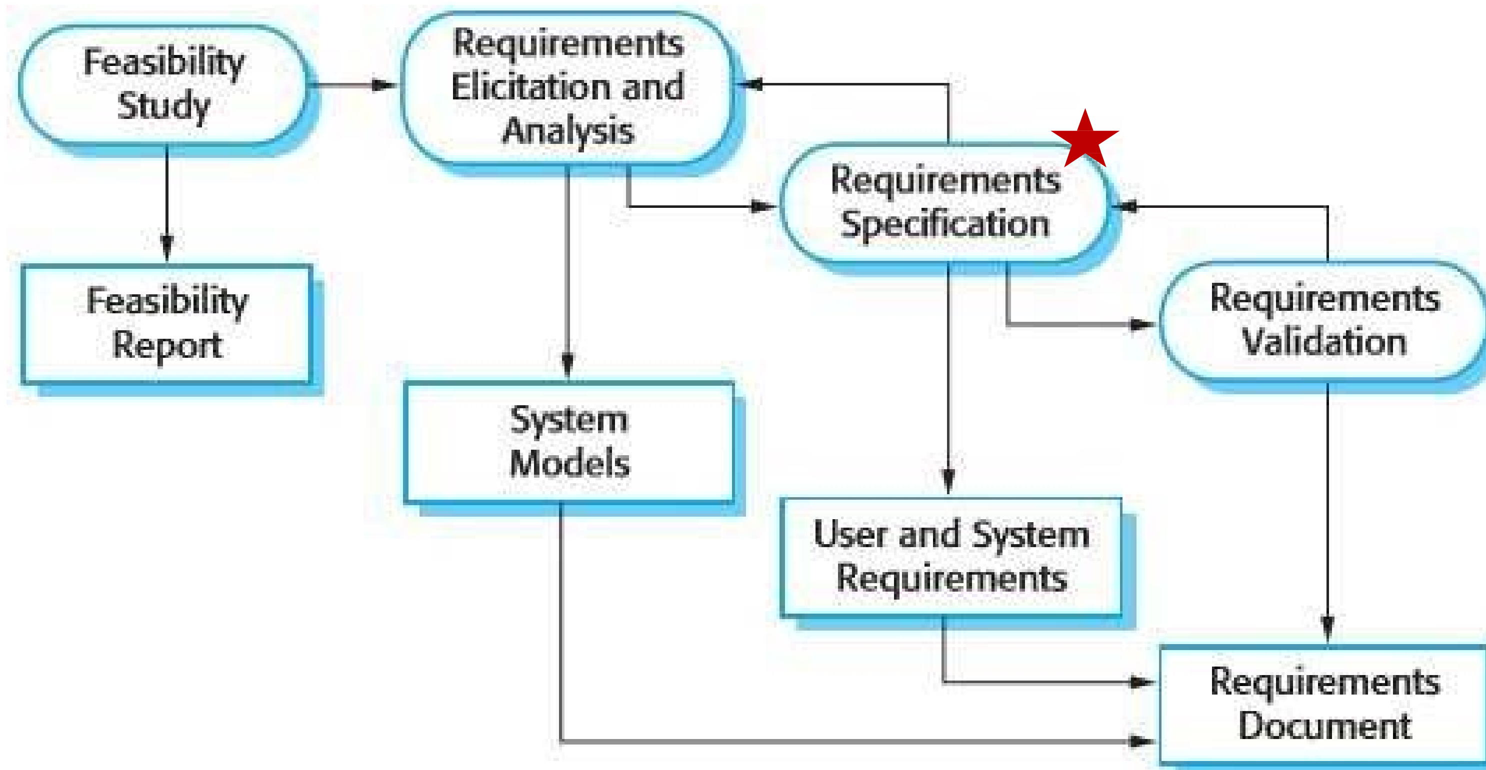
6. Requirements Specification

KU KONKUK UNIVERSITY

Requirements Engineering Process



KU KONKUK UNIVERSITY

Requirements

- Range from a high-level abstract statement of service or system constraint to detailed mathematical functional specification

- Types of requirements
  - **User requirements**
    - Statements in natural language, diagrams of the services the system provides and its operational constraints
    - Written for(from) customers
    - Defined
  - **System requirements**
    - Structured document setting out detailed descriptions of the system's functions, services and operational constraints.
    - Define what should be implemented to support user requirements
    - May be part of a contract between clients and contractors
    - Specified

Requirements Definitions and Specifications

User Requirement Definition

1. The software must provide a means of representing and accessing external files created by other tools.

System Requirement Specification

1. The user should be provided with facilities to define the type of external files.
2. Each external file type may have an associated tool which may be applied to the file.
3. Each external file type may be represented as a specific icon on the user's display.
4. Facilities should be provided for the icon representing an external file type to be defined by the user.
5. When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Functional vs. Non-Functional Requirements

- **Functional requirements**
  - Statements of services which the system should provide
  - How the system should react to particular inputs
  - How the system should behave in particular situations

- **Non-functional requirements**
  - Constraints on the services or functions offered by the system
    - Timing constraints
    - Constraints on the development process
    - Standards

- **Domain requirements**
  - Requirements that come from the application domain of the system
  - Reflect characteristics of the target domain
  - May be functional or non-functional or the both

Non-Functional Requirements

- **Define system properties and constraints**
  - Reliability, Response time, Storage requirements
  - Constraints on I/O device capability
  - System representations, Etc.

- **The challenge of NFRs**
  - Hard to model
  - Usually stated informally
    - Often contradictory, difficult to enforce during development
    - Difficult to evaluate for the customer prior to delivery
  - Hard to make them measurable requirements
    - We'd like to state them in a way that we can measure how well they've been met

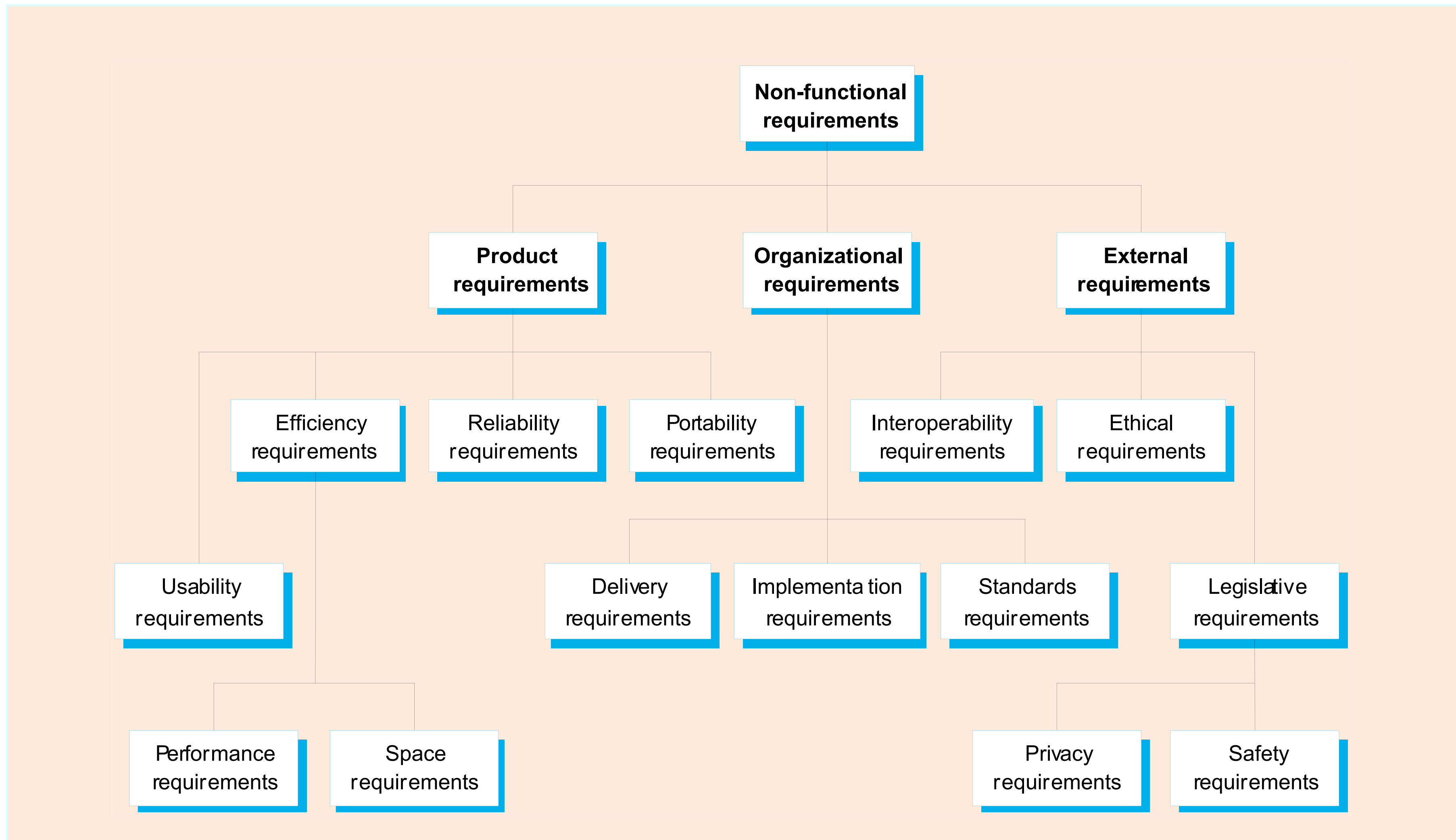
- Often called **Quality Attributes** or **Quality Requirements**
  - Often called just the "-ilities"
  - Non-functional requirements may be more critical than functional requirements.
  - If these are not met, the system is totally useless.

- Critical systems often include non-functional requirements into mandatory (i.e., functional) requirements.

Classification of Non-Functional Requirements

- Three types of non-functional requirements
  - **Product requirements**
    - Specify that the delivered product must behave in a particular way
    - e.g., execution speed, reliability, etc.
  - **Organizational requirements**
    - Requirements which are a consequence of organizational policies and procedures
    - e.g., process standards, implementation requirements, etc.
  - **External requirements**
    - Requirements which arise from the factors external to the development process
    - e.g. interoperability requirements, legislative requirements, etc.

