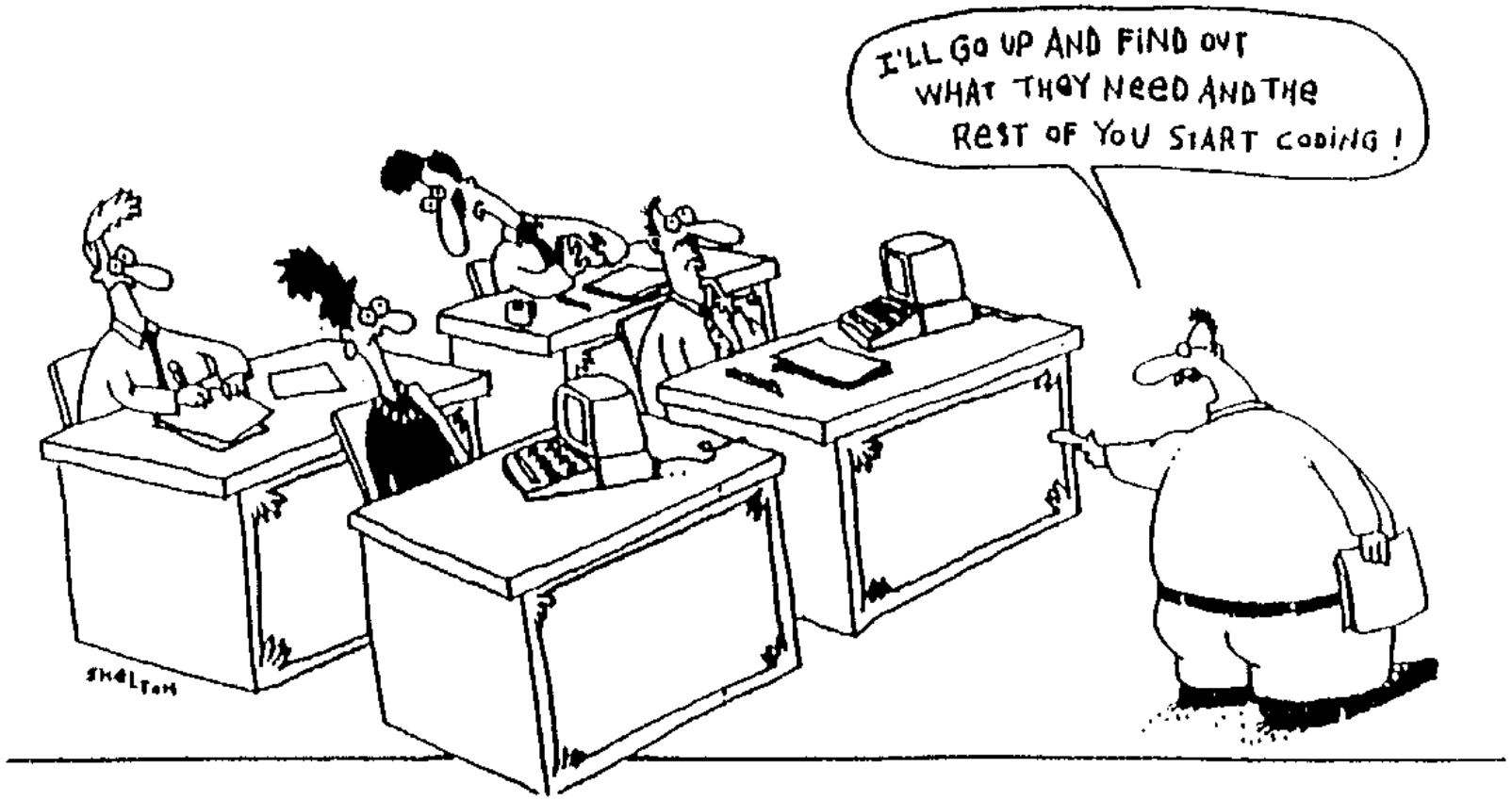
요구공학
Requirements Engineering

JUNBEOM YOO
KONKUK UNIVERSITY
http://dslab.konkuk.ac.kr

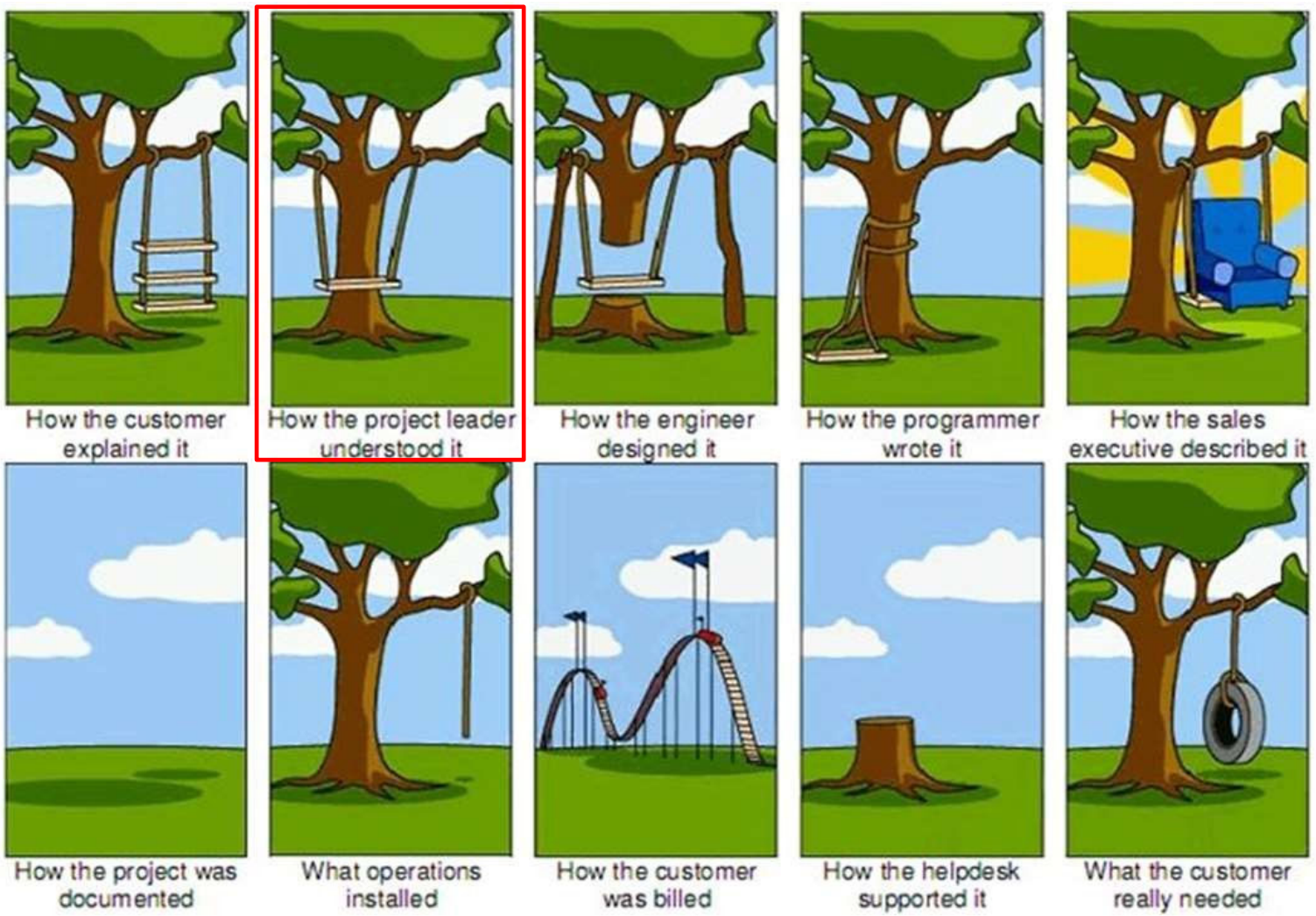
과정 구성

1. Requirements Engineering - Overview
2. Requirements
3. Feasibility Study
4. Requirements Elicitation
   - Exercise 1: Requirements Elicitation
5. Requirements Negotiation
   - Exercise 2: Requirements Negotiation
6. Requirements Analysis
   - Exercise 3: Use Case Analysis
   - Exercise 4: Goal-Tree Analysis
   - Exercise 5: Basic Prioritization & Selection
7. Requirements Specification
8. Quality Attributes
   - Exercise 6: Mini-QAW
9. Requirements Validation
10. Requirements Change Management

