

Requirements Engineering

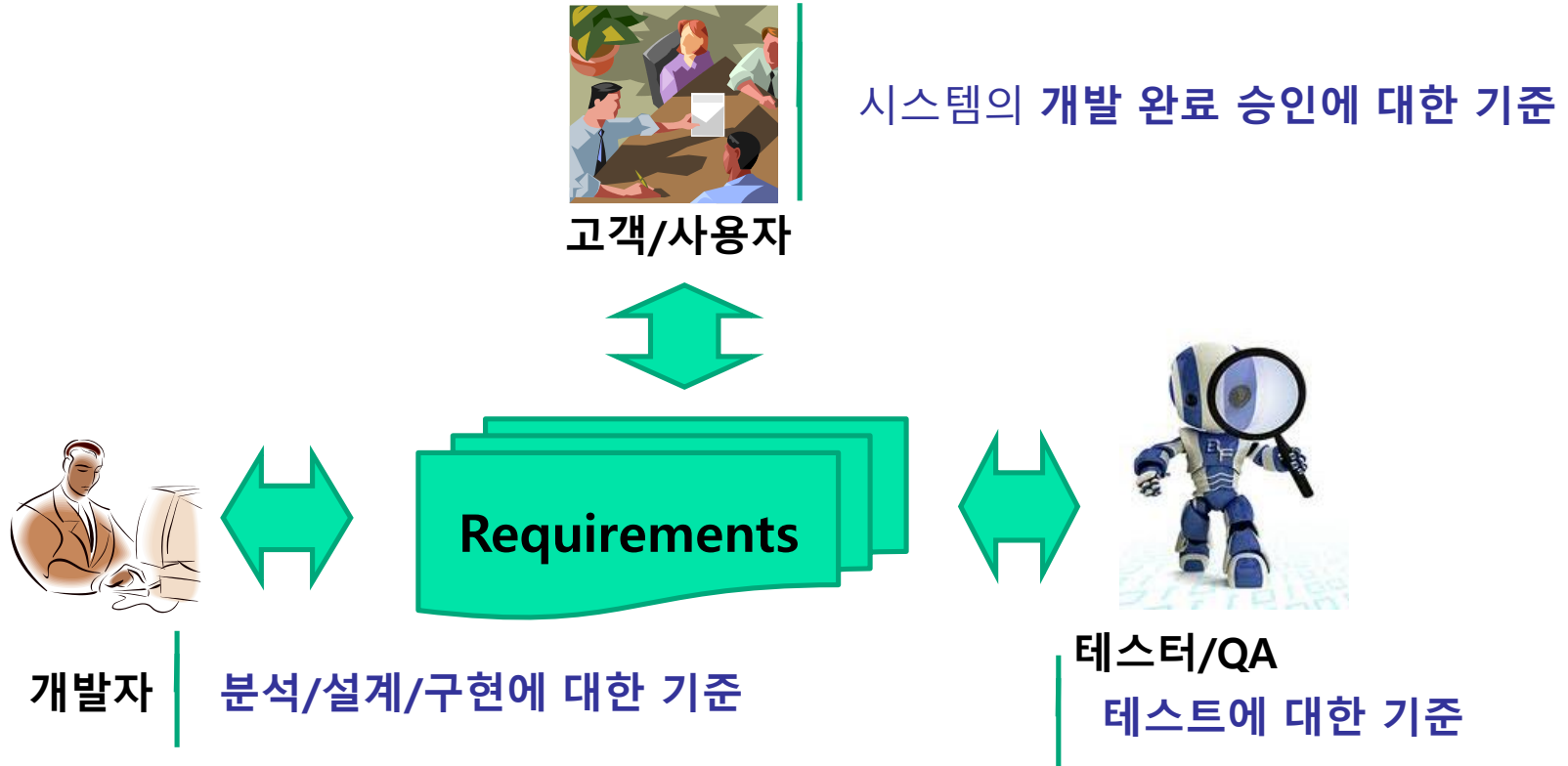
Heung-Seok Chae

Dept. of Computer Engineering

Pusan National University



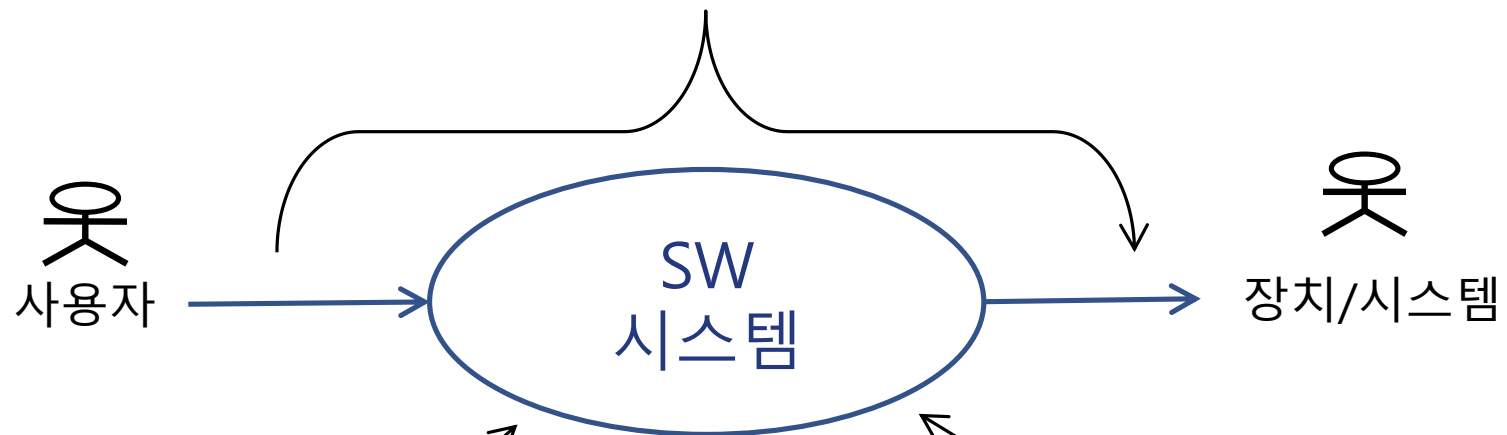
Role of Requirements



Requirement Type

기능적 요구사항

소프트웨어에 주어진 입력에 따른 동작 및 출력에 대한 요구사항



품질 요구사항

성능, 보안, 안전성, 가용성 등과 같은 기능적인 요구사항에 대한 제약

시스템 제약사항

외부 인터페이스에 대한 제약
법규, 표준 등에 대한 준수
설계, 구현, 설치, 운영 등에 대한 제약

Requirement Type

기능적 요구사항		소프트웨어에 주어진 입력에 따른 동작 및 출력에 대한 요구사항
비기능적 요구사항	품질 요구사항	성능, 신뢰성, 보안, 안전성 등과 같은 기능적인 요구사항에 대한 제약사항
	시스템 제약사항	외부 인터페이스에 대한 제약사항 법규, 표준 등에 대한 준수(compliance) 요구사항 설계, 구현, 설치, 운영 등에 대한 제약사항

기능적 요구사항

- 시스템에 주어지는 특정 입력에 대해서 시스템이 산출하는 출력에 의해서 정의된다.

식별자	입력	출력
FR-01	사용자가 휴대폰의 통화버튼을 누른다.	시스템은 최근 통화 목록을 표시한다. 첫 항목을 선택시킨다.
FR-02	사용자가 한번 더 통화버튼을 누른다.	시스템은 선택된 항목의 전화 번호로 통화를 시도한다. 통화 연결음을 들려 준다.
FR-03	사용자가 통화 중에 종료버튼을 누른다.	시스템은 연결을 종료시킨다.
FR-04	사용자가 종료버튼을 2초동안 누른다.	시스템은 휴대폰을 종료시킨다.

비기능적 요구사항: 품질요구사항

■ 대표적인 품질 요구사항

- 성능(performance): 시스템의 자원(CPU, 메모리 등)을 얼마나 효율적으로 사용하는가? 즉 사용자 입력에 대하여 얼마나 빠른 시간에 얼마나 적은 자원을 활용해서 결과를 출력할 수 있는가?
- 신뢰성(reliability): 시스템이 주어진 요구사항을 준수하여 동작하는 정도를 뜻한다. 일반적으로 장애 없이 동작하는 시간의 비율로서 정의된다.
- 보안성(security): 허가되지 않은 사용자가 시스템에 접근하거나, 사용자가 접근 권한이 없는 시스템의 정보를 접근하거나 해서는 안된다. 보안성은 이러한 측면에 대한 요구사항을 뜻한다.
- 안전성(safety): 시스템이 주변 환경, 인명, 재산에 피해를 주지 않아야 한다는 요구사항이다.
- 가용성(availability): 사용자가 원하는 순간에 시스템은 서비스를 제공해야 한다는 요구사항이다.