3 Types of Non-Functional Requirements



Examples of Non-Functional Requirements

- Product requirement
  - *8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.*
- Organizational requirement
  - *9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.*
- External requirement
  - *7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.*

KU KONKUK UNIVERSITY

Goals and Requirements

- **Non-functional requirements** may be very difficult to state precisely.
  - Imprecise requirements may be also difficult to verify.
  - Write a "**goal**" first → transform into "**verifiable non-functional requirements**"

- **Goal**
  - A general intention of the user
  - Example : "ease of use" → "*The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.*"

- **Verifiable** non-functional requirement
  - A statement using some measure that can be tested objectively
  - "*Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.*"

Domain Requirements

- Describe system characteristics and features of the target domain
  - **Derived from the application domain**

- Domain requirements may be
  - New functional requirements
  - Constraints on existing requirements
  - Definition of specific computations

- If domain requirements are not satisfied, the system may be unworkable.

Domain Requirements Example : LIBSYS

Domain Requirements

1. There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.

2. Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Requirements Completeness and Consistency

- Problems arise when requirements are not precisely stated.
  - Ambiguous requirements may be interpreted in different ways.

- In principle, requirements should be both complete and consistent.
  - **Complete** : Should include descriptions of all facilities required
  - **Consistent** : 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 with natural languages.
  - Need for (formal/informal/semi-formal) requirements models

Software Requirements Document

- **SRS** (Software Requirements Specification) or **SRD** (Software Requirements Document)

- The software requirements document is the **official statement** of **what is required of the system developers**.
  - Should include both a definition of user requirements and a specification of the system requirements
  - NOT a design document.
  - As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

- The goal of requirements engineering:
  - "***Not to write the perfect requirements specification, but create the best possible product at the right time***"

KU KONKUK UNIVERSITY

Purposes of SRS

- How do we communicate the Requirements to others?
  - It is common practice to capture them in an SRS
    - But, an SRS doesn't need to be a single paper document
- Purpose
  - Communication
    - Explains the application domain and the system to be developed
  - Contractual
    - May be legally binding!
    - Expresses agreement and a commitment
  - Baseline for evaluating the software
    - Supporting testing, V&V
    - "Enough information to verify whether delivered system meets requirements"
  - Baseline for change control

Features for Good Specifications

Features	Considerations
Valid (Correct)	- Expresses the real needs of the stakeholders (customers, users,…) - Does not contain anything that is not "required"
Unambiguous	- Every statement can be read in exactly one way
Complete	- All the things the system must do and all the things it must not do! - Conceptual Completeness • E.g., responses to all classes of input - Structural Completeness • E.g., no TBDs!!!
Understandable (Clear)	- E.g. by non-computer specialists
Consistent	- Doesn't contradict itself - Uses all terms consistently
Ranked	- Indicates relative importance / stability of each requirement
Verifiable	- A process exists to test satisfaction of each requirement
Modifiable	- Can be changed without difficulty • Good structure and cross-referencing
Traceable	- Origin of each requirement is clear - Labels each requirement for future referencing

SRS Contents

- **Software Requirements Specification** should address:
  - **Functionality**
    - What is the software supposed to do?
  - **External interfaces**
    - How does the software interact with people, the system's hardware, other hardware, and other software?
    - What assumptions can be made about these external entities?
  - **Required performance**
    - What is the speed, availability, response time, recovery time of various software functions, and so on?
  - **Quality attributes**
    - What are the portability, correctness, maintainability, security, and other considerations?
  - **Design constraints imposed on an implementation**
    - Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) and so on?

SRS Should Not Include

- **Project development plans**
  - E.g. cost, staffing, schedules, methods, tools, etc
    - Lifetime of SRS is until the software is made obsolete
    - Lifetime of development plans is much shorter
- **Product assurance plans**
  - Configuration Management, Verification & Validation, test plans, Quality Assurance, etc.
    - Different audiences
    - Different lifetimes
- **Designs**
  - Requirements and designs have different audiences
  - Analysis and design are different areas of expertise
  - Except where application domain constrains the design
    - E.g., limited communication between different subsystems for security reasons

Typical Mistakes in SRS

Mistakes	Description
Noise	text that carries no relevant information to any feature of the problem
Silence	a feature that is not covered by any text
Over-Specification	text that describes a detailed design decision, rather than the problem
Contradiction	text that defines a single feature in a number of incompatible ways
Ambiguity	text that can be interpreted in at least two different ways
Forward Reference	text that refers to a terms or features yet to be defined
Wishful Thinking	text that defines a feature that cannot possibly be verified
Requirements on Users	Cannot require users to do certain things, can only assume that they will
Jigsaw Puzzles	Distributing key information across a document and then cross-referencing
Duck Speak Requirements	Requirements that are only there to conform to standards
Unnecessary Invention of Terminology	e.g. 'user input presentation function'
Inconsistent Terminology	Inventing and then changing terminology
Putting the onus on the developers	i.e. making the reader work hard to decipher the intent
Writing for the hostile reader	There are fewer of these than friendly readers

Requirements and Design are Inseparable

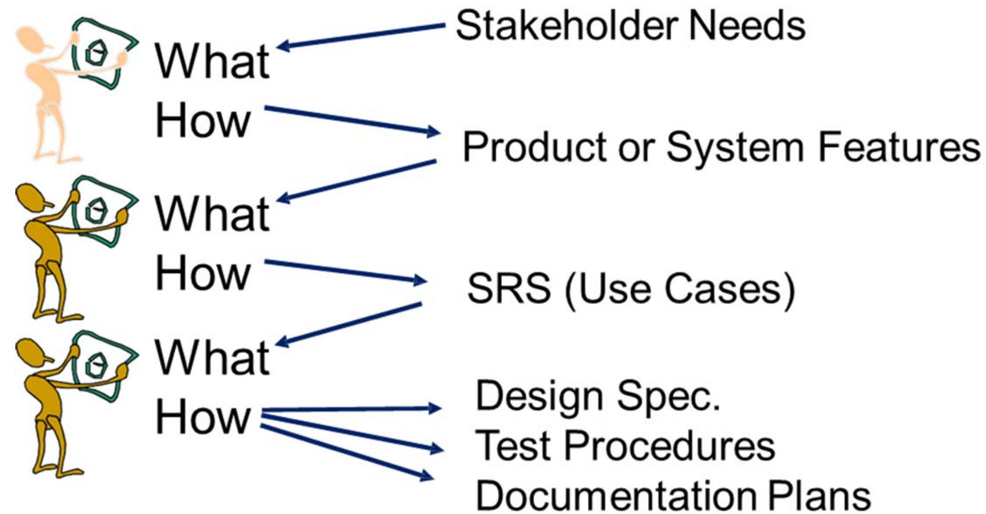
- In principle,
  - **Requirements** should state what the system should do.
  - **Design** should describe how it does this.

- In practice, requirements and design are inseparable.
  - A system architecture may be designed to structure the requirements.
  - The system may inter-operate with other systems that generate design requirements.
  - The use of a specific architecture to satisfy non-functional requirements may be a domain requirement.
  - This may be the consequence of a regulatory requirement.

What vs. How Dilemma

- Question : “How can you tell a requirement from design?”
- Answer : “It depends on your point of view.”

“One man’s ceiling is another man’s floor.”

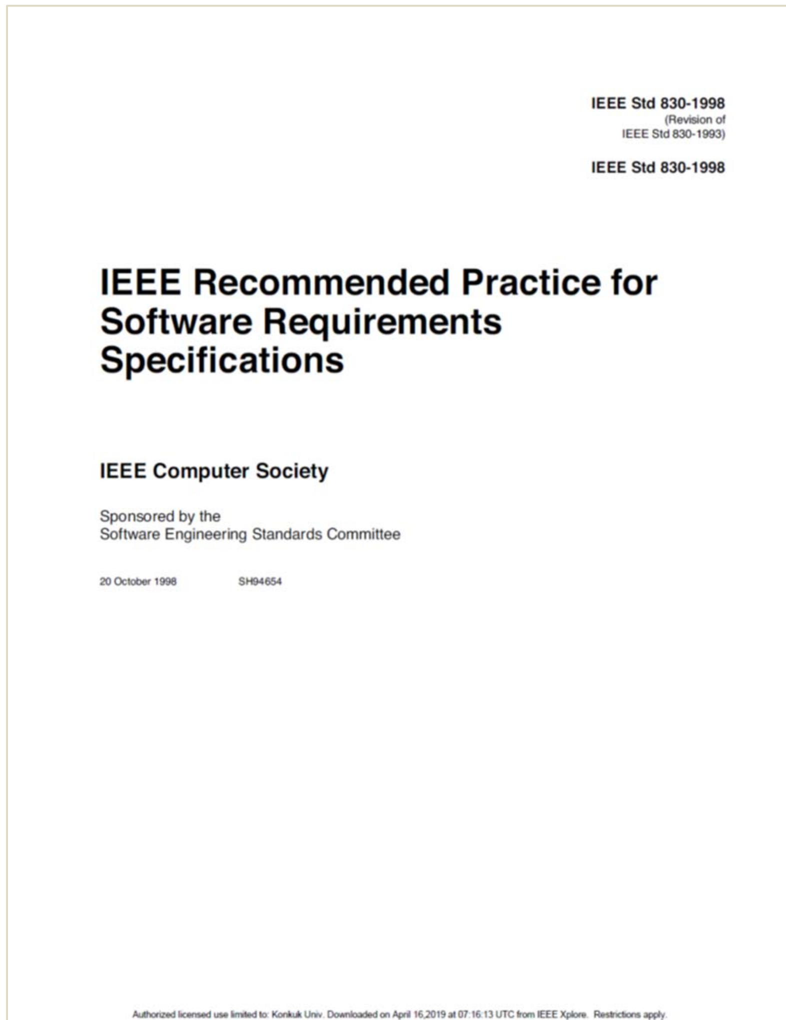


Requirements Document Variability

- Information in requirements document depends on the type of system and the approach to development used.
  - If systems are developed incrementally, it will typically have less detail in the requirements document.

