Chapter 8. Requirements

8.1 Introduction

- The goal of the requirements phase is to elicit the requirements from the user. This is usually achieved by the development of diagrams and the requirement specification after discussions with the user.
- The user then reviews the diagrams and specification to determine if the software developer has understood the requirements. Thus, it is essential that the diagrams and specifications communicate back to the user the essential aspects required of the software to be produced.

8.1 Introduction

- Requirement Engineering
  - The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
  - The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

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

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

Types of Requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.
- Domain requirements
  - Constraints on the system from the domain of operation

Functional Requirements

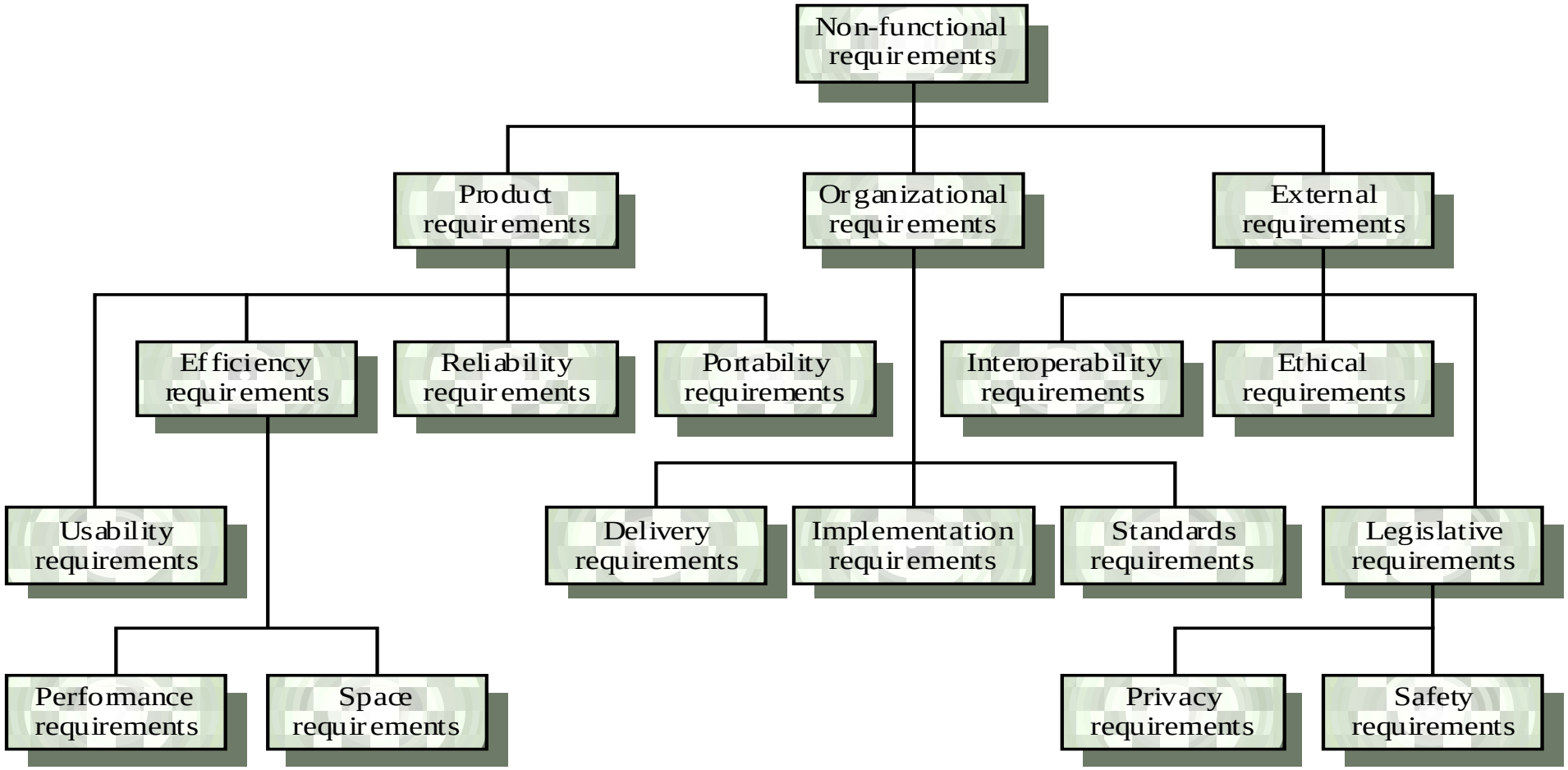
- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

- (Examples)
  - A user shall be able to search the appointments lists for all clinics.
  - The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
  - Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Non-functional Requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements.
- Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE(Integrated Development Environment), programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of Non-functional Requirements



Classification of Non-functional Requirements

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
  - (Example) The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
  - (Example) Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.
  - (Example) The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Domain Requirements

- The system’s operational domain imposes requirements on the system.
  - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
- Complete
  - They should include descriptions of all facilities required.
- Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, it is impossible to produce a complete and consistent requirements document.

Requirement Engineering Processes

- The processes used for requirement engineering vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, requirement engineering is an iterative activity in which these processes are interleaved.

Requirements elicitation and analysis

- Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*
- Stages include:
  - Requirements discovery
    - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
    - Formal or informal interviews with stakeholders are part of most requirement engineering processes
  - Requirements classification and organisation
    - Groups related requirements and organises them into coherent clusters.
  - Prioritisation and negotiation
    - Prioritising requirements and resolving requirements conflicts.
  - Requirements specification
    - Requirements are documented.