1. Requirements Engineering - Overview


I'LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING !
SHELTON

HERE YOU ARE


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Requirements Engineering

- **Requirements Engineering (RE)** is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used.

  Hence, RE acts as the bridge between the real world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies.

RE (Requirements Engineering)

- **Requirements engineering** is the **process** of establishing
  - **System services** that the customer requires from a system and
  - **Constraints** under which it operates and is developed.

- **Requirements** are
  - **Descriptions** of the **system services** and **constraints**, generated from the RE processes.
    - User-level facility descriptions
    - Detailed specifications of expected system behavior
    - A general system properties
    - Specific constraints on the system
    - Information on how to carry out some computation
    - Constraints on the development of the system

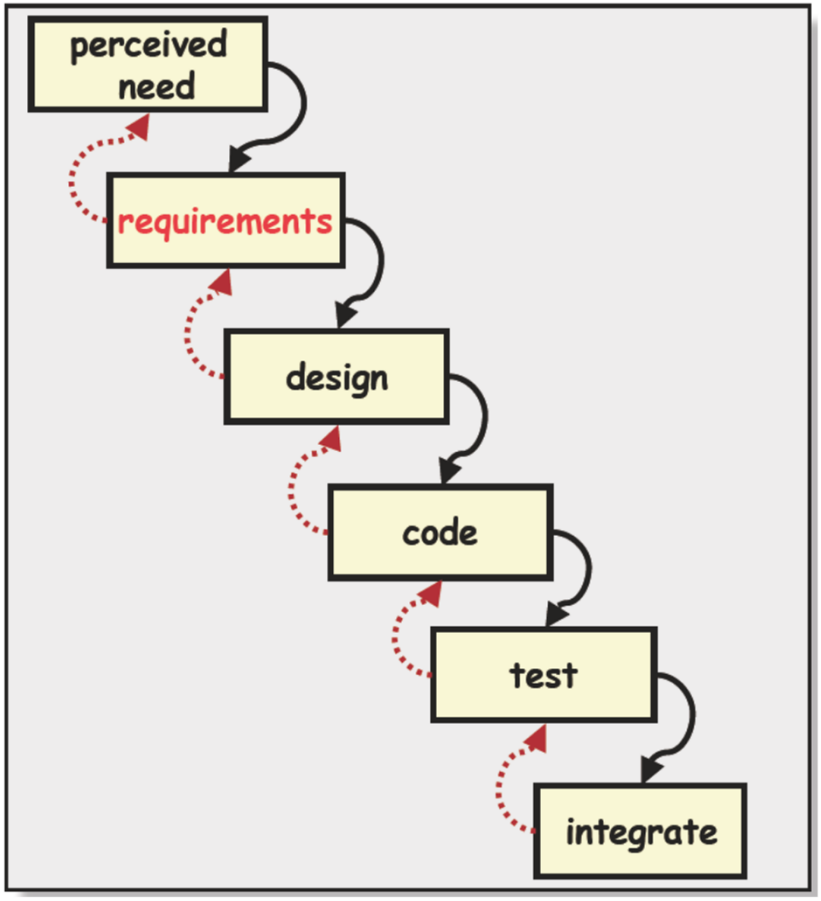
  - **System services** → **Functional requirements (FR)**
  - **Constraints** → **Non-functional requirements (NFR)**