- **Requirements documents standards** have been designed.
  - E.g., **IEEE standards**
  - Mostly applicable to the requirements for large systems engineering projects

SRS Standard: IEEE STD 830-1998


IEEE Std 830-1998
(Revision of
IEEE Std 830-1993)

IEEE Std 830-1998

IEEE Recommended Practice for Software Requirements Specifications

IEEE Computer Society

Sponsored by the
Software Engineering Standards Committee

20 October 1998 SH94654

Authorized licensed use limited to: Konkuk Univ. Downloaded on April 16,2019 at 07:16:13 UTC from IEEE Xplore. Restrictions apply.

Table of Contents

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Overall description
   - 2.1 Product perspective
   - 2.2 Product functions
   - 2.3 User characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and dependencies
3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)

Appendixes

Index

Figure 1—Prototype SRS outline

SRS Templates: IEEE STS 830.1998

A.1 Template of SRS Section 3 organized by mode: Version 1

3. Specific requirements
3.1 External interface requirements
    3.1.1 User interfaces
    3.1.2 Hardware interfaces
    3.1.3 Software interfaces
    3.1.4 Communications interfaces
3.2 Functional requirements
    3.2.1 Mode 1
        3.2.1.1 Functional requirement 1.1
        .
        .
        .
        3.2.1.*n* Functional requirement 1.*n*
    3.2.2 Mode 2
    .
    .
    .
    3.2.*m* Mode *m*
        3.2.*m*.1 Functional requirement *m*.1
        .
        .
        .
        3.2.*m*.*n* Functional requirement *m*.*n*
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

A.2 Template of SRS Section 3 organized by mode: Version 2

3. Specific requirements
3.1. Functional requirements
    3.1.1 Mode 1
        3.1.1.1 External interfaces
            3.1.1.1.1 User interfaces
            3.1.1.1.2 Hardware interfaces
            3.1.1.1.3 Software interfaces
            3.1.1.1.4 Communications interfaces
        3.1.1.2 Functional requirements
            3.1.1.2.1 Functional requirement 1
            .
            .
            .
            3.1.1.2.*n* Functional requirement *n*
        3.1.1.3 Performance
    3.1.2 Mode 2
    .
    .
    .
    3.1.*m* Mode *m*
3.2 Design constraints
3.3 Software system attributes
3.4 Other requirements

KU KONKUK UNIVERSITY

SRS Templates: IEEE STS 830.1998

A.3 Template of SRS Section 3 organized by user class

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 User class 1
    - 3.2.1.1 Functional requirement 1.1
    - .
    - .
    - .
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 User class 2
  - .
  - .
  - .
  - 3.2.*m* User class *m*
    - 3.2.*m*.1 Functional requirement *m*.1
    - .
    - .
    - .
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

A.4 Template of SRS Section 3 organized by object

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Classes/Objects
  - 3.2.1 Class/Object 1
    - 3.2.1.1 Attributes (direct or inherited)
      - 3.2.1.1.1 Attribute 1
      - .
      - .
      - .
      - 3.2.1.1.*n* Attribute *n*
    - 3.2.1.2 Functions (services, methods, direct or inherited)
      - 3.2.1.2.1 Functional requirement 1.1
      - .
      - .
      - .
      - 3.2.1.2.*m* Functional requirement 1.*m*
    - 3.2.1.3 Messages (communications received or sent)
  - 3.2.2 Class/Object 2
  - .
  - .
  - .
  - 3.2.*p* Class/Object *p*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

SRS Templates: IEEE STS 830.1998

A.5 Template of SRS Section 3 organized by feature

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 System features
  - 3.2.1 System Feature 1
    - 3.2.1.1 Introduction/Purpose of feature
    - 3.2.1.2 Stimulus/Response sequence
    - 3.2.1.3 Associated functional requirements
      - 3.2.1.3.1 Functional requirement 1
      - .
      - .
      - .
      - 3.2.1.3.*n* Functional requirement *n*
  - 3.2.2 System feature 2
  - .
  - .
  - .
  - 3.2.*m* System feature *m*
  - .
  - .
  - .
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

A.6 Template of SRS Section 3 organized by stimulus

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Stimulus 1
    - 3.2.1.1 Functional requirement 1.1
    - .
    - .
    - .
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 Stimulus 2
  - .
  - .
  - .
  - 3.2.*m* Stimulus m
    - 3.2.*m*.1 Functional requirement *m*.1
    - .
    - .
    - .
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

SRS Templates: IEEE STS 830.1998

A.7 Template of SRS Section 3 organized by functional hierarchy

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Information flows
    - 3.2.1.1 Data flow diagram 1
      - 3.2.1.1.1 Data entities
      - 3.2.1.1.2 Pertinent processes
      - 3.2.1.1.3 Topology
    - 3.2.1.2 Data flow diagram 2
      - 3.2.1.2.1 Data entities
      - 3.2.1.2.2 Pertinent processes
      - 3.2.1.2.3 Topology
    - .
    - .
    - .
    - 3.2.1.*n* Data flow diagram *n*
      - 3.2.1.*n*.1 Data entities
      - 3.2.1.*n*.2 Pertinent processes
      - 3.2.1.*n*.3 Topology
  - 3.2.2 Process descriptions
    - 3.2.2.1 Process 1
      - 3.2.2.1.1 Input data entities
      - 3.2.2.1.2 Algorithm or formula of process
      - 3.2.2.1.3 Affected data entities
    - 3.2.2.2 Process 2
      - 3.2.2.2.1 Input data entities
      - 3.2.2.2.2 Algorithm or formula of process
      - 3.2.2.2.3 Affected data entities
    - .
    - .
    - .
    - 3.2.2.*m* Process *m*
      - 3.2.2.*m*.1 Input data entities
      - 3.2.2.*m*.2 Algorithm or formula of process
      - 3.2.2.*m*.3 Affected data entities
  - 3.2.3 Data construct specifications
    - 3.2.3.1 Construct 1
      - 3.2.3.1.1 Record type
      - 3.2.3.1.2 Constituent fields
    - 3.2.3.2 Construct 2
      - 3.2.3.2.1 Record type
      - 3.2.3.2.2 Constituent fields
    - .
    - .
    - .
    - 3.2.3.*p* Construct *p*
      - 3.2.3.*p*.1 Record type
      - 3.2.3.*p*.2 Constituent fields
  - 3.2.4 Data dictionary
    - 3.2.4.1 Data element 1
      - 3.2.4.1.1 Name
      - 3.2.4.1.2 Representation
      - 3.2.4.1.3 Units/Format
      - 3.2.4.1.4 Precision/Accuracy
      - 3.2.4.1.5 Range
    - 3.2.4.2 Data element 2
      - 3.2.4.2.1 Name
      - 3.2.4.2.2 Representation
      - 3.2.4.2.3 Units/Format
      - 3.2.4.2.4 Precision/Accuracy
      - 3.2.4.2.5 Range
    - .
    - .
    - .
    - 3.2.4.*q* Data element *q*
      - 3.2.4.*q*.1 Name
      - 3.2.4.*q*.2 Representation
      - 3.2.4.*q*.3 Units/Format
      - 3.2.4.*q*.4 Precision/Accuracy
      - 3.2.4.*q*.5 Range
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

SRS Templates: IEEE STS 830.1998

A.8 Template of SRS Section 3 showing multiple organizations

3. Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 User class 1
    - 3.2.1.1 Feature 1.1
      - 3.2.1.1.1 Introduction/Purpose of feature
      - 3.2.1.1.2 Stimulus/Response sequence
      - 3.2.1.1.3 Associated functional requirements
    - 3.2.1.2 Feature 1.2
      - 3.2.1.2.1 Introduction/Purpose of feature
      - 3.2.1.2.2 Stimulus/Response sequence
      - 3.2.1.2.3 Associated functional requirements
    - .
    - .
    - .
    - 3.2.1.*m* Feature 1.*m*
      - 3.2.1.*m*.1 Introduction/Purpose of feature
      - 3.2.1.*m*.2 Stimulus/Response sequence
      - 3.2.1.*m*.3 Associated functional requirements
  - 3.2.2 User class 2
  - .
  - .
  - .
  - 3.2.*n* User class *n*
  - .
  - .
  - .
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

IEEE STD 830-1998 – Incorrect Translation

1. 소개
- 1.1 목적
- 1.2 범위
- 1.3 정의, 약어
- 1.4 참조
- 1.5 개요

2. 전반적 서술
- 2.1 제품 관점
  - 시스템 인터페이스
  - 사용자 인터페이스
  - 하드웨어 인터페이스
  - 소프트웨어 인터페이스
  - 통신 인터페이스
  - 메모리
  - 운영
- 2.2 제품 기능
- 2.3 사용자 특성
- 2.4 제약 사항
  - 규제 정책, 하드웨어 제약 사항, 다른 응용 프로그램과의 인터페이스, 병렬 수행, 감사 기능, 제어 기능, 신뢰성 요구 사항, 안전 및 보안 요구 사항
- 2.5 가정 및 의존성
- 2.6 요구 사항 할당

3. 구체적 요구 사항
- 3.1 외부 인터페이스
  - 사용자 인터페이스, 하드웨어 인터페이스, 소프트웨어 인터페이스, 통신 인터페이스 기능
- 3.2 성능 요구 사항
- 3.3 로컬 DB 요구 사항
- 3.4 설계 제약 사항
- 3.5 소프트웨어 시스템 속성
  - 신뢰성, 가용성, 보안성, 유지보수 용이성, 이식성

인덱스

부록

NAVER

Problems of SRS in Natural Languages

- **Lack of clarity**
  - Precision is difficult without making the document difficult to read.
- **Requirements confusion**
  - Functional and non-functional requirements tend to be mixed-up.
- **Requirements amalgamation**
  - Several different requirements may be expressed together.

- Example : insulin pump software system

> 3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*
>
> 3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured Specifications

- An approach to writing requirements where
  - The freedom of the requirements writer is limited and
  - Requirements are written in a standard way.