Requirements Validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high, so validation is very important.
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

- Requirements checking
  - Validity : Does the system provide the functions which best support the customer's needs?
  - Consistency : Are there any requirements conflicts?
  - Completeness : Are all functions required by the customer included?
  - Realism : Can the requirements be implemented given available budget and technology
  - Verifiability : Can the requirements be checked?

Requirements Validation

- Requirements validation techniques
  - Requirements reviews
    - Systematic manual analysis of the requirements.
    - Regular reviews should be held while the requirements definition is being formulated.
    - Both client and contractor staff should be involved in reviews.
    - Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.
  - Prototyping
    - Using an executable model of the system to check requirements.
  - Test-case generation
    - Developing tests for requirements to check testability.

Requirements management

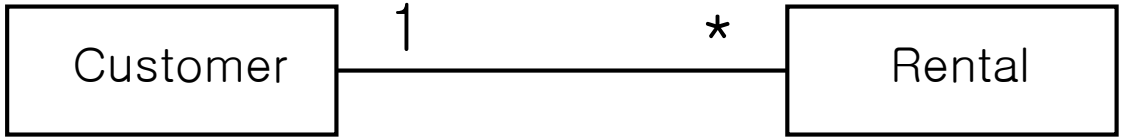
- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

8.2 Object Model

- The basic approach in an object-oriented (OO) methodology is to develop an object model (see Section 2.4) that describes that subset of the real world that is the problem domain.
- The purpose is modeling the problem domain and not designing an implementation. Thus, entities that are essential to understanding the problem will be included even if they are not going to be included in the solution. The attributes and methods included in the object model will also be those needed for understanding the problem and not those that will just be important for the solution.
- **The following are rules for object models for requirements:**
  1. All real-world entities that are important to understanding the problem domain must be included.
  2. All methods and attributes that are important to understanding the problem domain must be included.
  3. Objects, attributes, and methods that are only significant for the implementation should not be included.

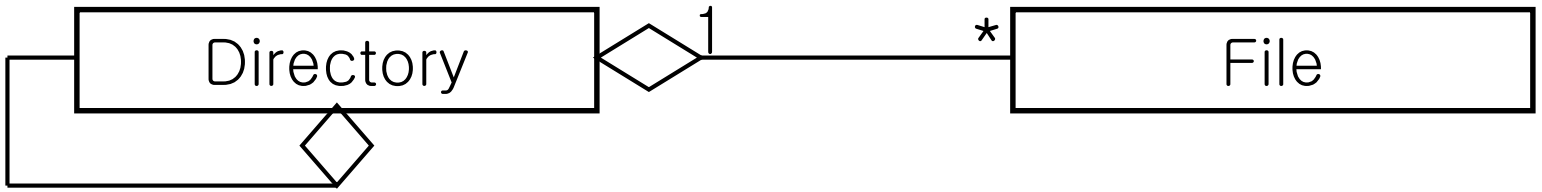
8.2 Object Model

- Association
  - Relationship between classes
  - Represented by a link
- Role
  - Represented at the end of link which representing association (has, exist, rent)
- Multiplicity
  - Represented by number of links
  - * means ‘many’.



8.2 Object Model

- Aggregation
  - Class of system : Directory
  - Class of component : File
  - Usually 1 to n relationship
  - Represented by a diamond



- Generalization
  - Relationship between generalized class and specialized class
  - Common attributes and operations of a child class are inherited from those of parents' class.

8.2 Object Model

EXAMPLE 8.1

Draw an object model for the library problem (see Example 2.6).

See Fig. 8-1.