품질요구사항의 예

성능	<ul style="list-style-type: none">● 사용자가 통화버튼을 누르면 2초 이내에 통화 연결이 성립되어야 한다.● 통화 연결 기능은 20KB 이내의 메모리만을 사용해야 한다.● 엘리베이터 버튼을 누르면 해당 층으로 10초 이내에 엘리베이터가 이동해야 한다.● 도서 검색은 3초 이내에 검색 결과를 화면으로 출력해야 한다.
신뢰성	<ul style="list-style-type: none">● 지대공미사일은 100개를 발사하면 90개는 목표에 명중해야 한다.● 100 통화 연결을 시도하면 95번은 성공해야 한다.
보안성	<ul style="list-style-type: none">● 등록된 사용자 만이 시스템에 접근할 수 있어야 한다.● 보안등급이 A, B인 정보는 OO, OO 그룹의 등록된 사용자 만이 접근할 수 있어야 한다.● 네트워크를 통하여 송수신되는 데이터가 노출되어서는 안 된다.● DoS 공격을 실시간으로 탐지하여 회피할 수 있어야 한다.
안전성	<ul style="list-style-type: none">● 엘리베이터 문이 열린 상태에서는 엘리베이터는 이동해서는 안 된다.● 엘리베이터가 이동 중일 때는 문이 열려서는 안 된다.● 원자로의 온도와 압력이 정상 범위를 벗어나면 자동으로 원자로를 종료(shutdown) 시켜야 한다.
가용성	<ul style="list-style-type: none">● 인터넷 뱅킹 시스템은 1년 365, 하루 24시간 동안 서비스를 제공해야 한다.● 수강신청시스템은 수강신청 기간 동안에는 중단 없이 서비스를 제공해야 한다.

Quality: Performance

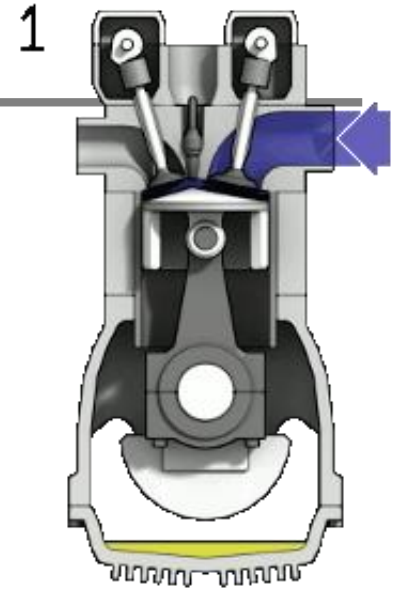
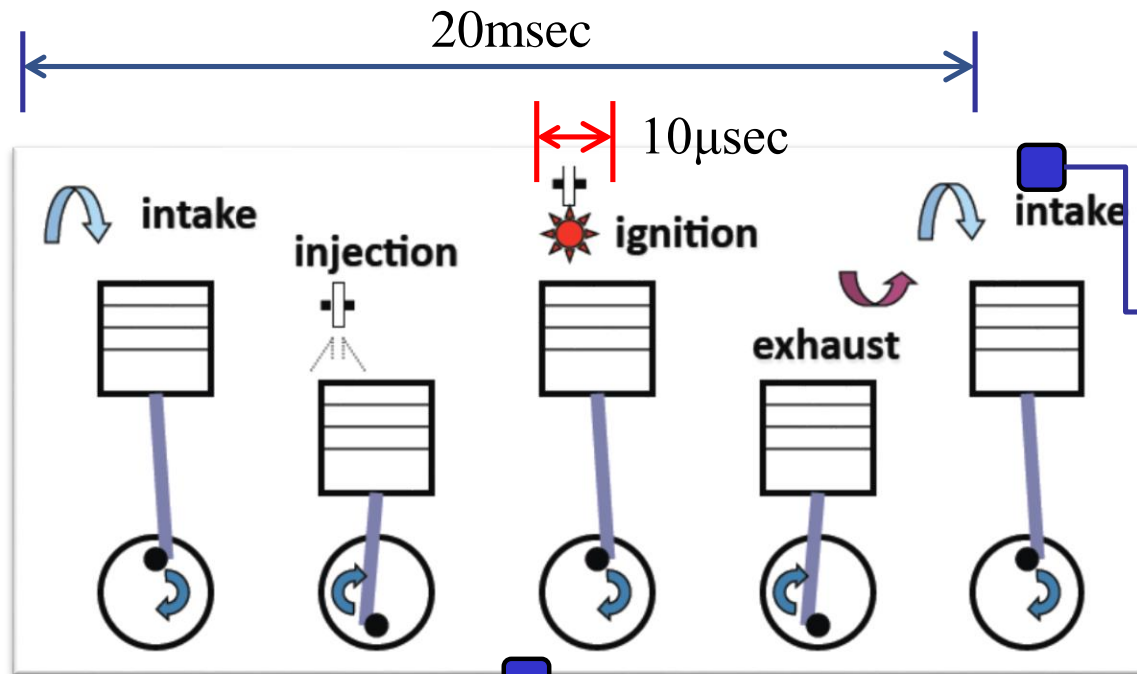
- A requirement that defines the extent or how well, and under what conditions, a function or task is to be performed.

- These are quantitative requirements of system performance and are verifiable individually.
 - Response time for a transaction(average, maximum)
 - Throughput (for example, transactions per second)
 - Capacity (for example, the number of customers or transactions the system can accommodate)
 - Degradation modes (what is the acceptable mode of operation when the system has been degraded in some manner)
 - Resource use: memory, disk, communications, and so forth

Quality: Performance

■ Example

- In the case of 6000rpm, one cycle is 20msec.
- Timing precision of the ignition is $10\mu\text{sec}$. order



The calculation of the fuel injection volume must be finished

The calculation of the ignition timing must be finished



Quality: Reliability

- The probability of failure-free software operation for a specified period of time in a specified environment
- Failure: An unacceptable effect or behavior under permissible operating conditions.
- Reliability can be quantitatively evaluated
 - Mean Time Between Failures (MTBF)
 - Mean Time To Repair (MTTR)
 - Accuracy – specify precision (resolution) and accuracy (by some known standard) that is required in the systems output.
 - Maximum bugs or defect rate – usually expressed in terms of bugs/KLOC (thousands of lines of code), or bugs/function-point.

Quality: Robustness

- The degree to which a system or component can function correctly in the presence of **invalid inputs** or **stressful environment conditions** - IEEE Standard Glossary of Software Engineering Terminology. IEEE, IEEE Std 610.12-1990
- Robust if it behaves reasonably, even in circumstances that were not anticipated in the requirements specification.

Quality: Robustness

- Example: Video Conferencing Systems (VCS)
 - The robustness behavior of a VCS in the presence of hostile environment conditions (regarding the network and other communicating VCSs), such as a high percentage of packet loss and corrupt packets.
 - The VCS should not crash, halt, or restart in the presence of such problems.
 - Furthermore, the VCS should continue to work in a degraded mode, such as continuing the video-conference with low audio and video quality.

Quality: Safety

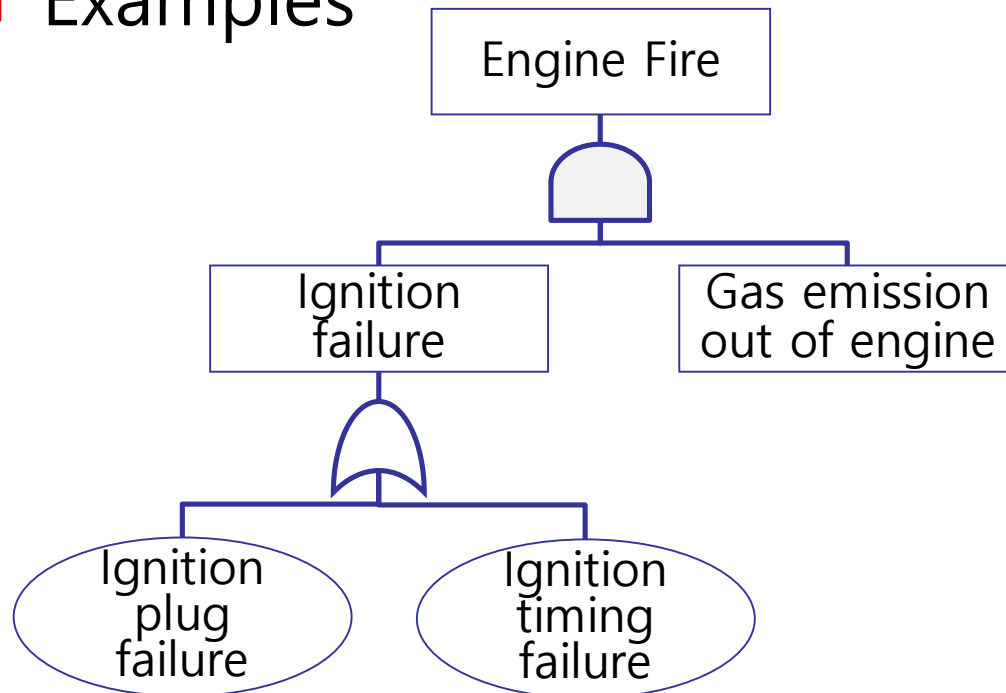
- Failure may cause injury or death to human beings and damage to environment
- e.g. an aircraft or nuclear power station control system



Quality: Safety

Freedom from those conditions that can cause **death, injury**, occupational illness, or damage to or loss of equipment or property, or **damage to the environment** [MIL-STD-882C]