- This works well for some types of requirements such as embedded control system.
  - But, sometimes too rigid for writing business system requirements
  - **Form-based specification**
  - **Tabular specification**

Form-based Specifications

- Specification includes information **in a form format** :
  - Definition of the function or entity
  - Description of inputs, source, outputs, and destination
  - Description of the action to be taken
  - Pre and post conditions (if appropriate)
  - The side effects (if any) of the function

- Requirements for the insulin pump software system

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

Action

CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements

Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition

The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition r0 is replaced by r1 then r1 is replaced by r2.

Side effects None.

Tabular Specification

- Particularly useful when you have to define a number of possible alternative courses of action
  - For example, the insulin pump systems bases its computations on the rate of change of blood sugar level, and the tabular specification explains how to calculate the insulin requirement for different scenarios.

- Requirements for the insulin pump software system

Condition	Action
Sugar level falling ($r2 < r1$)	CompDose = 0
Sugar level stable ($r2 = r1$)	CompDose = 0
Sugar level increasing and rate of increase decreasing $((r2 – r1) < (r1 – r0))$	CompDose = 0
Sugar level increasing and rate of increase stable or increasing $((r2 – r1) \geq (r1 – r0))$	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

KU KONKUK UNIVERSITY

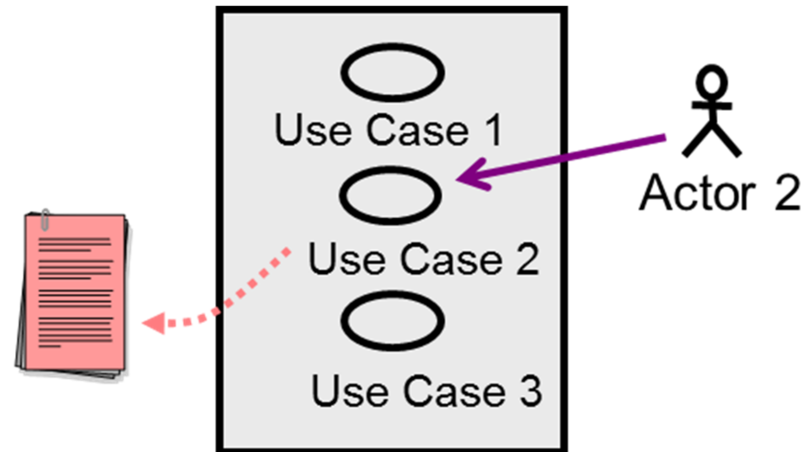
7. Use Case Analysis

What is Use Case Modeling?

- A means for capturing the desired behavior for the system under development
  - A way to communicate the system's behavior with various stakeholders
  - A way to verify all requirements are captured

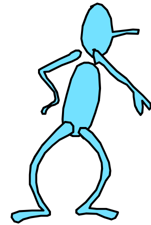
- Identifies
  1. Who or what interacts with the system
  2. What the system should do

- A planning instrument

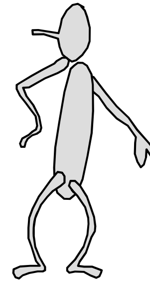


Who Reads Use Cases?

- Client Team

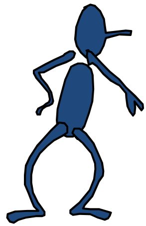


Client

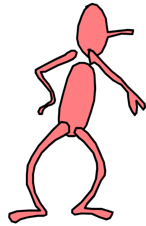


Users

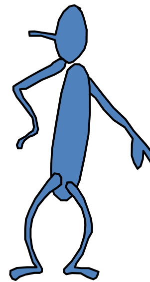
- Developer Team



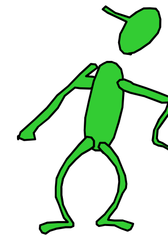
Tester



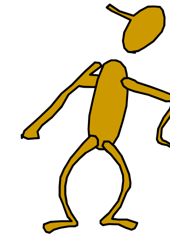
Designer



Requirements Specifier



Technical Writer



Project Manager

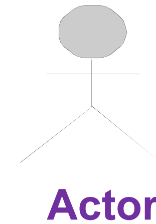
Benefits of Use Cases

- Give context for requirements
- Easy to understand
- Facilitate agreement with customers
- Illustrate why the system is needed
  - Use cases: why the system is used
  - Actors: who/what wants to interact with the system

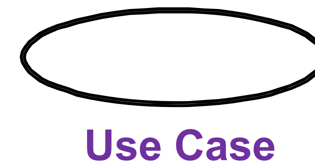
- The idea behind use cases is to **decide what the system will be used for** **before** **defining what the system is supposed to do.**

Actors and Use Cases

- Actor
  - Someone/something outside the system that interacts with the system



- Use case
  - What an actor wants to use the system to do

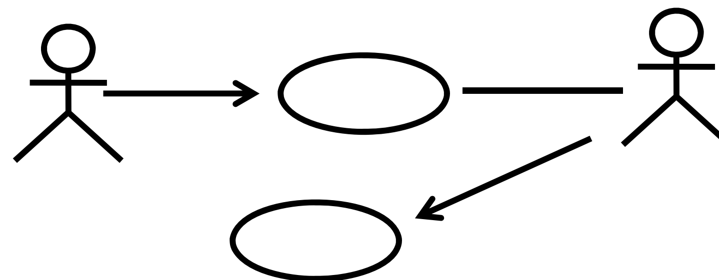


What is a Use Case?

- A use case defines a sequence of actions performed by a system that yields an observable result of value to an actor.

Use Cases Contain Software Requirements

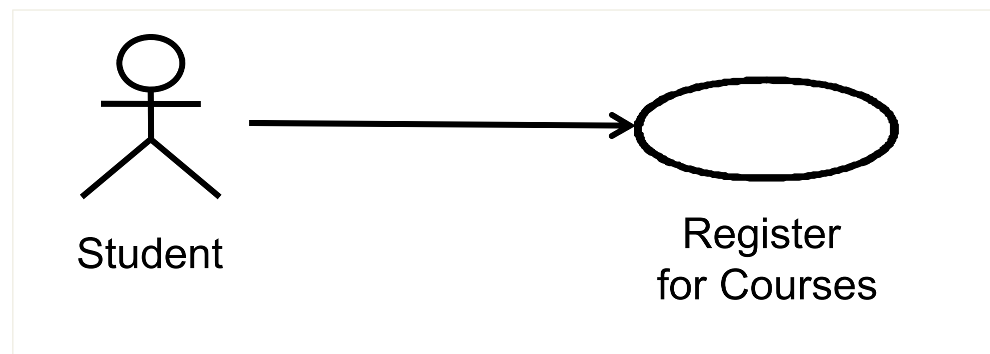
- Each Use Case
  - Describes actions the system takes to deliver something of value to the actor
  - Shows the system functionality an actor uses
  - Models a dialog between the system and actors
  - Shows a complete and meaningful flow of events from the perspective of a particular actor



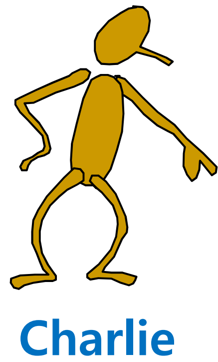
Instances of Actors

Sam
acts as a Student

Jody
acts as a Student

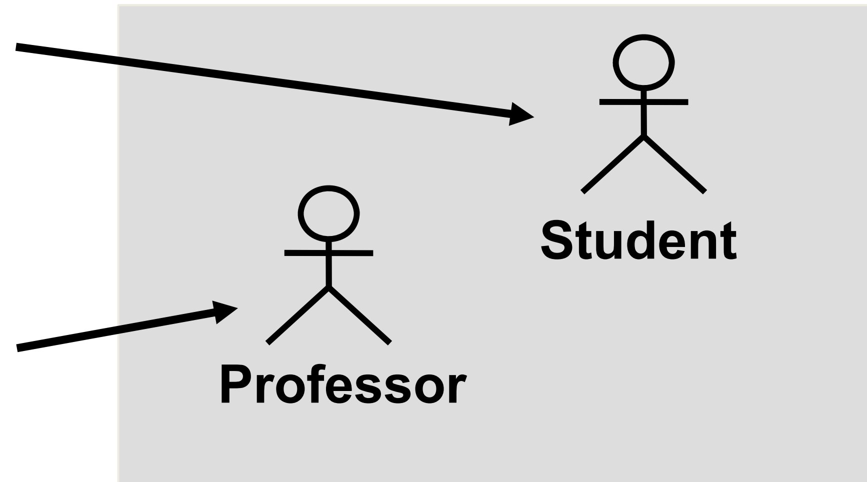


A User Can Act as Several Actors



acts as a Student

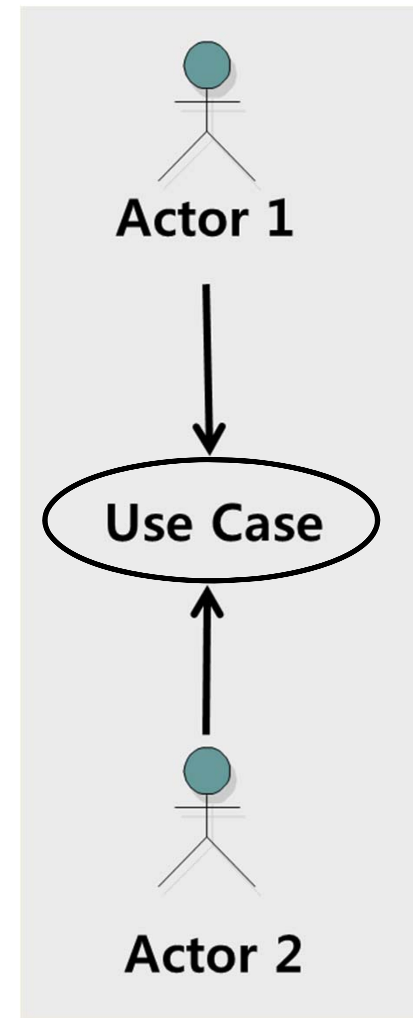
acts as a Professor



Communicates-Association

- A channel of communication between an actor and a use case
  - A line (arrow) is used.