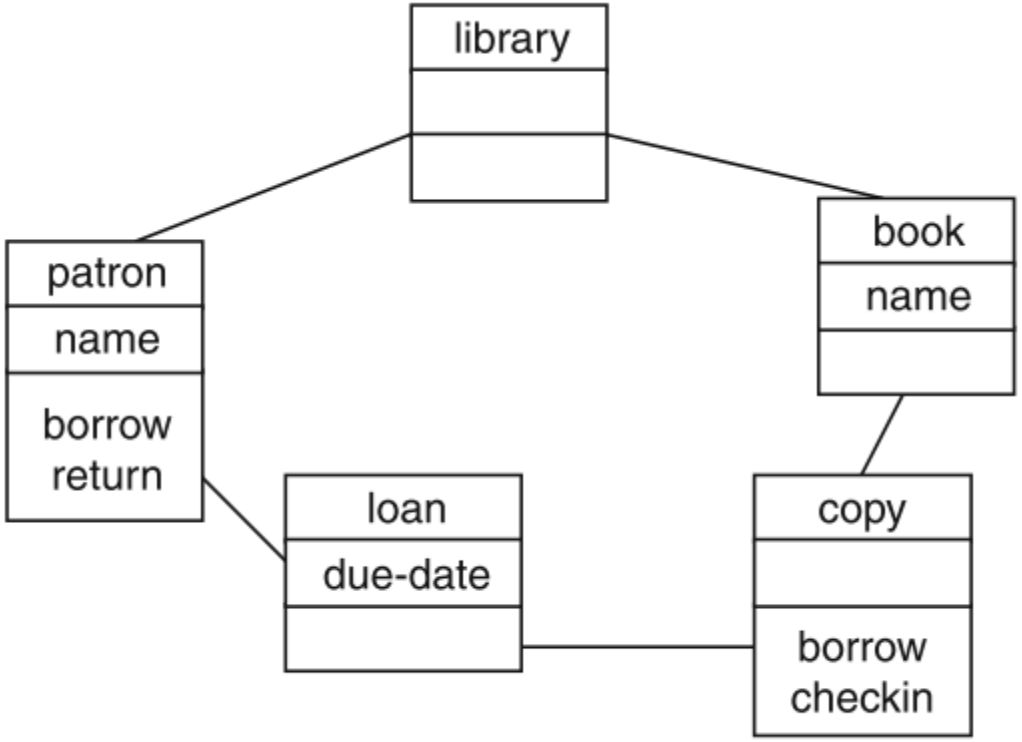
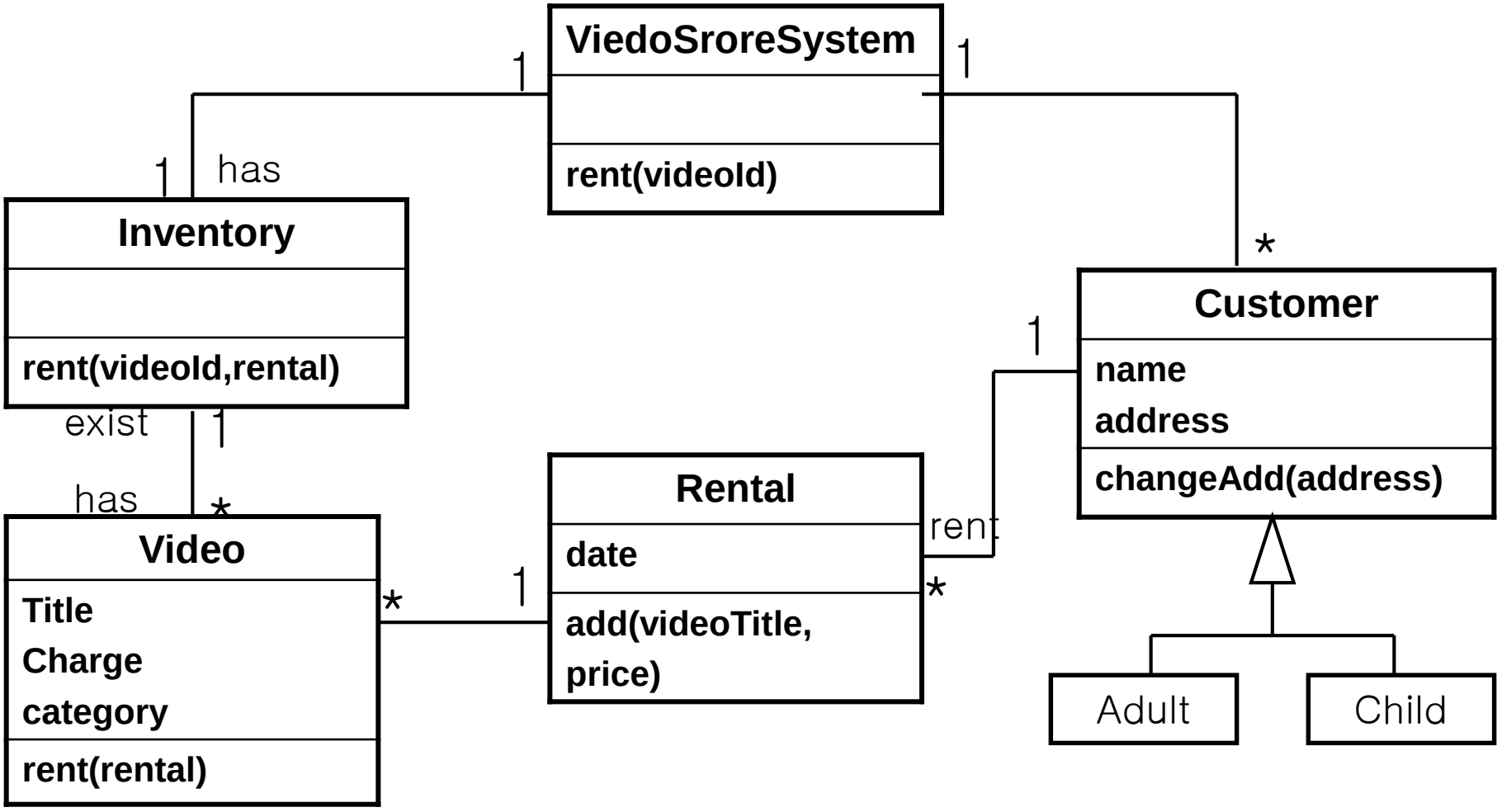


Fig. 8-1. Object model for library problem.

8.2 Object Model



8.3 Data Flow Modeling

- Although not used much in OO development, data flow diagrams (seeSection2.2) were essential parts of pre-OO software development. Data flow diagrams (DFDs) still have an important role in the specification of many systems. The importance of data flow diagrams is in specifying what data is available to a component.
- Knowing the data available often helps in the understanding of what a component is expected to do and how it will accomplish the task.