■ Examples



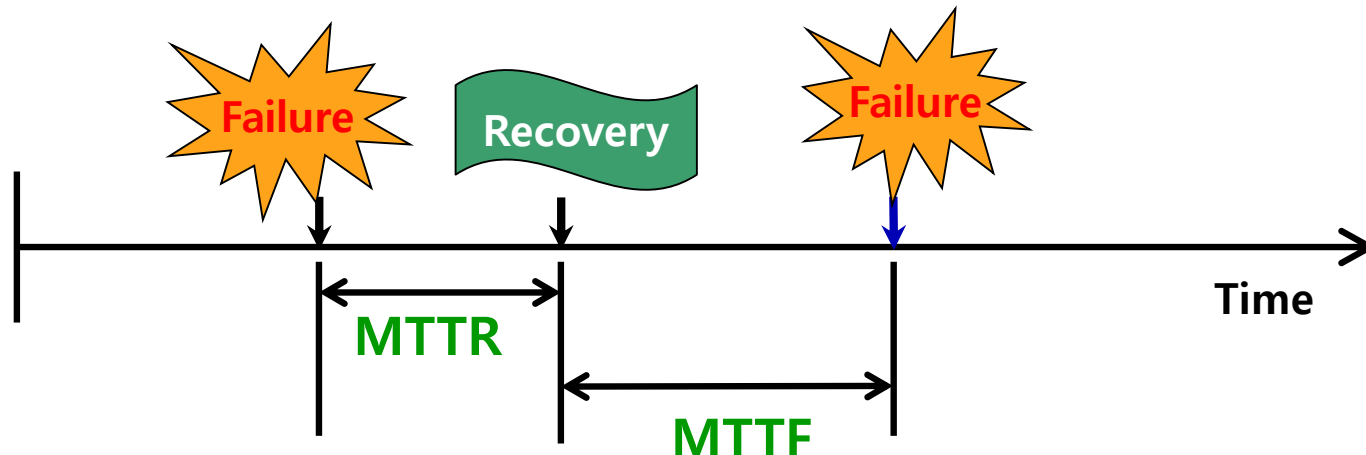
Quality: Availability

The degree to which a system or component is **operational and accessible when required for use** [IEEE 610]

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

MTTF : Mean Time To **F**ailures

MTTR : Mean Time To **R**epair



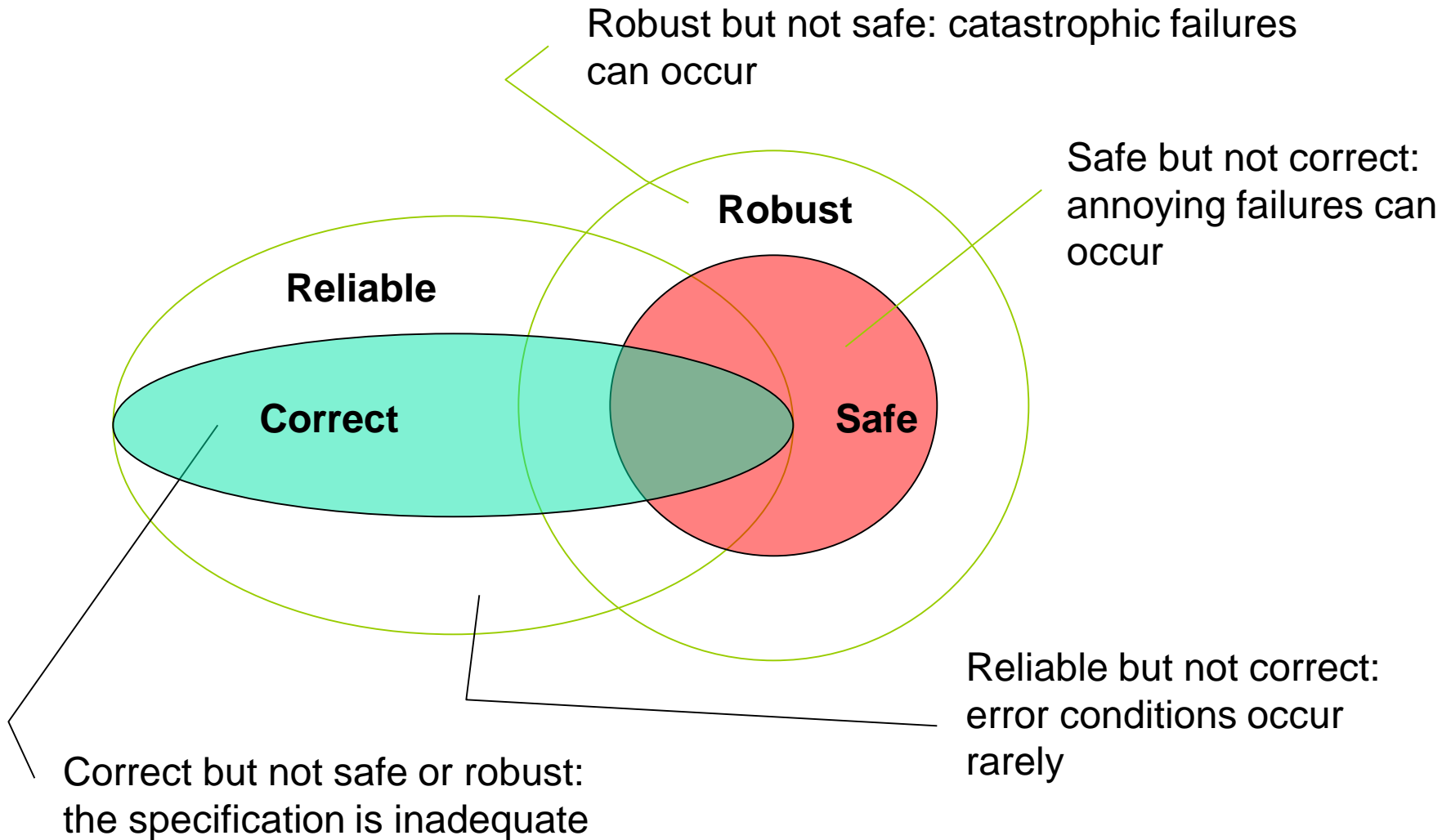
Quality: Availability

가용성	연간 장애 시간	주당 장애 시간
98%	7.3일	3시간 22분
99%	3.65일	1시간 41분
99.8%	17시간 30분	20분 10초
99.9%	8시간 45분	10분 5초
99.99%	52분 30초	1분
99.999%	5분25초	6초

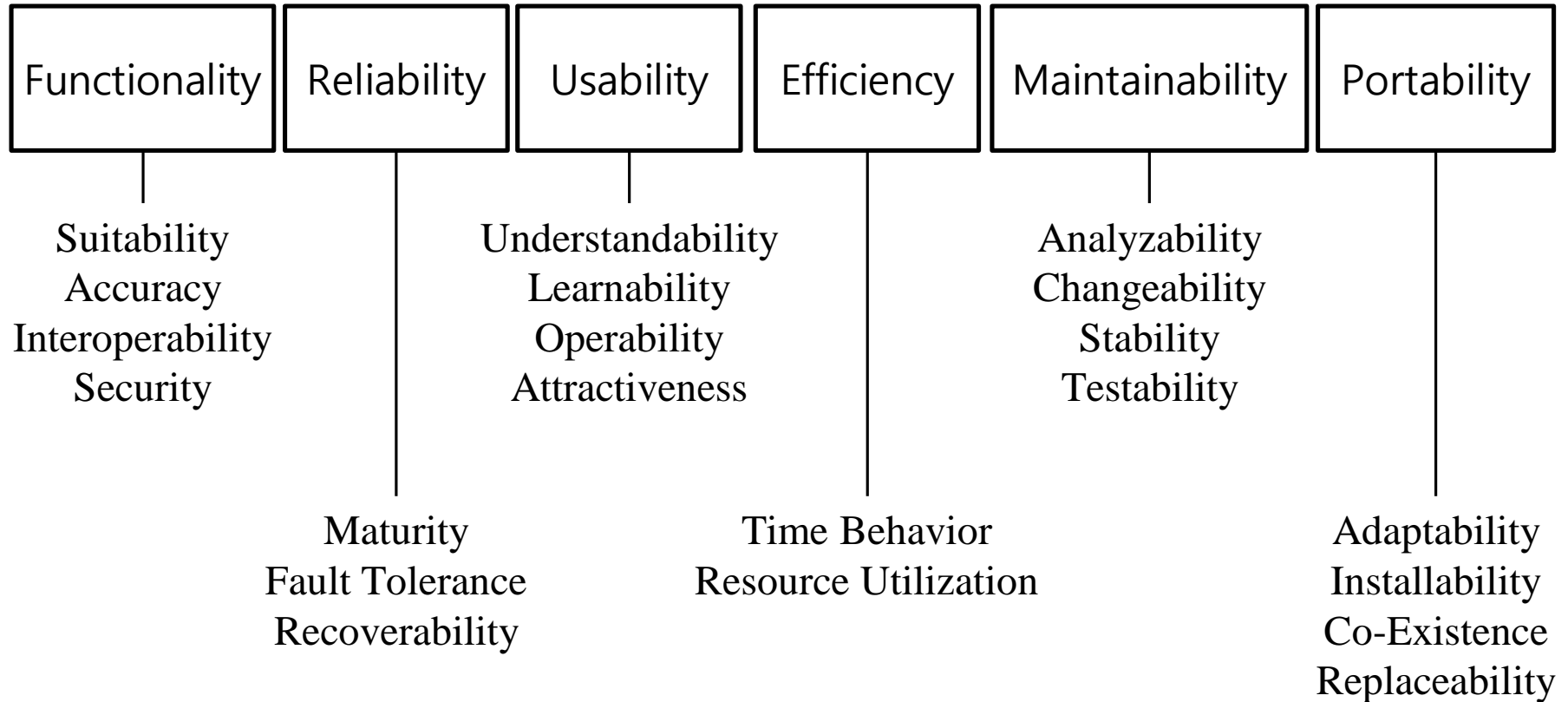
Monthly Uptime Percentage	Service Credit Percentage
99.0% ~ 99.95%	10%
~ 99.0%	30%

