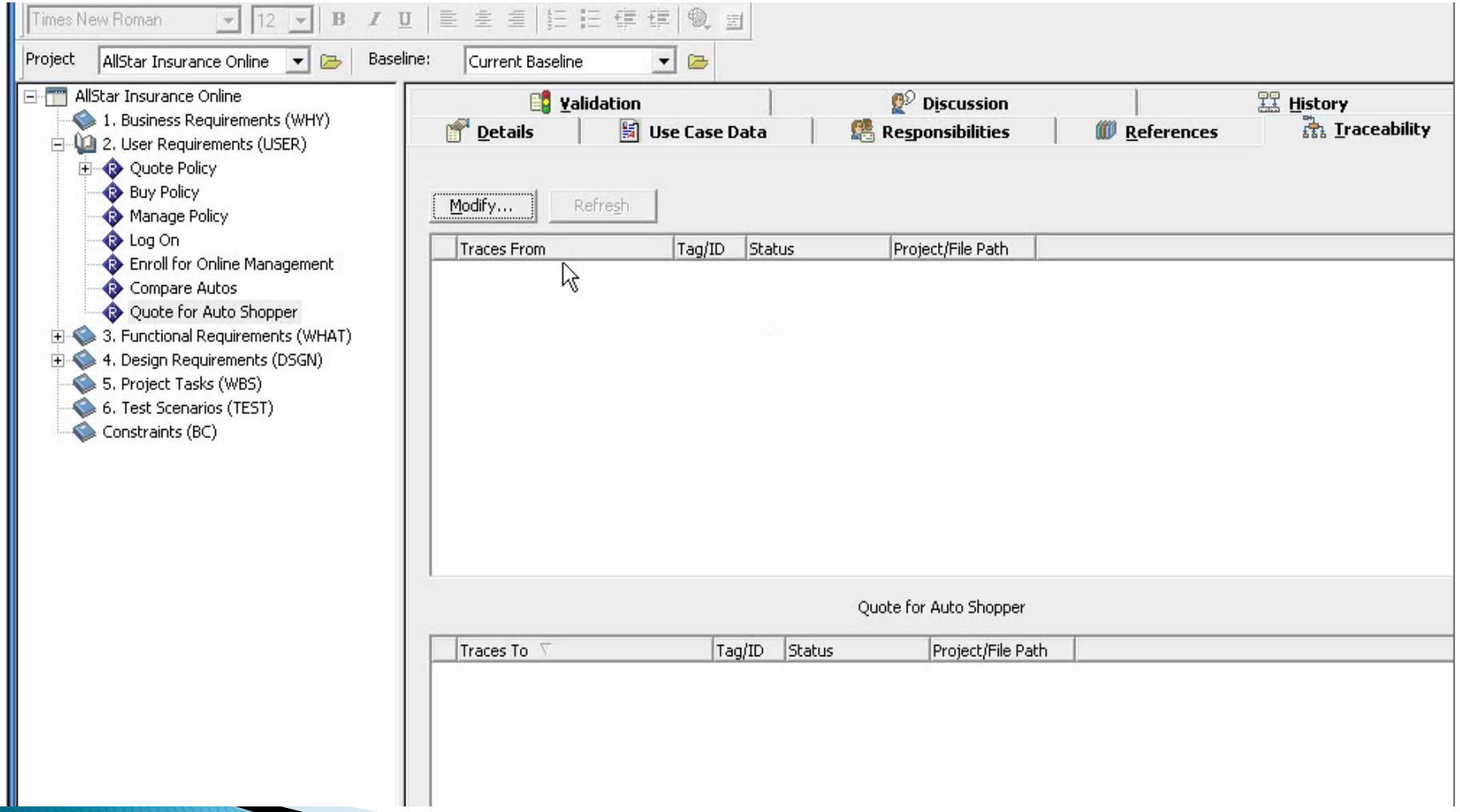
Requirement traceability Tools

- 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

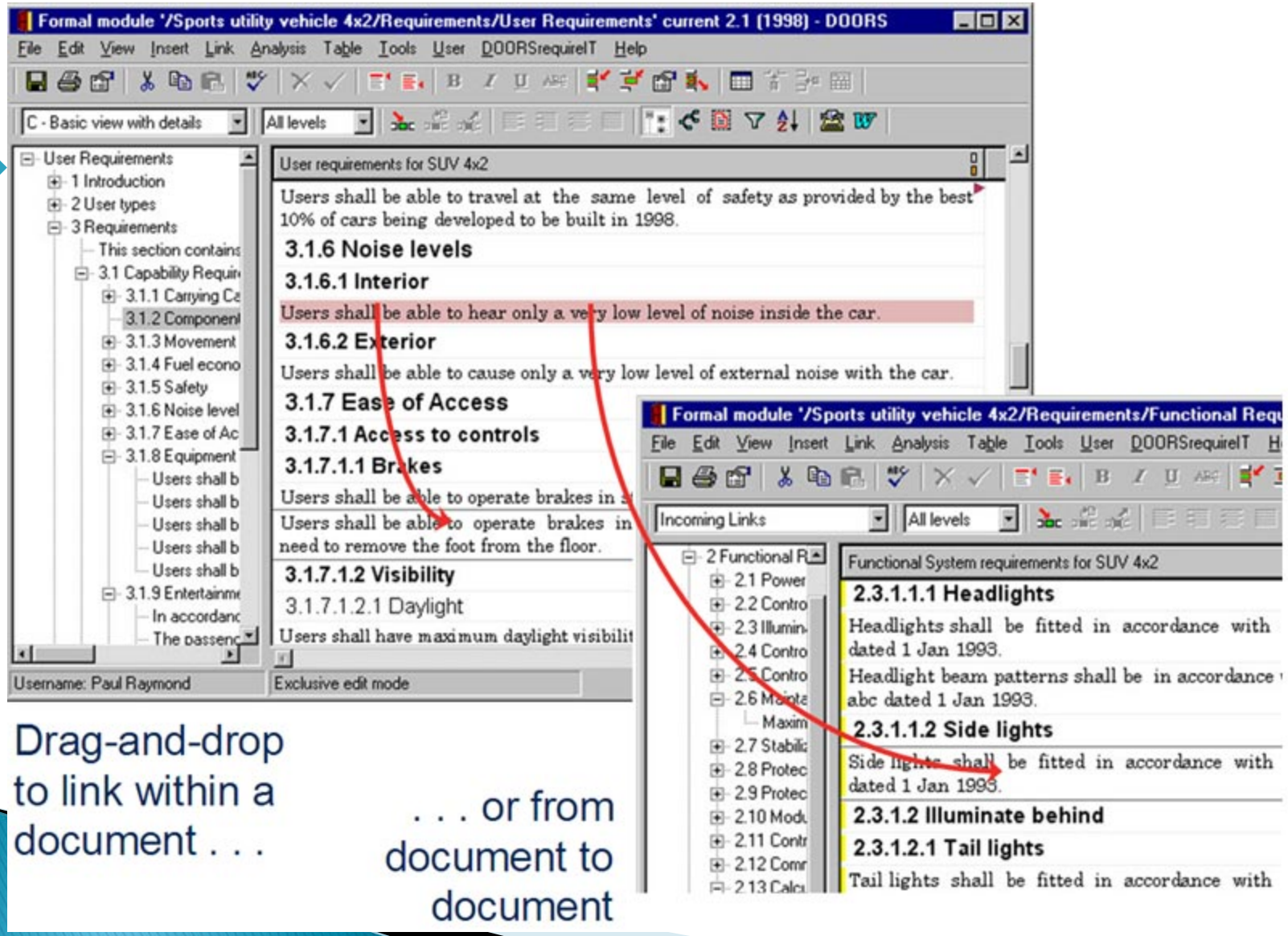
Caliber-RM


Times New Roman
12
Project
AllStar Insurance Online
Baseline:
Current Baseline
AllStar Insurance Online
1. Business Requirements (WHY)
2. User Requirements (USER)
Quote Policy
Buy Policy
Manage Policy
Log On
Enroll for Online Management
Compare Autos
Quote for Auto Shopper
3. Functional Requirements (WHAT)
4. Design Requirements (DSGN)
5. Project Tasks (WBS)
6. Test Scenarios (TEST)
Constraints (BC)
Validation
Discussion
History
Details
Use Case Data
Responsibilities
References
Traceability
Modify...
Refresh
Traces From
Tag/ID
Status
Project/File Path
Quote for Auto Shopper
Traces To
Tag/ID
Status
Project/File Path

Rational Dynamic Object Oriented Requirements System(DOORS)

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

DOORS



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

Software Requirement and Specification

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

Requirement document

- 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

Requirement document

- 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

Parts of IEEE/ANSI Standard 830-1993

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

Parts of IEEE/ANSI Standard 830-1993

- 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

- General discussion on IEEE Standard
  - 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.

cyclomatic complexity

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