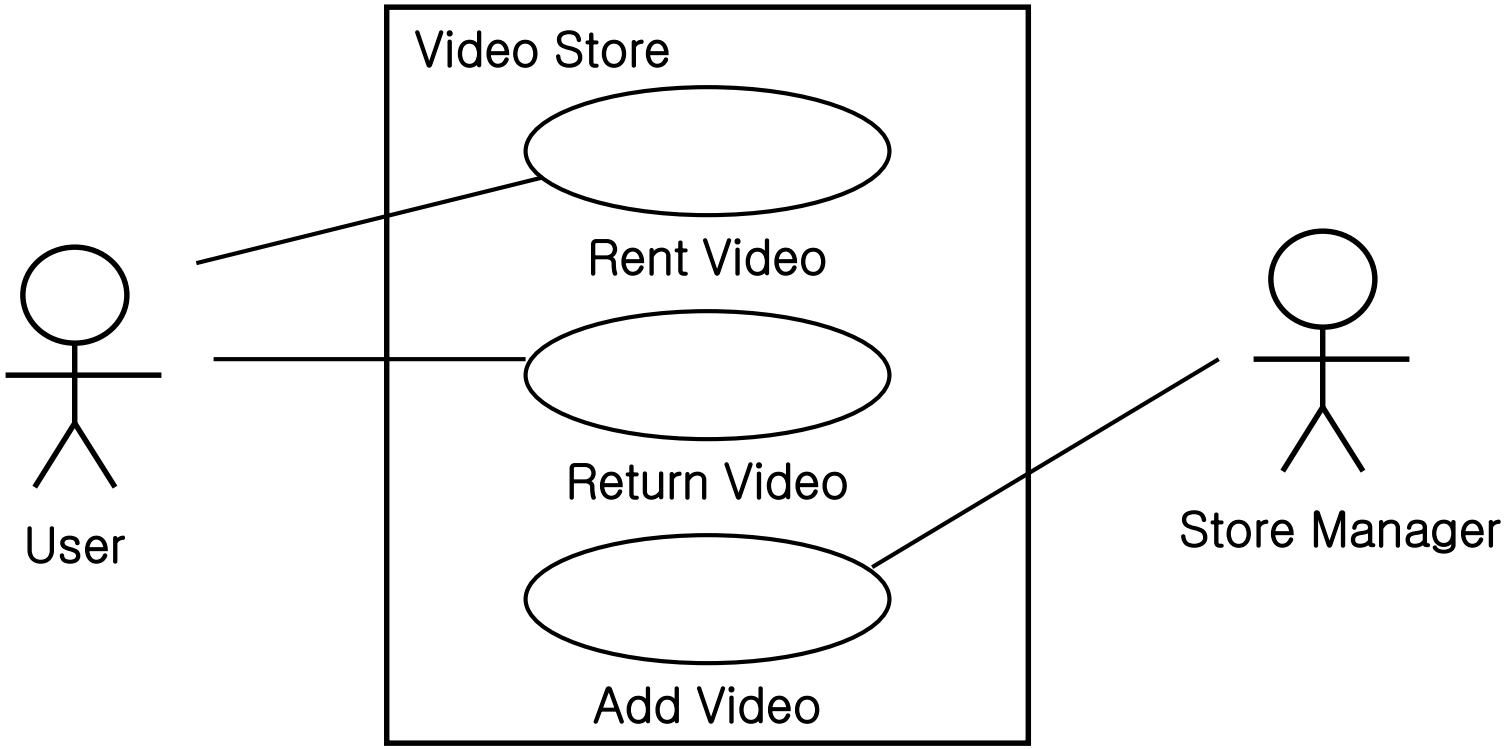
8.4 Behavioral Modeling

- ***Behavioral modeling*** refers to the behavior of the system, usually from the user point of view.
- These diagrams are used to specify aspects of the proposed system. It is important that the diagrams capture the essential aspects of the system and are able to communicate those aspects both to the developer and to the user for confirmation that this is the system that he or she wants.

8.4.1 USE CASE

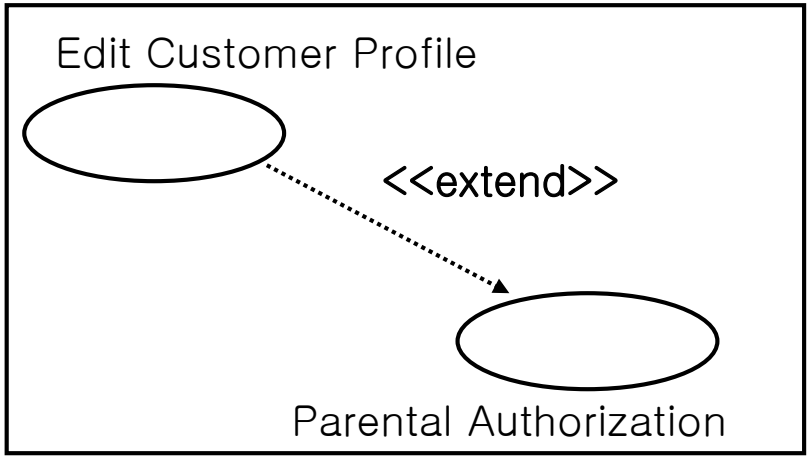
- The use case diagram represents the functionality of the system from the user's point of view (see Section 2.5). All critical functionality must be mentioned. However, routine functions that are implied by a higher-level phrase do not have to be specifically mentioned.