SDLC and RE Process

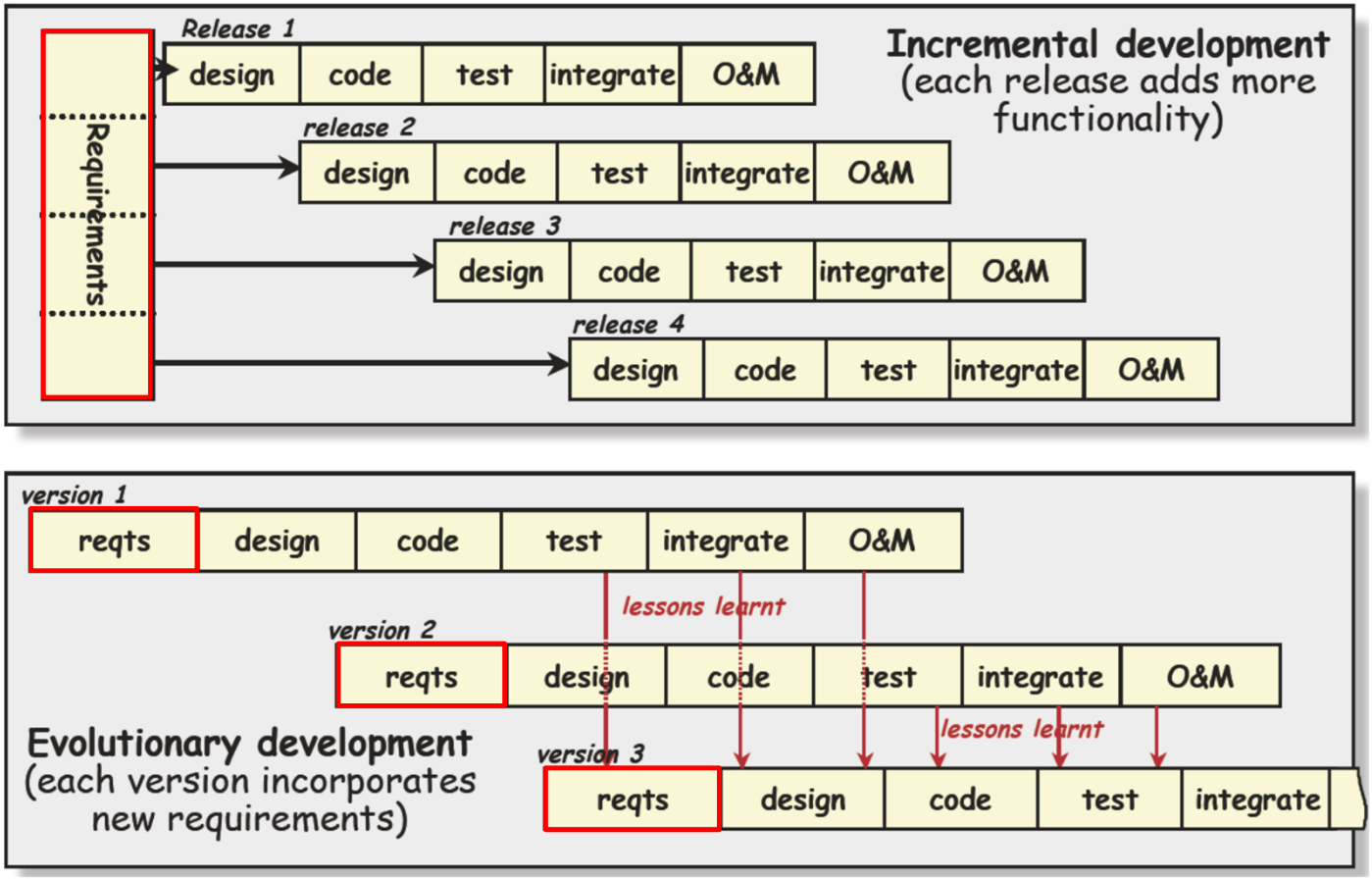
- Requirements engineering process should be adapted to a specific SDLC.
  - **RE process + Development process (SDLC)**

- **Software development life-cycle (SDLC)** models
  - Waterfall
  - Iterative
    - Incremental , Evolutionary
    - Agile , XP
    - UP