- An arrow indicates who initiates the communication.

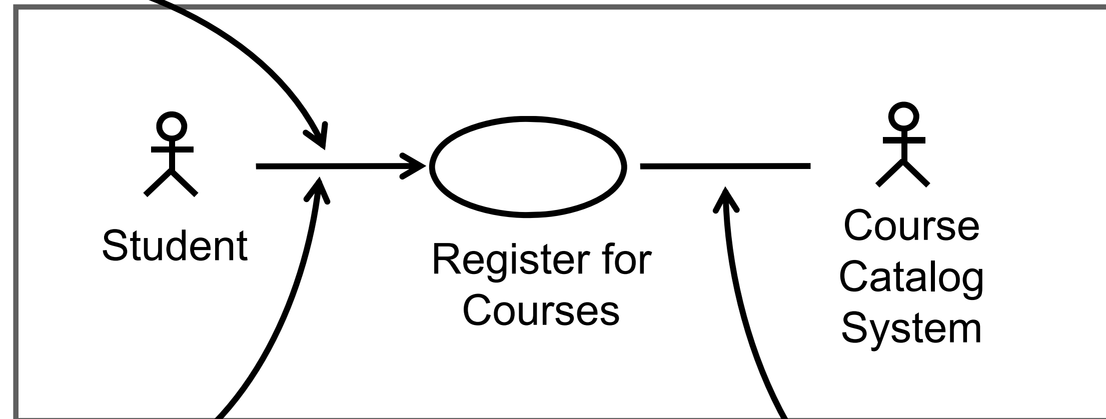


Each Communicates-Association is a Whole Dialog

Student logs on to system

System approves log on

Student requests course info



System displays course list

Student select courses

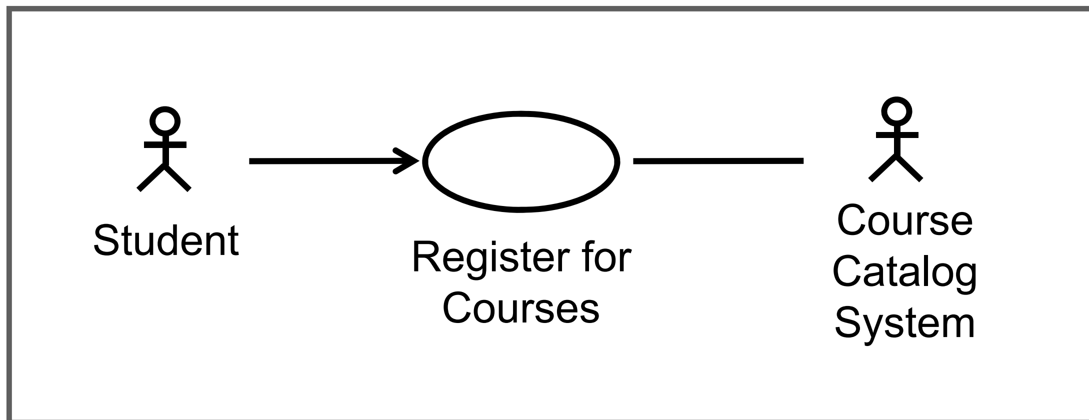
System confirms course availability

System displays approved schedule

System transmits request

Course Catalog returns course info

A Scenario is a Use Case Instance



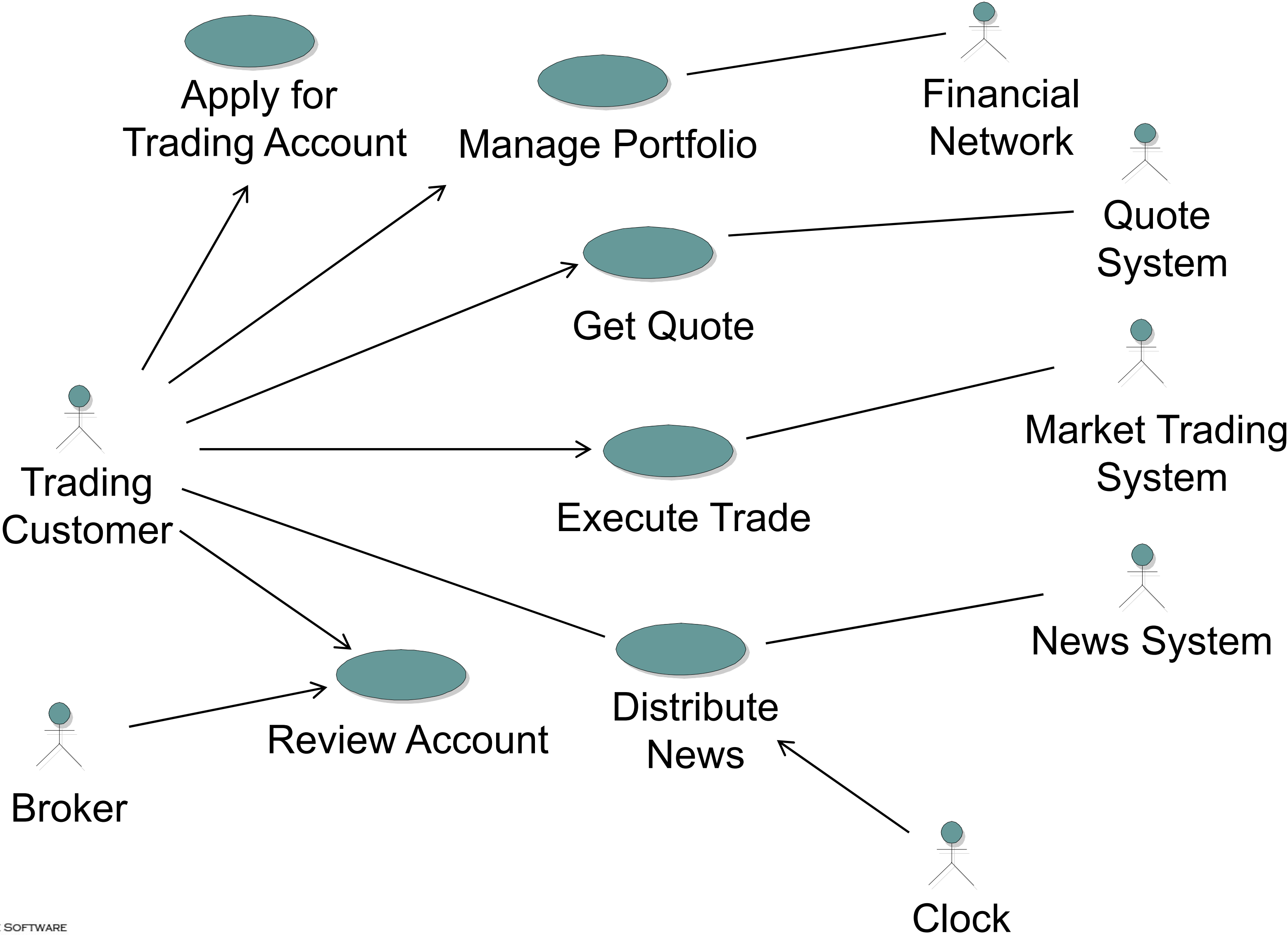
Scenario 1

- Log on to system
- Approve log on
- Enter subject in search
- Get course list
- Display course list
- Select courses
- Confirm availability
- Display final schedule

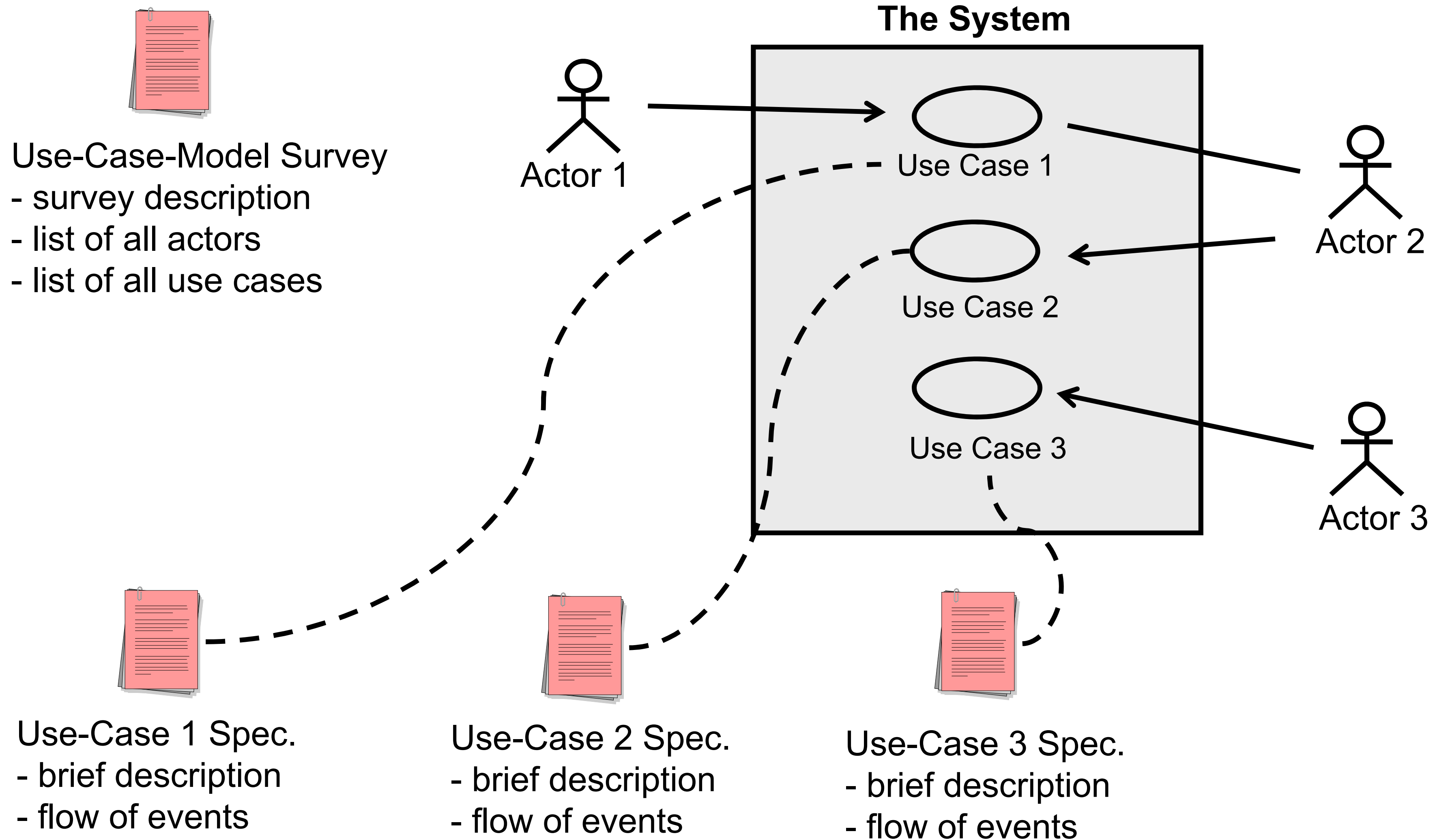
Scenario 2

- Log on to system
- Approve log on
- Enter subject in search
- *Invalid subject*
- *Re-enter subject*
- Get course list
- Display course list
- Select courses
- Confirm availability
- Display final schedule