8.4.1 USE CASE

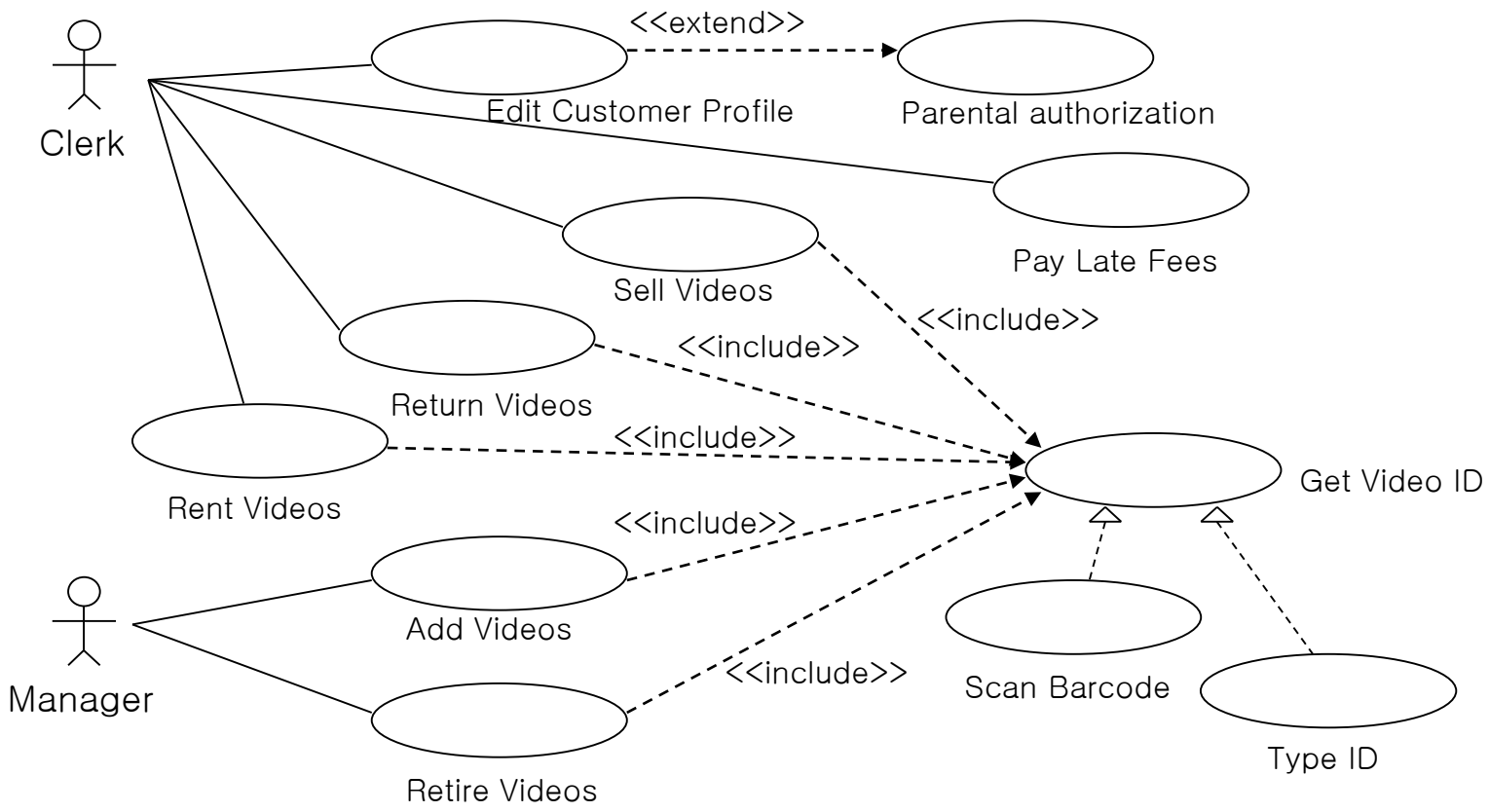

Video Store
Rent Video
Return Video
Add Video
User
Store Manager

8.4.1 USE CASE

- Include
  - Common function
  - Represented by a dotted arrow and <<include>>
- Extend
  - Exceptional condition
  - When a Use Case includes another Use Case
  - Represented by a dotted arrow and <<extend>>



8.4.1 USE CASE


Clerk
<<extend>>
Edit Customer Profile
Parental authorization
Pay Late Fees
Sell Videos
<<include>>
<<include>>
Return Videos
<<include>>
Rent Videos
Get Video ID
<<include>>
Add Videos
Manager
<<include>>
Scan Barcode
Type ID
Retire Videos

8.4.2 SCENARIOS

- A scenario is a sequence of actions that accomplishes a user task. Alternative sequences are only shown by having a separate scenario for each alternative. Scenarios are used to illustrate an important capability or proposed use of the system.
- In UML, an interaction diagram (see Chapter2) is used to specify the scenarios. Scenarios can also be specified by listing the sequence of actions.
- Scenarios are real-life examples of how a system can be used.
- They should include
  - A description of the starting situation;
  - A description of the normal flow of events;
  - A description of what can go wrong;
  - Information about other concurrent activities;
  - A description of the state when the scenario finishes.