<http://aws.amazon.com/ko/ec2-sla/>

Relationship between Quality Factors

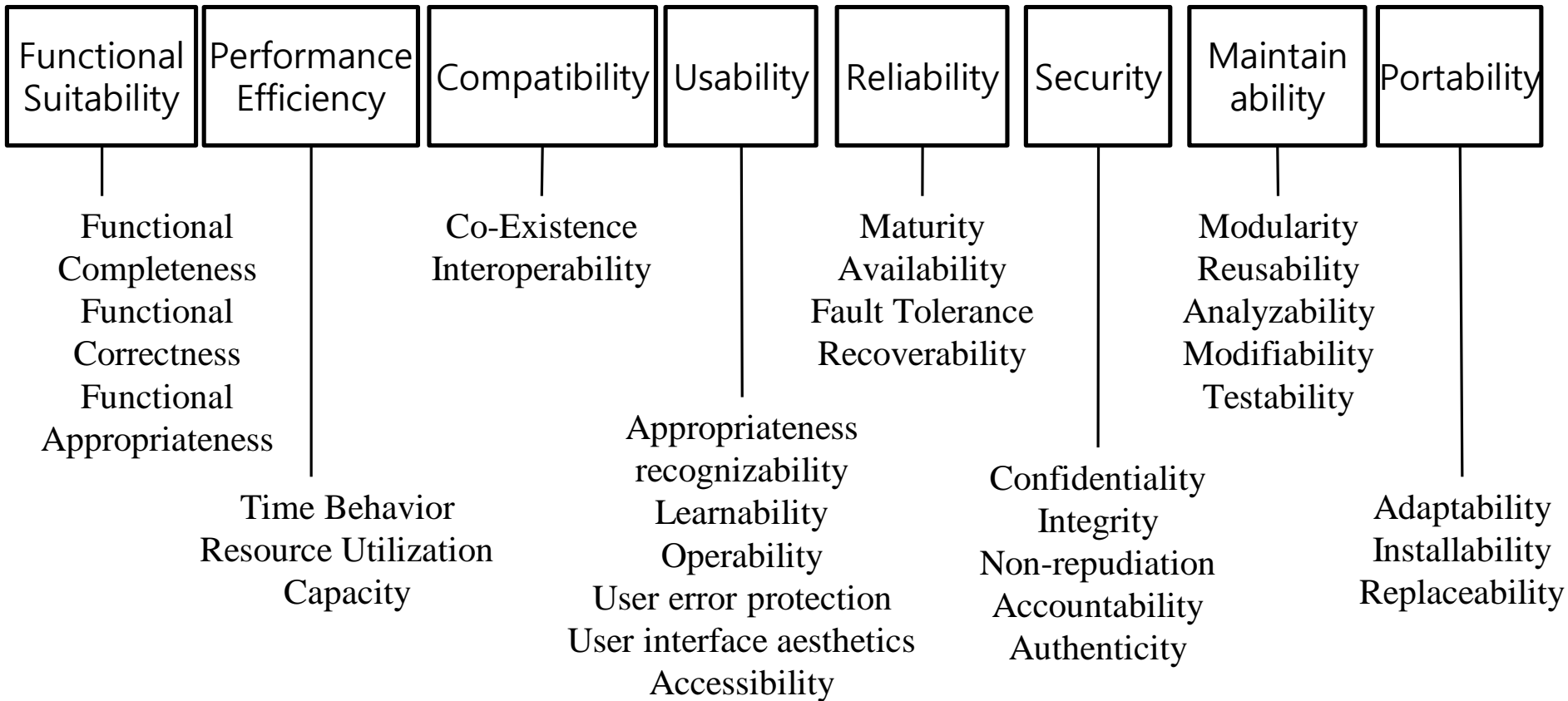


ISO 9126-1:2001 Quality Model



ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model

ISO/IEC 25010:2011 Quality Model



ISO/IEC 25010:2011 Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models

Constraints

- A constraint is fixed premade decisions before design begins
 - Business constraints limit decisions about people, process, costs, and schedule.
 - Technical constraints limit decisions about the technology we may use in the software system; Externally imposed limitation on system requirements, design, or implementation or on the process used to develop or modify a system

Constraints: Examples

Technical Constraints	Business Constraints
Programming Language Choice Anything that runs on the JVM.	Team Composition and Makeup Team X will build the XYZ component
Operating System or Platform It must run on Windows, Linux, and BeOS.	Schedule or Budget It must be ready in time for the Big Trade Show and cost less than \$80,000.
Use of Components or Technology We own DB2 so that's your database.	Legal Restrictions There is a 5GB daily limit in our license

PROBLEMS WITH REQUIREMENTS

What Are the Problems with Requirement Engineering?

- Can we start up programming with the following statements ?

공통 요구사항

1. 주기적으로 온도센서를 통하여 현재 온도를 구한다.
2. 현재 온도와 희망 온도를 바탕으로 냉난방제어장치를 제어한다.

건물주 요구사항

- 희망 온도: 20도 – 28도
- 월 전기 요금은 000 이하이어야 한다.

세입자 희망 온도: 22도 – 25도

Unambiguous Requirement



How the customer explained it



How the PL understood it



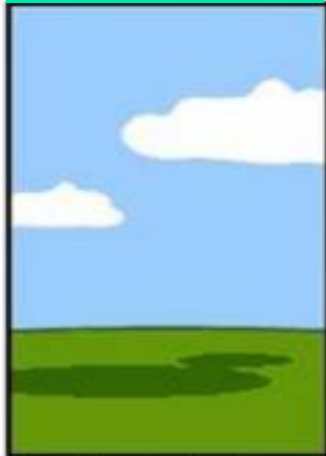
How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



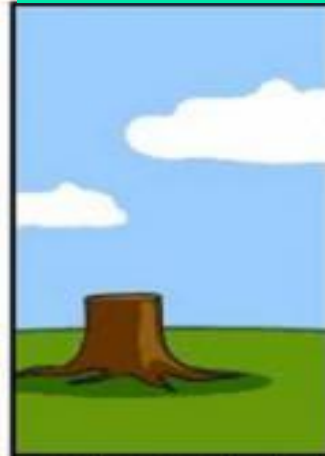
How the project was documented



What operations installed



How the customer was billed



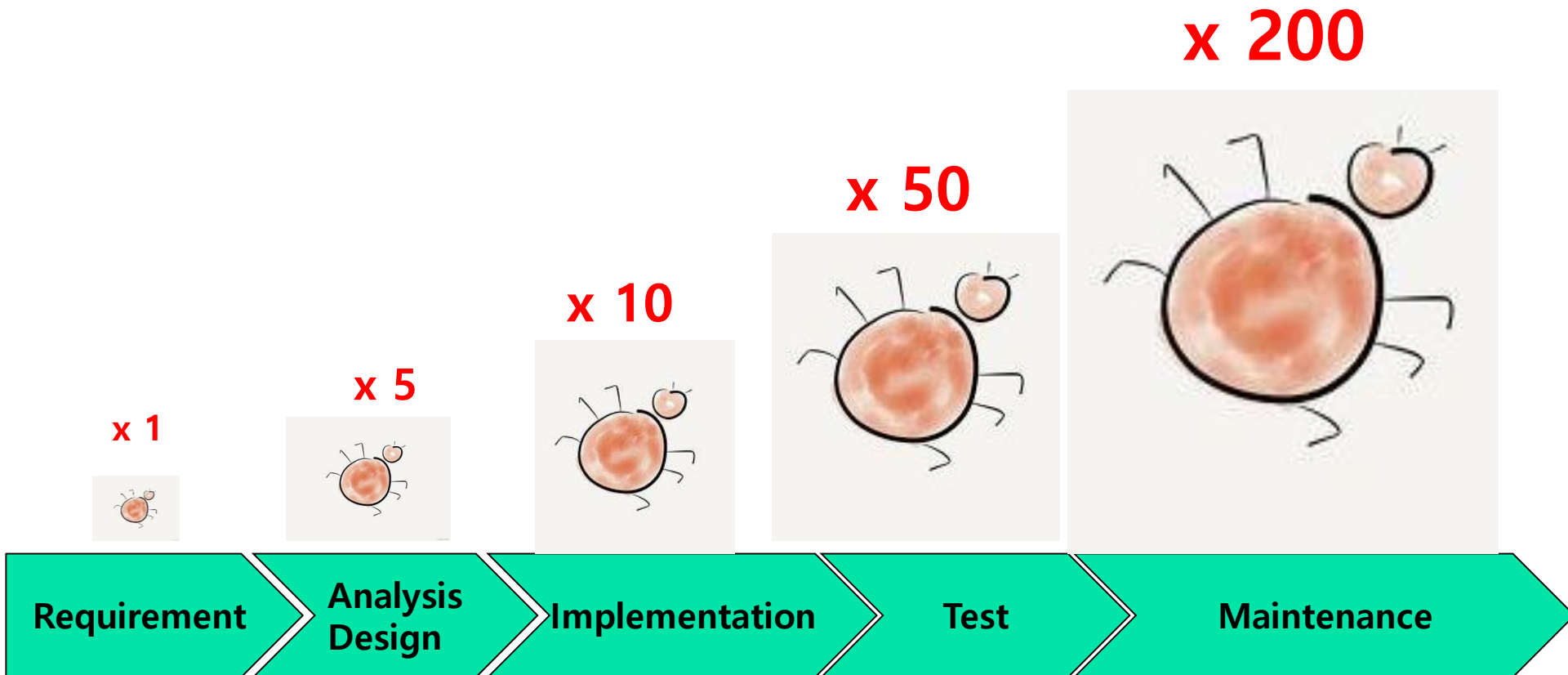
How it was supported



What customer really needed

Defect Grows !