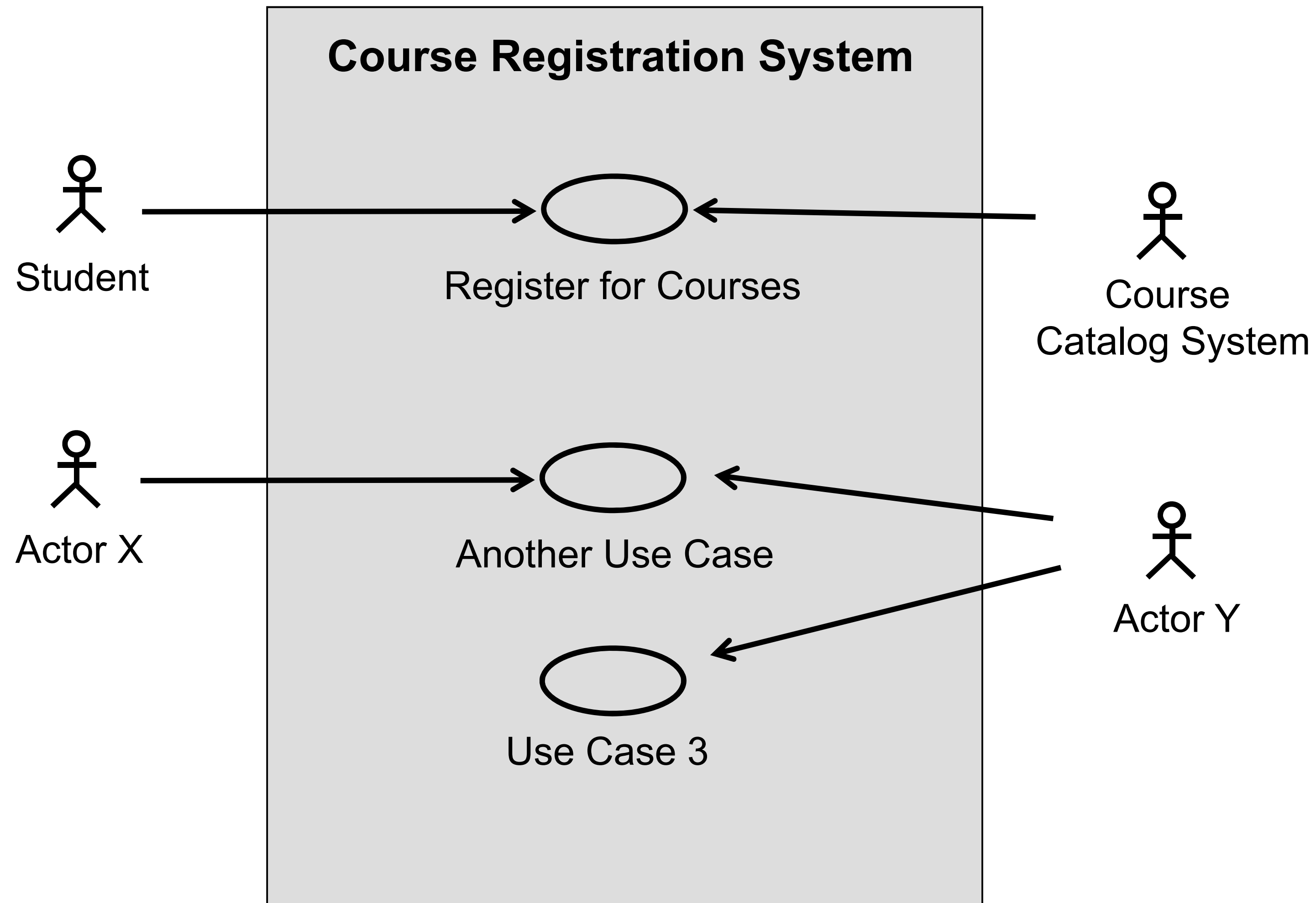
Use Case Diagram



A Use Case Model Contains Diagrams and Text



Example: Online Course Registration System



How Should I Name a Use Case?

- **A use case name indicates the value or goal.**

- Use the active form: begin with a verb
  - Imagine a to-do list

- Examples of variations
  - Register for Courses
  - Registering for Courses
  - Acknowledge Registration
  - Course Registration
  - Use Registration System

- Which variations show the value to the actor? Which do not?
- Which would you choose as the use-case name? Why?

Use Case Tips

- Describe only the events visible to the actor:
  - What the actor does
  - What the system does in response
    → "**Actor-activated Use Case**"

- Make use cases provide **value** to an actor
  - Detail until everyone has a common understanding of the requirements, then stop

- Make all use cases of the same level

- Sketch the user interface, but don't detail it.

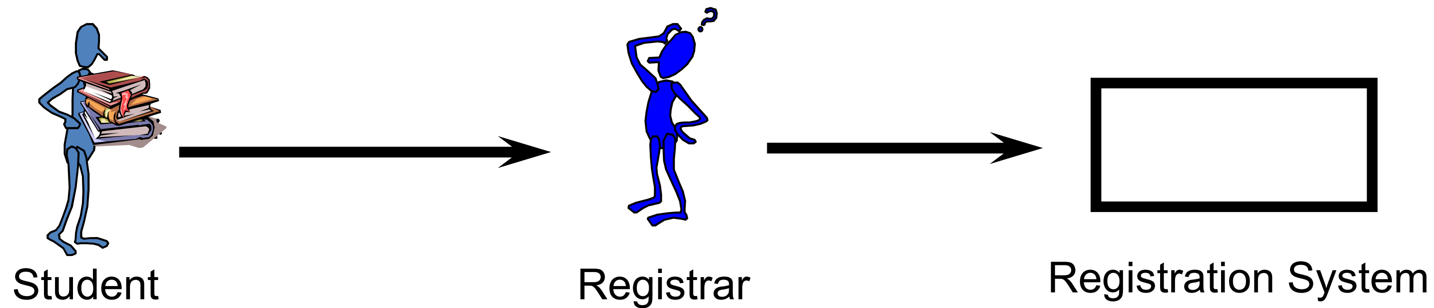
Steps for Creating a Use Case Model

1. **Find actors and use cases**
   - Identify and describe actors
   - Identify and describe use cases

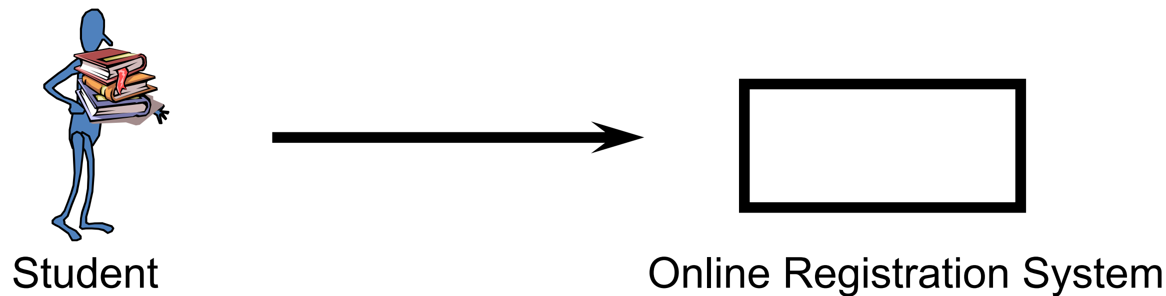
2. **Write the use cases**
   - Outline all use cases
   - Prioritize and detail the use cases

Find Actors

- Who is pressing the keys (interacting with the system)?



The student never touches this system; the registrar operates it.
Or perhaps you are building an Internet application?



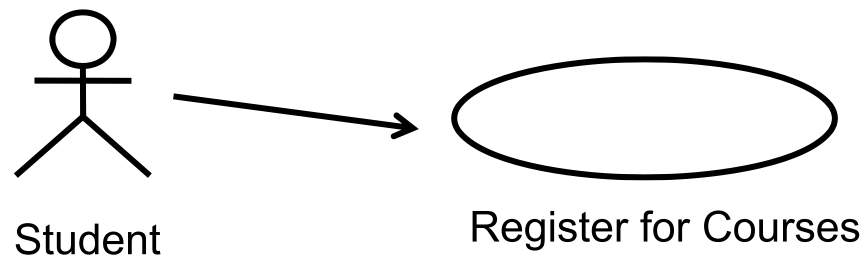
Identify Actors

- Who/what uses the system?
- Who/what gets information from this system?
- Who/what provides information to the system?
- Where in the company is the system used?
- Who/what supports and maintains the system?
- What other systems use this system?