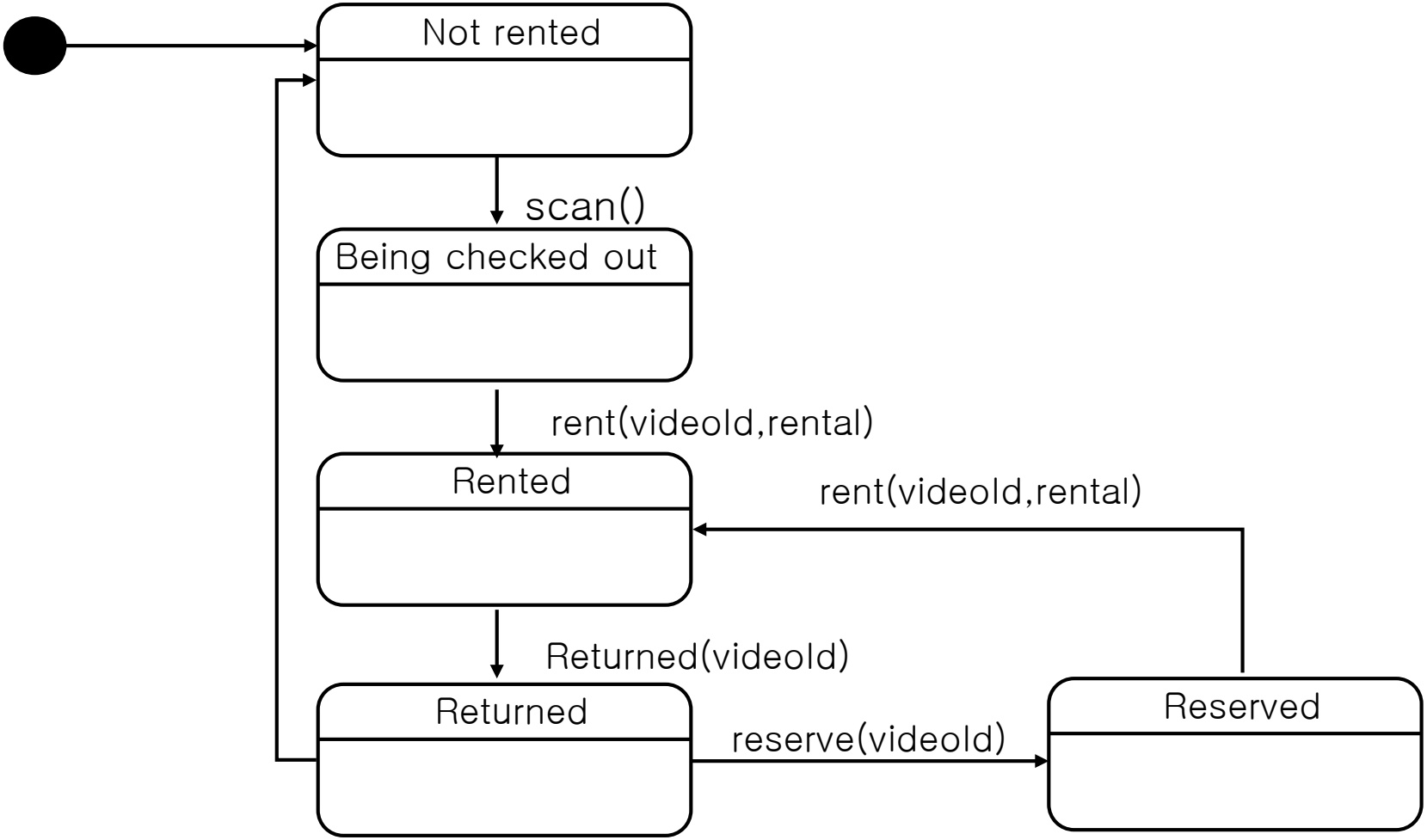
8.4.3 STATE DIAGRAMS

- The details of state diagrams were covered in Chapter 2. When being used as part of the requirements specification, it is important that the states reflect domain conditions that are understandable to the users. States that are only significant to the implementation should be coalesced into domain significant states.
- Additionally, the allowed transitions must include all allowed transitions from the scenarios. Sequences that are not intended in the proposed system should not be allowed in the state diagram. This may be difficult, since the existence of a transition in a scenario does not prohibit other transitions.

The following are rules for using state diagrams in requirement specifications:

1. All states must be domain significant.
2. All sequences from scenarios must be allowed.
3. All prohibited scenarios must not be allowed.

8.4.3 STATE DIAGRAMS



8.7 IEEE Standard for Software Requirements Specification

- The following SRS outline is based on IEEE 830-1993:

1. Introduction—This section is intended to provide an overview of the rest of the specification.

1.1 Purpose—This section must describe the purpose of the SRS and the intended audience.

1.2 Scope—This section must identify the product, explain what the product will and will not do, and describe the application of the software, including benefits, objectives, and goals.

1.3 Definitions—This section must identify all terms, acronyms, and abbreviations used in the specification.

1.4 References—This section must identify all documents referenced elsewhere in the specification.

1.5 Overview—This section must describe what the rest of document contains.

8.7 IEEE Standard for Software Requirements Specification

2. Overall Description—This section is intended to provide the background to understand the rest of the requirements.

2.1 Product Perspective—This section must put the product into perspectivewithotherproducts.Itwillusuallyincludeablockdiagramofthe larger system. It should specify constraints (e.g., system interfaces with other software), user interfaces (e.g., screen formats, timing), hardware interfaces, software interfaces (e.g., versions of interfaced software), memory constraints, operations (e.g., modes of operations), and site adaptation constraints.