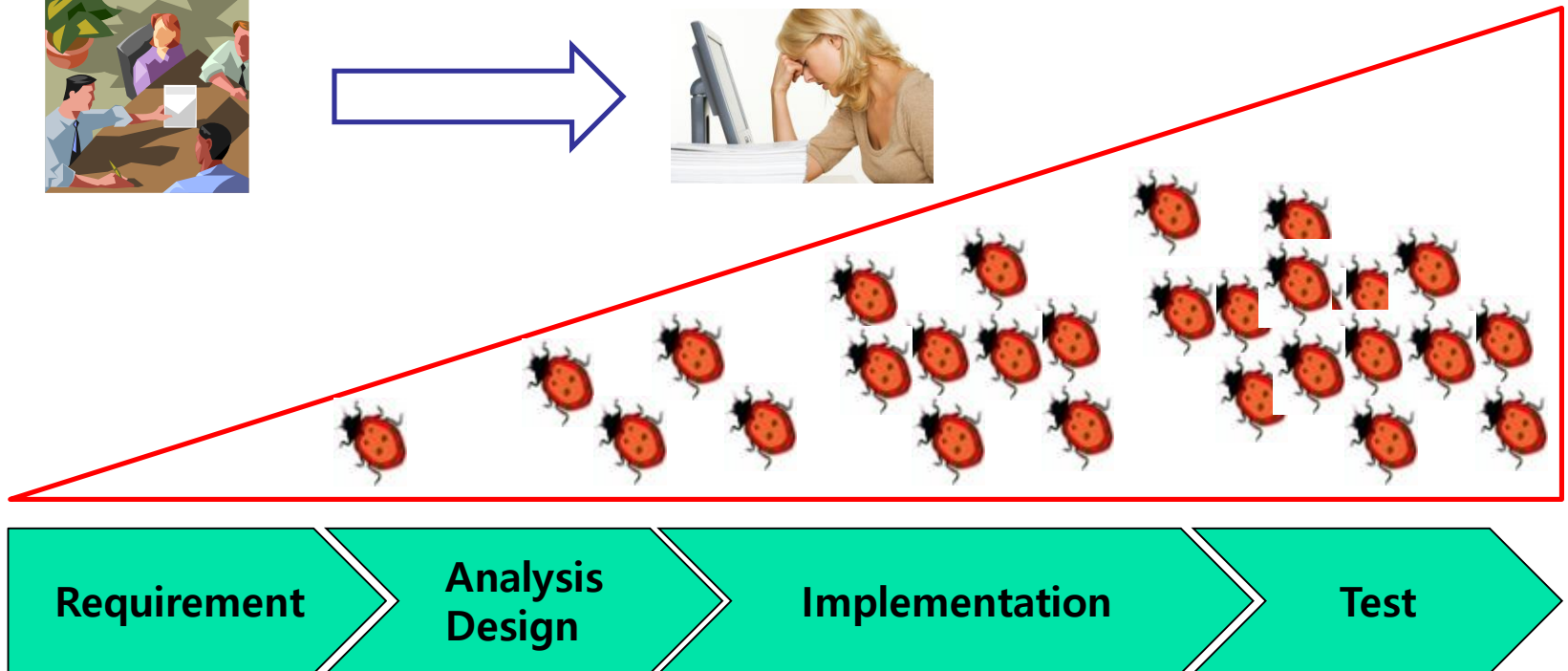
- Unresolved defects require more cost later



More Changes Cause More Defects

- It is inevitable for developer to make errors because of complex activities and tight schedule.

Requirement changes can lead to more defects



Success Factors of Software Projects

Successful Projects	% of Responses	Challenged Projects	% of Responses
User involvement	15.9	Lack of user input	12.8
Executive management support	13.9	Incomplete requirements	12.3
Clear statement of requirements	13.0	Changing requirements	11.8
Proper planning	9.6	Lack of executive support	7.5
Realistic expectations	8.2	Technology incompetence	7.0
Smaller project milestones	7.7	Lack of resources	6.4
Competent staff	7.2	Unrealistic expectations	5.9
Ownership	5.3	Unclear objectives	5.3
Clear vision and objectives	2.9	Unrealistic time frames	4.3
Hard-working, focused staff	2.4	New technology	3.7
other	13.9	other	23.0

Source: The Standish Group, "Chaos," 1995

DESIRABLE PROPERTIES OF REQUIREMENTS

Desirable properties of requirements

Unambiguous	It has only one interpretation
Correct	It is one that stakeholder really wants
Complete	It includes all the requirements
Consistent	no subset has conflict
Feasible	It is technically achievable
Traceable	It traces to its source and implementation

Requirements: Example

Ambiguous: 1분 ? 10분?

공동 요구사항

1. 주기적으로 온도센서를 통하여 현재 온도를 구한다.
2. 현재 온도와 희망 온도를 바탕으로 냉난방제어장치를 제어한다.

Incomplete: 온도가 너무 높을 때는 화재 경보 발생 기능이 누락됨

건물주 요구사항

- 희망 온도: 20도 - 28도
- 월 전기 요금은 000 이하이어야 한다.

Incorrect

실제 건물주의 요구사항: 18도 - 28도

세입자 희망 온도: 22도 - 25도

Inconsistent: 건물주의 희망 온도와 상충됨

Infeasible: 전력요금에 영향을 받으므로 구현 불가능일 수가 있음

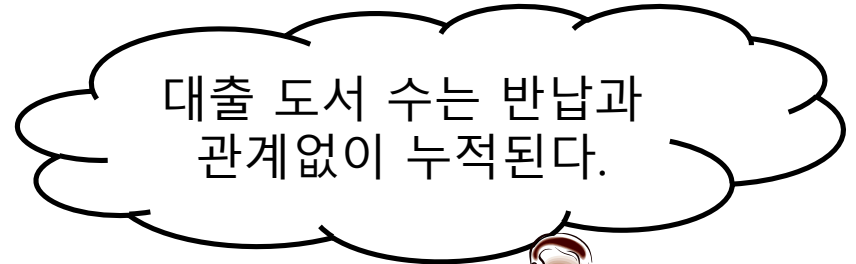
명확성

- 요구사항은 항상 동일한 의미로 해석되어야 한다. 즉 모호하지 않아야 한다.
- 모호한 요구사항의 예



고객/사용자

반납된 도서 수는 제외된다.



대출 도서 수는 반납과 관계없이 누적된다.



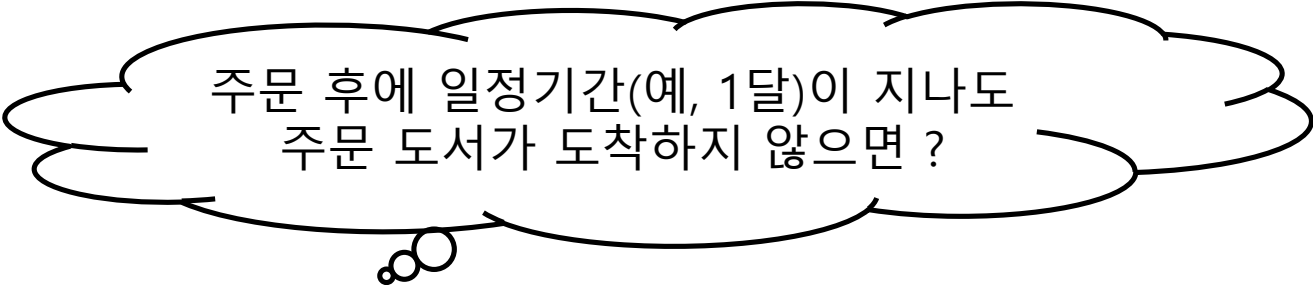
개발자

학생은 총 10권까지 대출할 수 있다.

- YT에게 작업 지시를 한 후 작업 수행 상태를 출력한다.

완전성

- 사용자가 기대하는 모든 기능/비기능적 요구사항이 기술되어야 한다. 즉 누락되어서는 안 된다.
- 불완전한 요구사항의 예



주문 후에 일정기간(예, 1달)이 지나도
주문 도서가 도착하지 않으면 ?

주문한 도서가 도착하면 주문자에게 이를 통보한다.

일관성

- 서로 상충되는 요구사항이 있어서는 안 된다.
- 상충되는 요구사항의 예


R1: 사용자는 시스템의 모든 기능을 이용하기에 앞서 시스템에 로그인을 해야 한다.