Waterfall Model

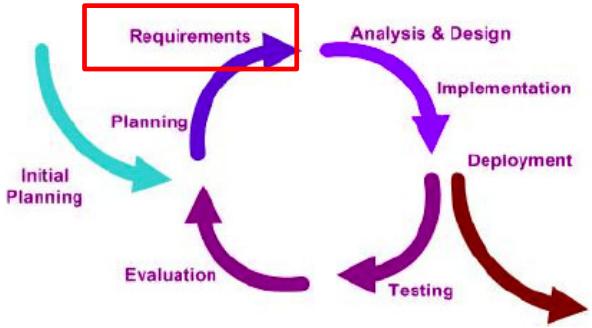


Iterative Models

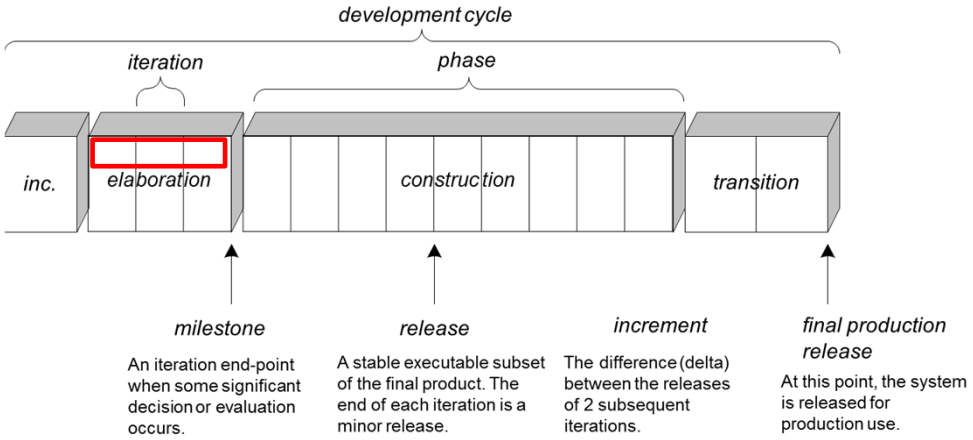


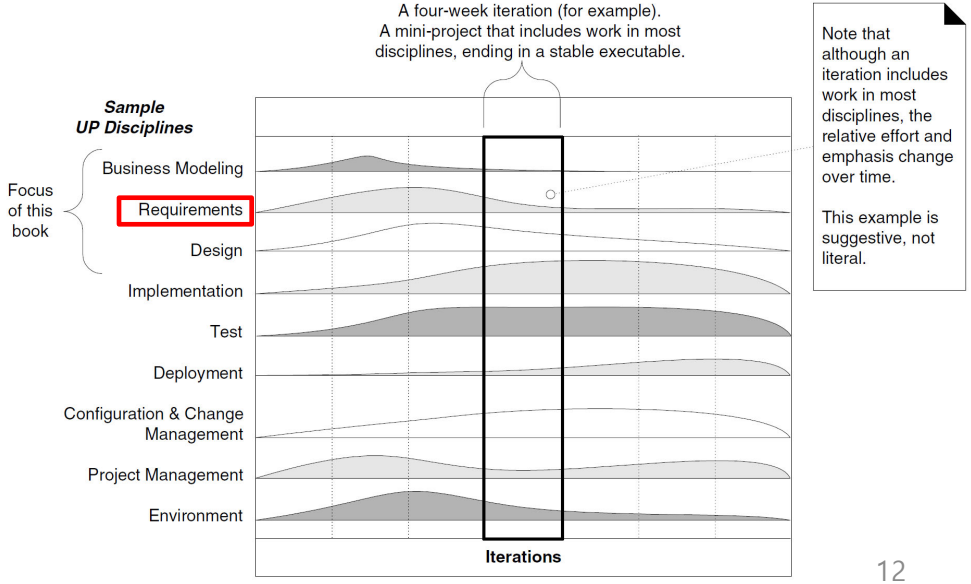
Agile Models and UP

- **Basic Philosophy of Agile**
  - Individual over processes and tools
  - Working software over documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan