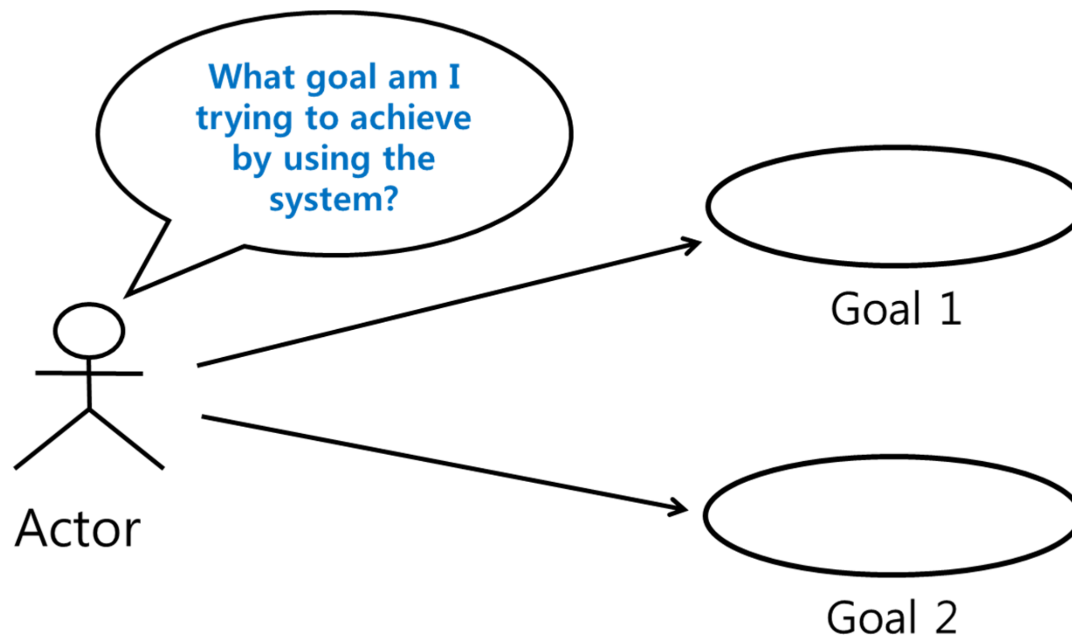
Description of an Actor

- Text
  - Name
  - Brief description
  - Relationship with use cases

- Example
  - Student : “A person who signs up for a course”



Find Use Cases



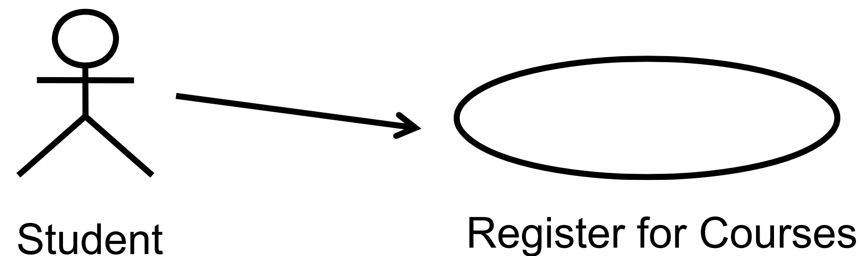
Identify Use Cases

- What are the goals of each actor?
  - Why does the actor want to use the system?
  - Will the actor create, store, change, remove, or read data in the system? If so, why?
  - Will the actor need to inform the system about external events or changes?
  - Will the actor need to be informed about certain occurrences in the system?
- Does the system supply the business with all of the correct behavior?

Description of a Use Case

- Text description of a use case
  - Name
  - Brief description
  - Relationship with actors

- Example
  - Register for Courses : "*The student registers for courses. The student obtains course information prior to registering*."



Functional Decomposition

- **Functional Decomposition**
  - Breakdown of a problem into small isolated parts
  - The parts:
    - Works together to provide the functionality of the system
    - Often do not make sense in isolation
- **Use Cases:**
  - **Are NOT functional decomposition**
  - Keep the functionality together to describe a complete use of the system
  - Provide context

Avoid Functional Decomposition

- **Symptoms**
  - Very small use cases
  - Too many use cases
  - Uses cases with no result of value
  - Names with low-level operations
    - “Operation” + “object”
    - “Function” + “data”
    - Example: “Insert Card”
  - Difficulty on understanding the overall model
- **Corrective Actions**
  - Search for larger context
    - “Why are you building this system?”
  - Put yourself in user’s role
    - “What does the user want to achieve?”
    - “Whose goal does this use case satisfy?”
    - “What value does this use case add?”
    - “What is the story behind this use case?”

Checkpoint for Use Cases

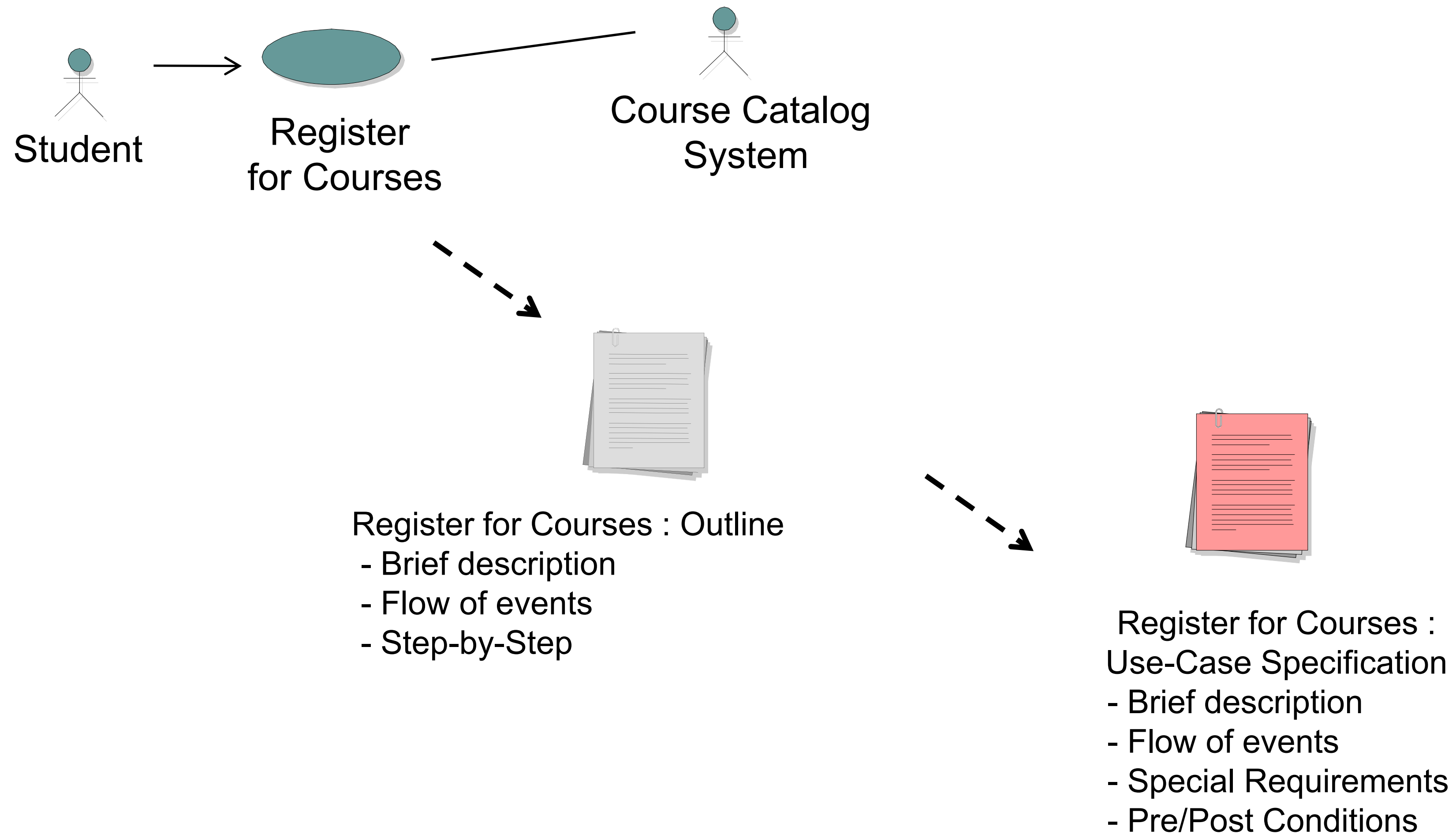
- The use-case model clearly presents the behavior of the system; it is easy to understand what the system does by reviewing the model.
- All use cases have been identified; the use cases collectively account for all required behavior.
- All functional requirements are mapped to at least one use case.
- The use-case model contains no superfluous behavior; all use cases can be justified by tracing them back to a functional requirement.
- Do the use cases have unique, intuitive and explanatory names so that they cannot be mixed up at a later stage? If not, change their names.
- Do customers and users alike understand the names and descriptions of the use cases?
- Does the brief description give a true picture of the use case?
- Is each use case involved with at least one actor?
- Do any use cases have very similar behaviors or flows of events?