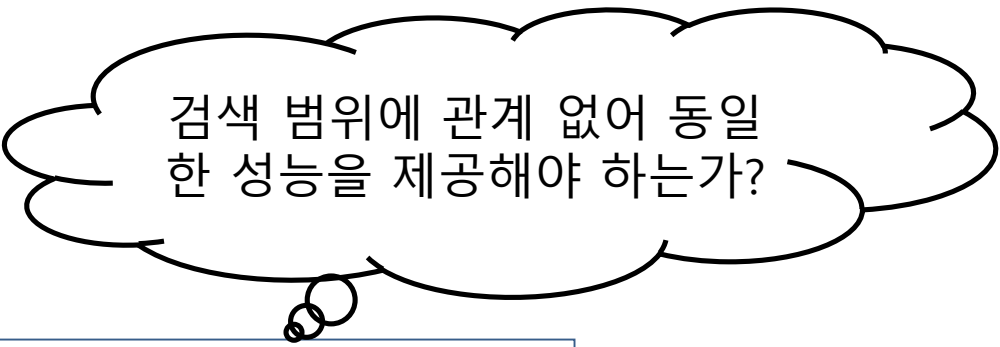
R2: 도서 검색은 로그인을 하지 않고도 가능하다.

검증가능성

- 객관적으로 검증할 수 있도록 구체적이어야 한다.
- 검증이 어려운 요구사항의 예



"빠른"의 기준은 무엇인가?



검색 범위에 관계 없어 동일한 성능을 제공해야 하는가?

도서 검색은 최대한 빨리 수행되어야 한다.

구현가능성(Feasibility)

- 가용한 기술과 한정된 일정/비용으로 구현이 가능해야 한다.
- 구현이 어려운 요구사항의 예
 - 엘리베이터 버튼을 누른 후 임의의 층으로 5초 이내에 도착해야 한다.
 - 1억 개의 컨테이너에 대한 야적 위치 결정을 2초 이내에 완료해야 한다.

Question

■ 다음 요구사항을 평가하십시오

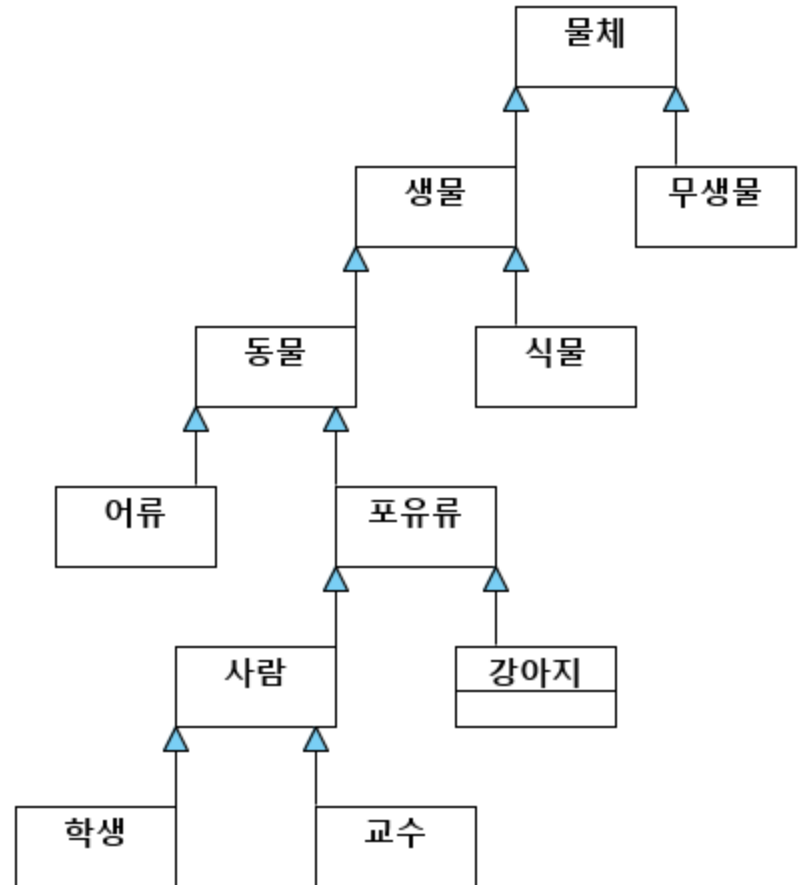
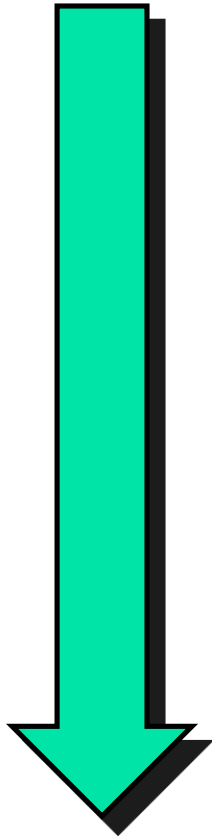
ECS는 건물에 설치된 복수 개의 엘리베이터를 제어한다.

- Req1: 탑승자는 엘리베이터 내부에서 버튼을 눌러서 목적지층을 선택한다.
- Req2: 대기자는 층에서 버튼을 눌러서 엘리베이터를 요청한다.
- Req3: ECS는 5초 이내에 해당 층으로 엘리베이터를 이동시킨다.

- 명확성:
- 완전성:
- 구현가능성:

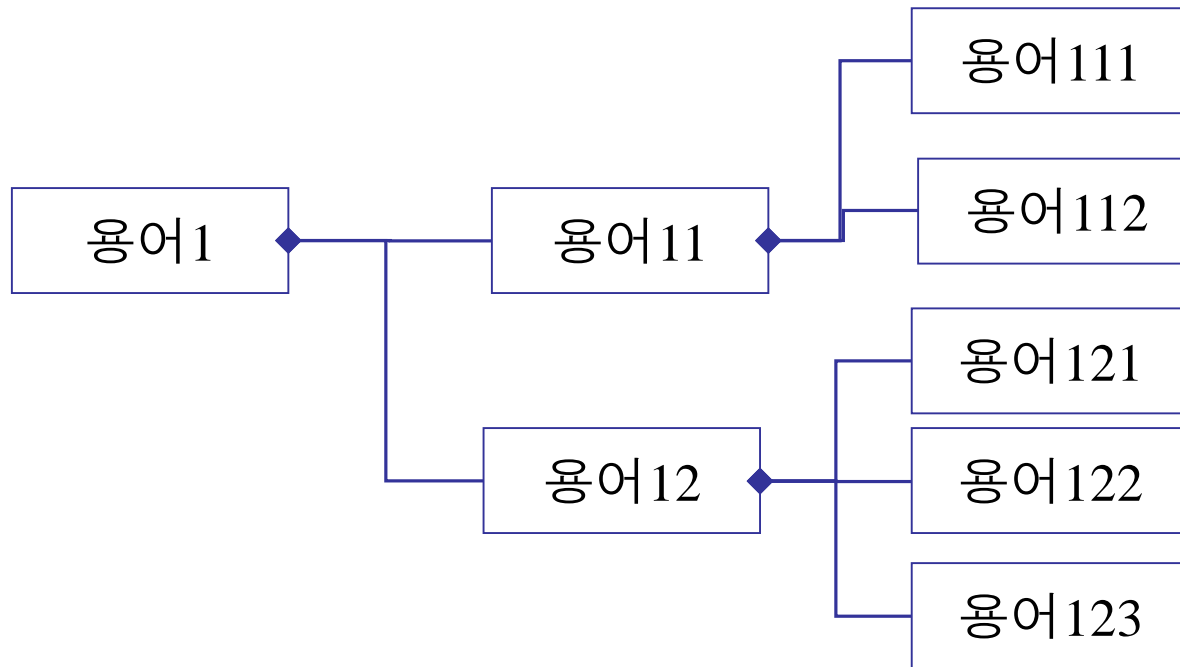
보다 구체적인 용어의 사용

- 1) 일반화 관계를 활용: 보다 구체적인(즉 하위 클래스) 용어 사용



보다 구체적인 용어의 사용

- 2) 포함 관계를 활용: 보다 세부적인(즉 부분 클래스) 용어 사용



도메인의 용어만을 사용

- 3) 도메인의 용어 사전을 활용
 - SW는 새로운 개념을 창조하지 않음
 - 사용자/고객이 이해할 수 있는 도메인의 용어만을 사용해야 함
 - 개발자들의 용어를 사용하지 않아야 함

Terms to avoid in Requirements (1/4)

Ambiguous Terms	Description
acceptable, adequate	Define what constitutes acceptability and how the system can judge this.
as much as practicable	Don't leave it up to the developers to determine what's practicable.
at least, at a minimum, not more than, not to exceed	Specify the minimum and maximum acceptable values.
between	Define whether the end points are included in the range.
depends on	Describe the nature of the dependency. Does another system provide input to this system, must other software be installed before your software can run, or does your system rely on another one to perform some calculations or services?



Terms to avoid in Requirements (2/4)

Ambiguous Terms	Description
efficient	Define how efficiently the system uses resources, how quickly it performs specific operations, or how easy it is for people to use.
flexible	Describe the ways in which the system must change in response to changing conditions or business needs.
improved, better, faster, superior	Quantify how much better or faster constitutes adequate improvement in a specific functional area
maximize, minimize, optimize	State the maximum and minimum acceptable values of some parameter.
optionally	Clarify whether this means a system choice, a user choice, or a developer choice.