2.2 Product Functions—This section must include a summary of the major functions of the product.

2.3 User Characteristics—This section must include the educational level, experience, and technical expertise of the users.

2.4 Constraints—This section must include any other (e.g., regulatory) constraint that is not covered in Section 2.1.

2.5 Assumptions and Dependencies— This section must include any assumptions that, if not true, would require changes to the requirements.

2.6 Apportioning of Requirements—This section must identify requirements that may be delayed to future versions of the product.

8.7 IEEE Standard for Software Requirements Specification

3. Specific Requirements—According to IEEE Standard 830: ‘‘This section of the SRS should contain all the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. ’’ This is an important criteria to remember: The SRS should be sufficiently detailed so designs and tests can be constructed directly from the SRS. Also, according to IEEE Standard 830: ‘‘These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output.’’

3.1 External Interface Requirements—This section must describe all inputs and outputs of the system. This is detailing the information from Section 2.1

3.2 Functions—This section must describe all the functions of the system. These must include validity checks on inputs, responses to abnormal situations, effect of parameters, and the relationship of outputs to inputs.

3.3 Performance Requirements—This section must describe the static and dynamic requirements.

3.4 Design Constraints—This section must describe any constraints on the design.

8.8 Contents for SRS

1. Problem Description

Problem : Software for which requirements will be

Problem description is the document for scenarios with which software will be developed and operated.

In this scenario, functions of software and constraints under which software will be developed are described by a lot of sentences.

Example of problem description

Scenario Example: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, ***reports the emergency*** from her car.
- Alice enters the address of the building, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. She confirms her input and waits for an acknowledgment.
- John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

8.8 Contents for SRS

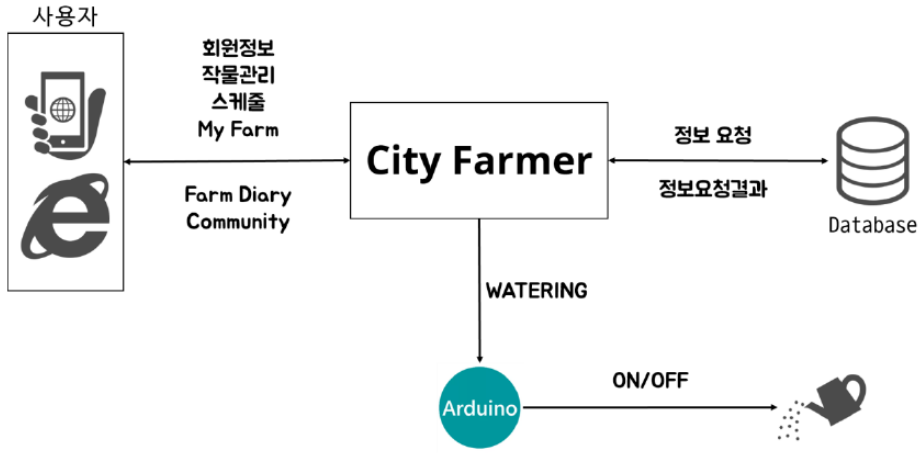
2. System Environment

2.1 System Configuration : Overall description for the environment under which software will be developed and operated.

Diagram for system configuration and explanation for diagram will be included.

Users, Database, Devices, Subsystems, Equipments and so on are shown.

Example of system configuration



[그림 3 Context Diagram]

[그림 3]은 City Farmer의 Context Diagram이다.
사용자는 본 시스템을 실질적으로 사용하는 도시 농부다. 사용자는 City Farmer를 통해 콘텐츠를 제공 받는다. Database는 정보를 요청하고 요청 결과를 받을 수 있다. 웹어플리케이션 City Farmer의 Web Controller를 통해 스프링클러가 작동하며, 텃밭에 물 주는 기능을 제공한다.

8.8 Contents for SRS

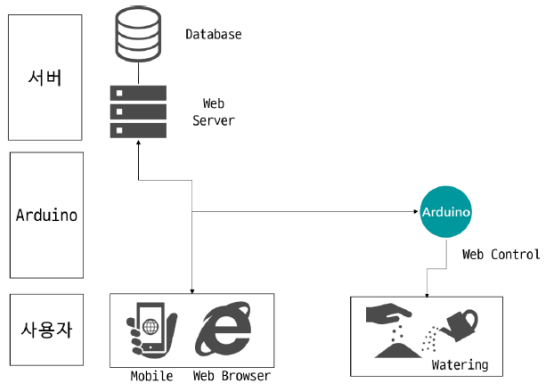
2. System Environment

2.2 Hardware and Network Configuration : Overall description for the hardware environment under which software will be developed and operated.

Diagram for Hardware and Network configuration and explanation for diagram will be included.

This is needed for identifying the hardware constraints.

I ıration



[그림 4 하드웨어 구성도]

Client	Server	IoT
·Computer i5-8250U 1.6GHz(3.4GHz) 8GB / DDR4 / M.2 / 256GB	·Application Server CPU : AMD Athlon(tm) 64 Processor 3200+ Memory : SDRAM ECC 1GB HDD : Segate Barracuda 7200.7 120 GB 7200RPM Ethernet : 100Mbps Ethernet ·Database Server CPU : AMD Athlon(tm) 64 Processor 3200+ Memory : SDRAM ECC 1GB HDD : Segate Barracuda 7200.7 120 GB 7200RPM Ethernet : 100Mbps Ethernet	·Arduino UNO Ethernet Relay Shield Sensor Shield BACOENG 3/4" Stainless Steel Valve

[표3 하드웨어 구성표]

8.8 Contents for SRS

2. System Environment

2.3 Software Configuration : Overall description for the system software environment under which software will be developed and operated.