KU KONKUK UNIVERSITY

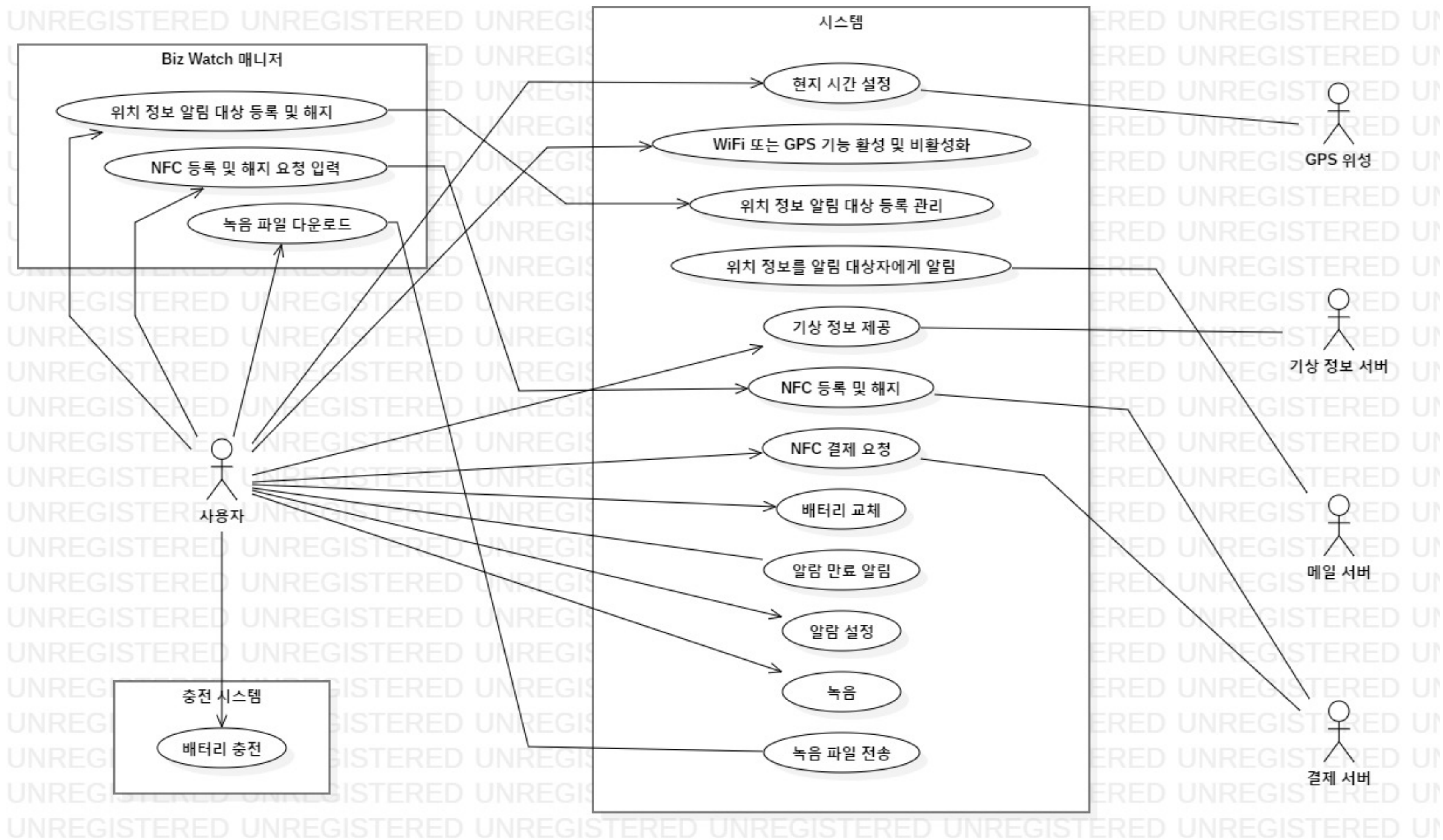
Checkpoint for Actors

- Have you found all the actors? That is, have you accounted for and modeled all roles in the system's environment?
- Is each actor involved with at least one use case?
- Can you name at least two people who would be able to perform as a particular actor?
- Do any actors play similar roles in relation to the system? If so, you should merge them into a single actor.

Diagram → Outline → Detail

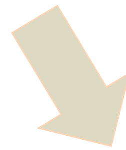


Use-Case Diagrams : Examples



Use Case	1. Make Reservation
Actors	Librarian
Description	- This use case begins when a borrower arrives at the counter and then requests reservation. - For a registered borrower, it makes a reservation slip (software-wise). - For an unregistered borrower, the librarian registers the person and makes a reservation for the person.

(Planning)

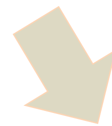


(Analysis)

Use Case	1. Make Reservation
Actor	Librarian
Purpose	(As in the business use case)
Overview	(As in the business use case)
Type	Primary and Essential
Cross Reference	System Functions: R1.1, R3.1 Use Case: "Add Borrower"
Pre-Requisites	Borrower should have an id_card.
Typical Courses of Events	(A) : Actor, (S) : System 1. (A) A librarian requests the reservation of title 2. (S) Check if a corresponding title exists 3. (S) Check if a corresponding borrower exists 4. (S) If the borrower does not exist, invoke "Add Borrower" 5. (S) Create reservation information
Alternative Courses of Events	N/A
Exceptional Courses of Events	Line 1: If invalid reservation information is entered, indicate an error.

Use Case	1. Make Reservation
Actor	Librarian
Purpose	(As in the business use case)
Overview	(As in the business use case)
Type	Primary and Essential
Cross Reference	System Functions: R1.1, R3.1 Use Case: "Add Borrower"
Pre-Requisites	Borrower should have an id_card.
Typical Courses of Events	(A) : Actor, (S) : System 1. (A) A librarian requests the reservation of title 2. (S) Check if a corresponding title exists 3. (S) Check if a corresponding borrower exists 4. (S) If the borrower does not exist, invoke "Add Borrower" 5. (S) Create reservation information
Alternative Courses of Events	N/A
Exceptional Courses of Events	Line 1: If invalid reservation information is entered, indicate an error.

(Analysis)



(Design)

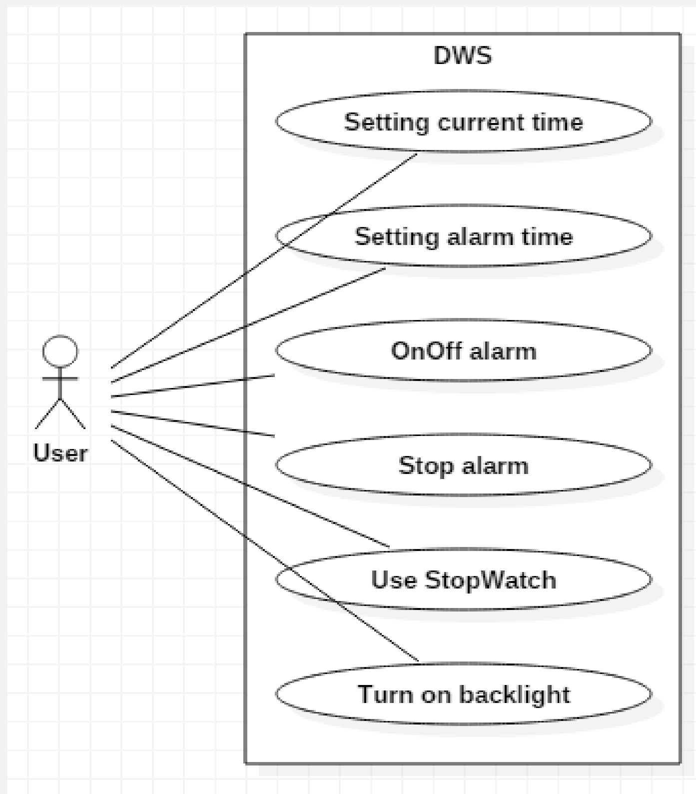
Use Case	1. Make Reservation
Actor	Librarian
Purpose	Create a new reservation
Overview	(As in the business use case)
Type	Primary and Real
Cross Reference	System Functions: R1.1, R3.1 Use Case: "Add Borrower"
Pre-Requisites	A borrower should be registered.
Typical Courses of Events	(A) : Actor, (S) : System 1. (A) A librarian inputs an *isbn* and *ssn* of the title 2. (S) Find a corresponding title 3. (S) Find a corresponding borrower 4. (S) Create a new reservation 5. (S) Store the new reservation 6. (S) Increase *reservationCount* in the borrower 7. (S) Increase *reservationCount* in the title
Alternative Courses of Events	N/A
Exceptional Courses of Events	Line 2: If the title does not exist, display an error message. Line 3: If the borrower does not exist, display an error message.

KU KONKUK UNIVERSITY

DEPENDABLE SOFTWARE LABORATORY

Exercise 4: Identify Actors and Use Cases

- **Identify actors and use cases** for **the new OOO advanced digital watch**
  - Sketch a **use-case diagram** and **descriptions** for each use case, as detail as possible (casual format).
  - Use a UML tool
  - Each use case should link to user requirements defined at the Exercise 2.



UC01: Setting current time	
수준	사용자 목적
주요 액터	User
사전 조건	시스템이 동작 중이며, 현재 시간을 표시하고 있다. 알람이 울리고 있지 않은 상태이다.
사후 조건	시스템에 현재 시간에 대한 정보(연, 월, 일, 시, 분, 초, 요일)가 갱신된다.
주요 시나리오	
1. User는 시스템의 시간 정보를 설정하기 위해 A버튼을 입력한다.	2. 시스템은 시간 정보 항목 중 '초' 항목이 변경 가능하도록 선택하고, 해당 항목을 깜빡이게 출력한다.
3. User는 설정하려는 시간 정보의 항목을 선택하기 위해 C버튼을 입력한다.	4. 시스템은 다른 시간 정보 항목을 선택하고, 선택된 항목을 깜빡이게 출력한다.
User는 설정하려는 시간 정보 항목이 선택될 때까지 3-4를 반복한다.	
5. User는 선택한 항목의 값을 변경하기 위해 B버튼을 입력한다.	6. 시스템은 선택된 항목의 값을 증가시키고, 이를 화면에 출력한다.
User는 선택한 항목의 값이 변경하려는 값에 도달할 때까지 5-6을 반복한다.	
7. User는 원하는 시간 정보를 설정하였음을 확인하고 A버튼을 입력한다.	8. 시스템은 현재 시간 설정을 종료하고, 현재 시간을 출력한다.
확장 시나리오	
1-6a. 언제든지, User가 현재 시간 설정을 종료하기를 원하는 경우 1. User는 A버튼을 입력한다.	2. 시스템은 현재 시간 설정을 종료하고, 현재 시간을 출력한다.
5-6a. User가 선택한 항목의 값이 변경하려는 값을 초과한 경우 1. User는 선택한 항목의 값을 변경하기 위해 B버튼을 입력한다.	2. 시스템은 선택된 항목의 값을 증가시키고, 이를 화면에 출력한다.
User는 선택한 항목의 값이 최대값에 도달할 때까지 1-2를 반복한다. 3. User는 선택한 항목의 값이 최대값에 도달했음을 확인하고, B버튼을 입력한다.	4. 시스템은 선택된 항목의 값을 최소값으로 변경한다.
5. User는 선택한 항목의 값이 변경하려는 값에 도달할 때까지 1-2를 반복한다.	