Terms to avoid in Requirements (3/4)

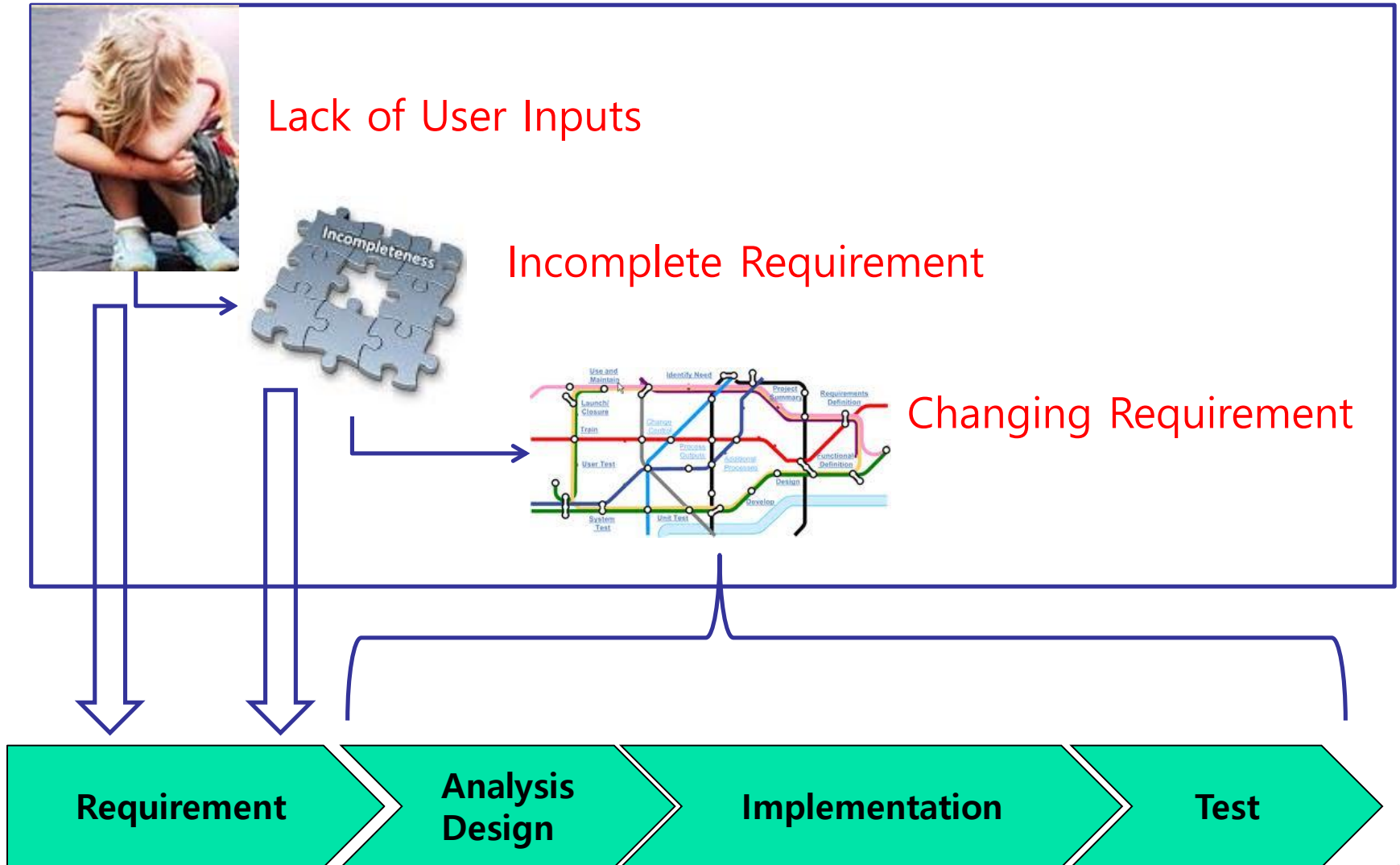
Ambiguous Terms	Description
reasonable, when necessary, where appropriate	Explain how to make this judgment.
robust	Define how the system is to handle exceptions and respond to unexpected operating conditions
seamless, transparent, graceful	Translate the user's expectations into specific observable product characteristics.
several	State how many, or provide the minimum and maximum bounds of a range.
shouldn't	Try to state requirements as positives, describing what the system will do

Terms to avoid in Requirements (4/4)

Ambiguous Terms	Description
state-of-the-art	Define what this means.
sufficient	Specify how much of something constitutes sufficiency.
support, enable	Define exactly what functions the system will perform that constitute supporting some capability.
user-friendly, simple, easy	Describe system characteristics that will achieve the customer's usage needs and usability expectations.