System software means programming language, operating system, framework, NMS, DBMS, and so on.

Diagram for system software configuration and explanation for diagram will be included.

This is needed for identifying the software constraints.

Example of Hardware and Network Configuration

Client	Server
Javascript Boot Strap HTML/CSS Spring mobile	Server OS : Linux Kernel 2.6.x Framework : Spring Framework Test Server : Tomcat 8.0 DB : Oracle SQL / My Batis

소프트웨어를 구성하는 방식은 세리정보기술의 요구사항에 맞춰 Spring framework, My Batis, Oracle Database로 구성했다.

8.8 Contents for SRS

3***. Functional Requirement Specification***

From the problem description, functional requirements are derived.

Name of requirement, group name of requirement, brief description for requirement, requirement ID will be given to each requirement.

Example of functional requirement specification

Req. Number	**FR-01**
Req. Name	Log in
Req. Category	Functional requirements - account
Req. Description	- The user should be able to access the IdeaSquare application by entering his login information - Students get access with Student-ID and password - Investors get access with username and password - A successful login will redirect the user to the main-page - If the login is not successful, the user will be notified

Req. Number	**FR-02**
Req. Name	Log off
Req. Category	Functional requirements - account
Req. Description	- The user should be able to logout from the IdeaSquare application - A successful login will redirect the user to the login-page

8.8 Contents for SRS

4. ***Nonfunctional Requirement Specification***

From the problem description and system environment, nonfunctional requirements are derived.

Name of requirement, group name of requirement, brief description for requirement, requirement ID will be given to each requirement.

Domain requirements may be specified if necessary.

Example of nonfunctional requirement specification.

Product requirements

Req. Type	Req. Name	Req. Number	Req. Specification
Efficiency	Response time	NF-PE-01	- Every response time should be less than 3 seconds on stable internet condition
	Maximum user capacity in real time	NF-PE-02	- Maximum number of users that system enables to manage stably in real time is less than 5,000 users
	DB management	NF-PE-03	- All user's data stored in the DB immediately and back-up data should be generated as well - Unused Idea data more than a year old should be deleted with together with the respective back-up data

8.8 Contents for SRS

5. ***Scenarios***

For each functional requirement, detailed scenario will be described.

Name of requirement, requirement ID, related actors, initial condition, final condition, exeptional case, flow of action will be specified.

Flow of action should be described step by step.

Example of scenarios:

Scenario Title	**User registration**
Actors	Investor, Administrator
Initial condition	Investor has no account
Final condition	Potential investors get investor account
Exceptional case	- Potential investor is rejected by the administrator - Potential investor is notified
Process	1. Register page is opened 2. Potential investor answers form 3. Potential investor submits form 4. Potential investor is checked by administrator 5. Account is verified in database 6. The investor is notified about the successful registration 7. The investor is redirected to the main page

8.8 Contents for SRS

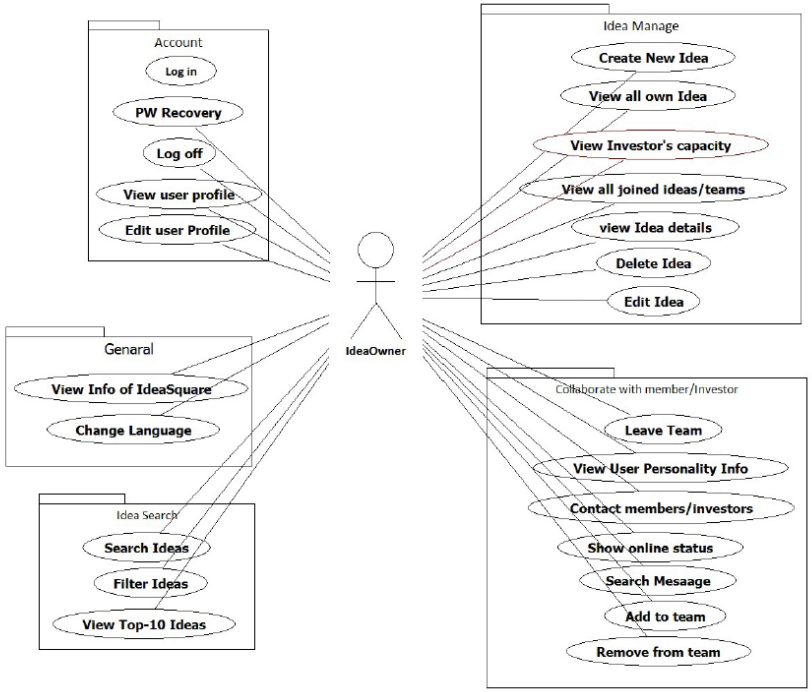
6. ***Use Case Diagram***

Use case diagram will be provided.

System context should be expressed.

The name of a use case should be same to the name of a functional requirement.

Example of use case diagram



8.8 Contents for SRS

7***. Supplementary Diagrams***

Class diagram, state diagram, activity diagram may be added to express requirements.