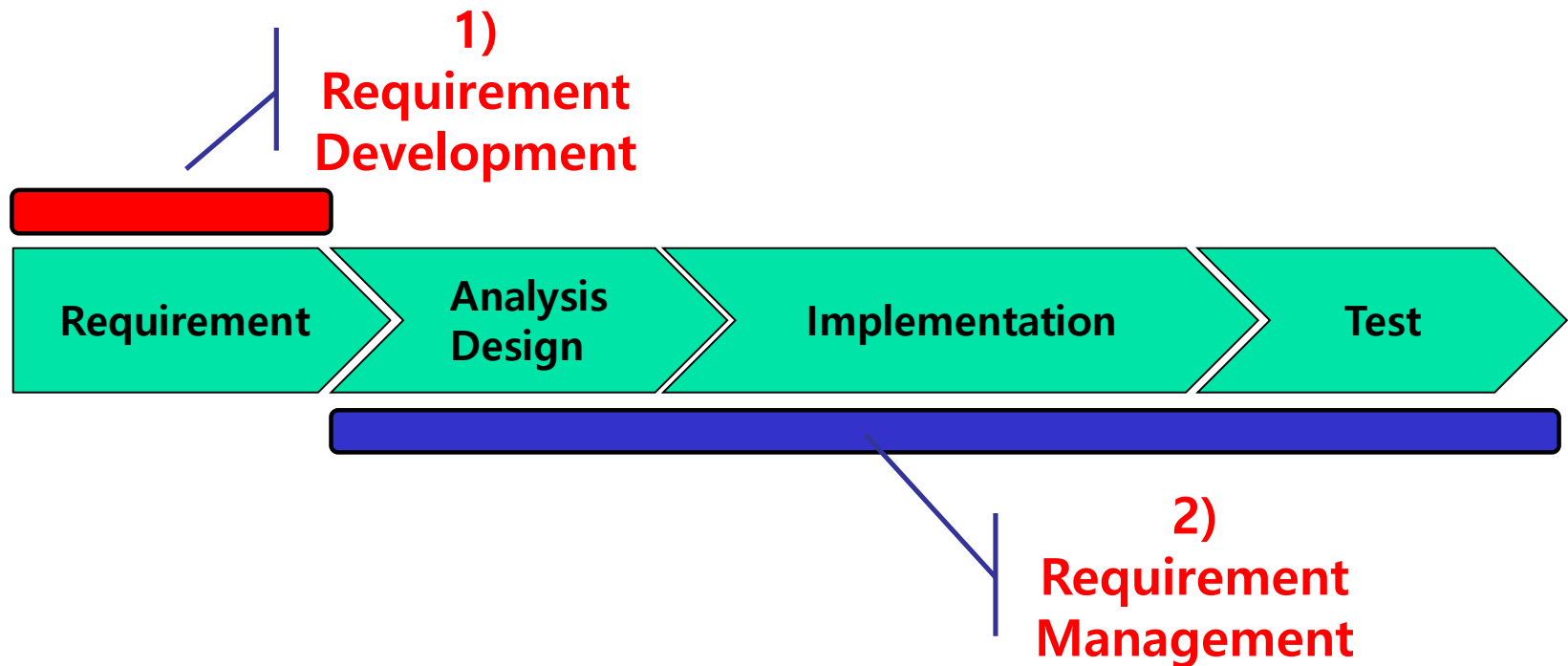
REQUIREMENT ENGINEERING

Problems with Requirements

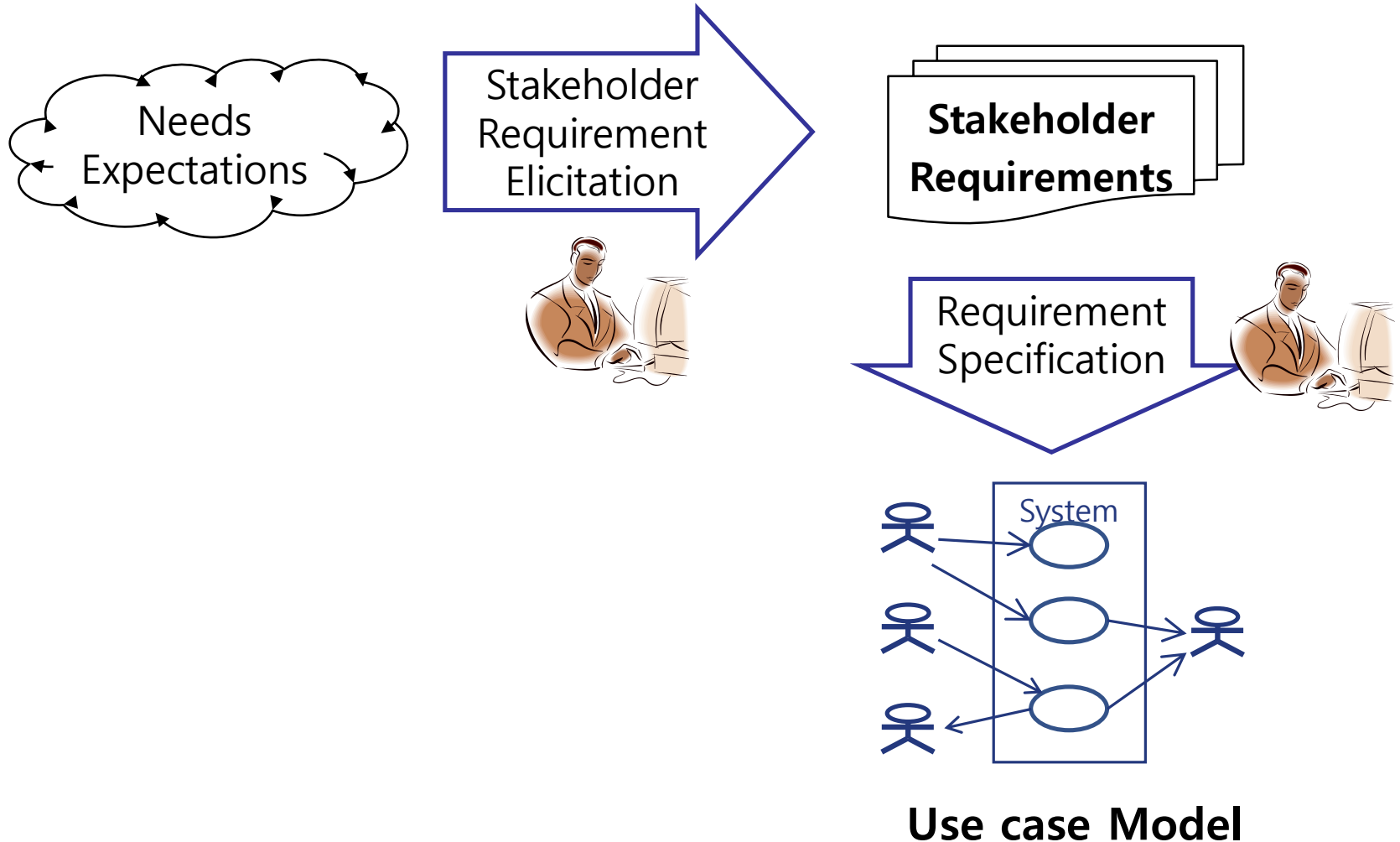


How to Manage the Problems?

- User/Stakeholder involvement is essential for successful project!

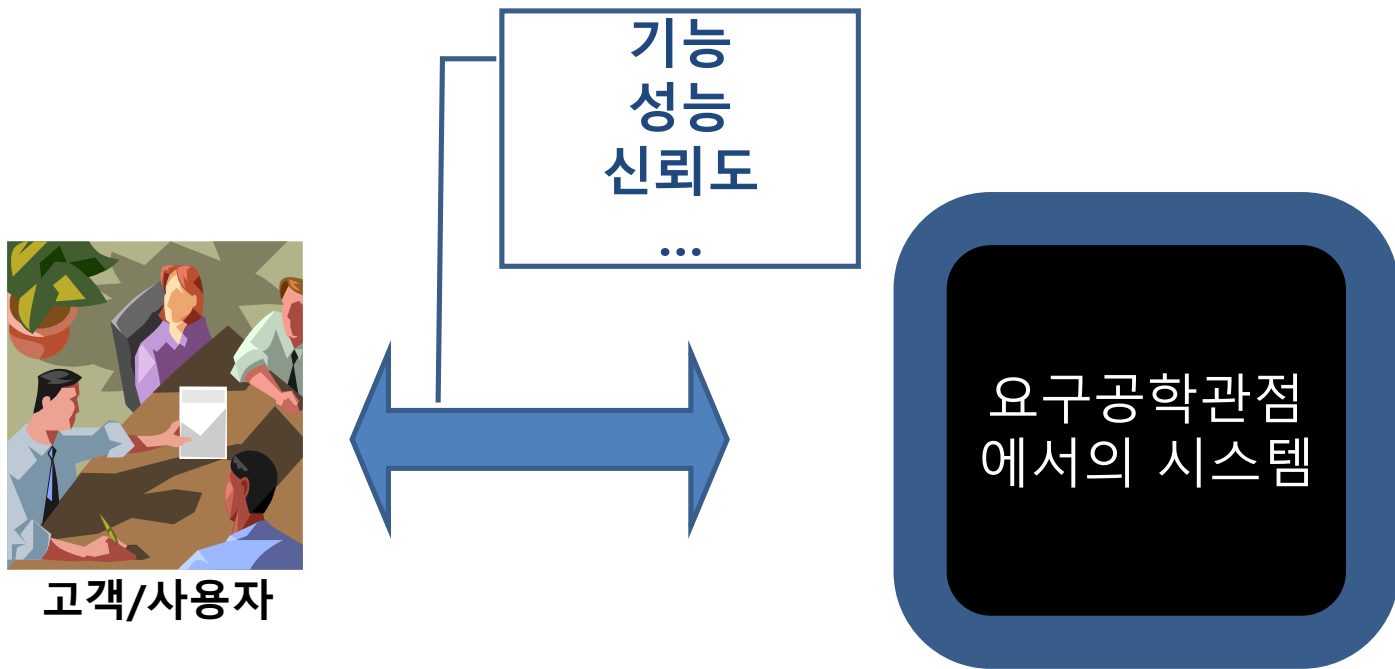


Requirement Development



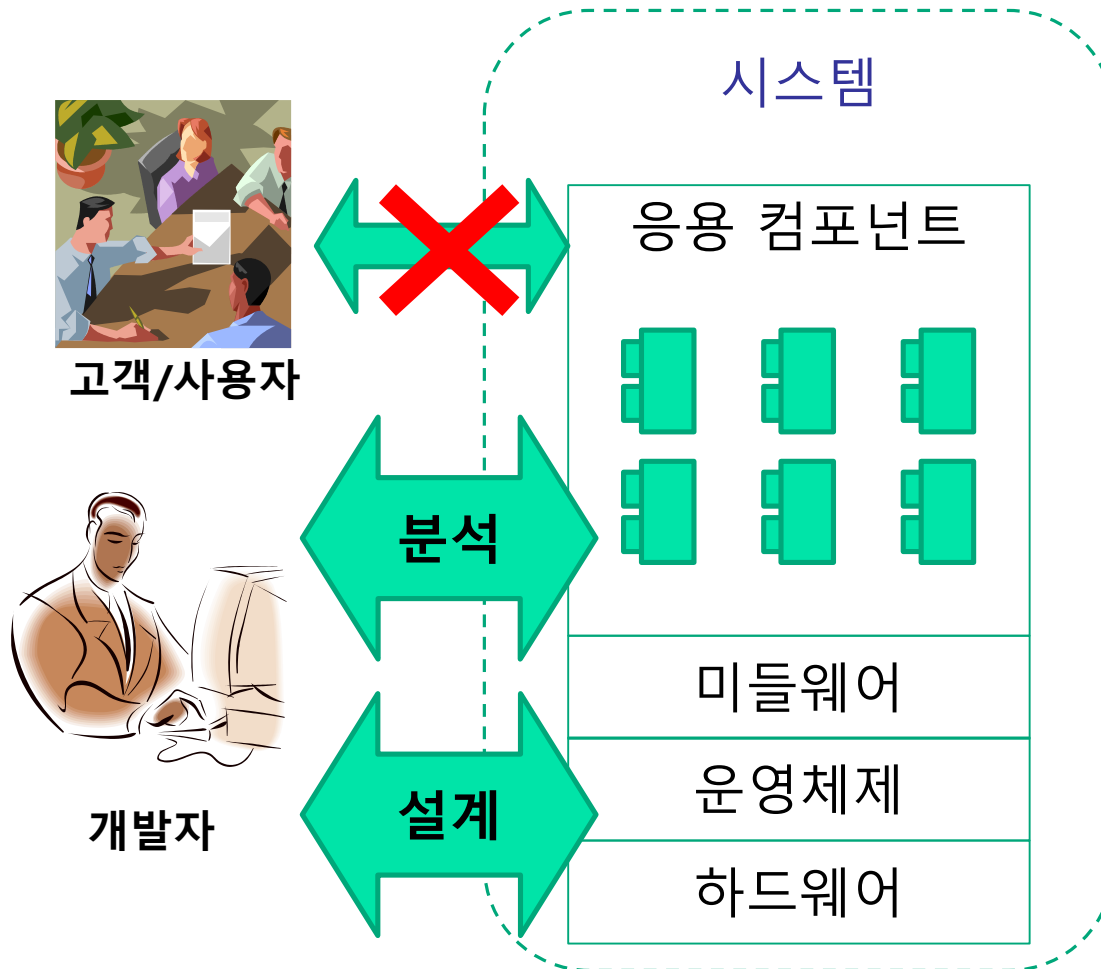
요구사항 정의 단계의 특징 (1/2)

- 요구사항 정의 관점에서의 시스템
 - It is NOT a design.
 - It should set **WHAT** the system should do rather than **HOW** it should do it
 - It should focus on **externally observed behaviors/features**



요구사항 정의 단계의 특징 (2/2)

■ 분석/설계 관점에서의 시스템



Q&A