- Evolved into **UP** **(Rational Unified Process)**

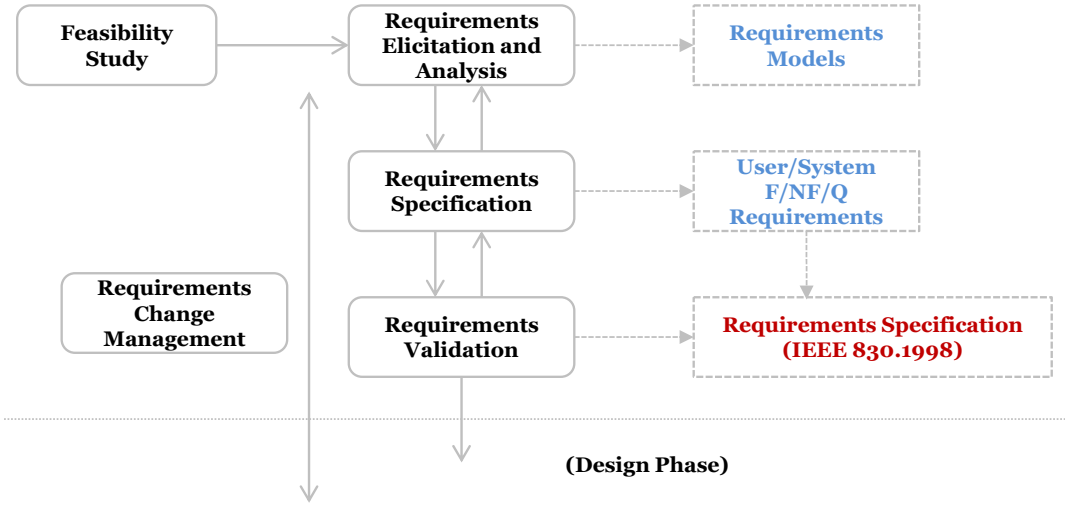




Requirements Engineering Processes

- **Requirement engineering processes** vary depending on
  - Application(target) domain
  - People involved
  - Organization developing the requirements
  - Software development processes used

- **Generic activities** common to all RE processes :



1. Feasibility Study

- Decides whether or not **the proposed system** is worth to develop

- **A short-focused study** to check
  - *"If the system contributes to organizational objectives"*
  - *"If the system can be engineered using current technology and within budget"*
  - *"If the system can be integrated with other systems that are used"*

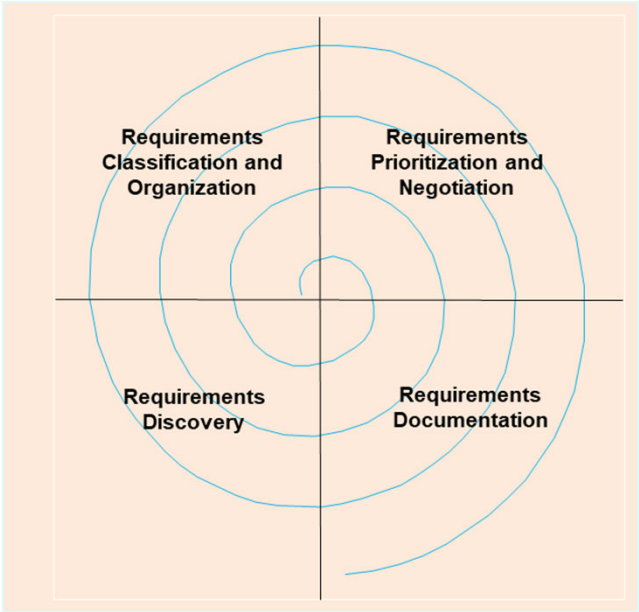
- **Questions**
  - What if the system was not implemented?
  - What are the problems in the current process?
  - How will the proposed system help to satisfy customer's requirements?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

2. Requirements Elicitation and Analysis

- Called also **Requirements Discovery** to find out
  - Application domain, services that the system should provide : **FR**
  - System's operational constraints : **NFR (QA)**

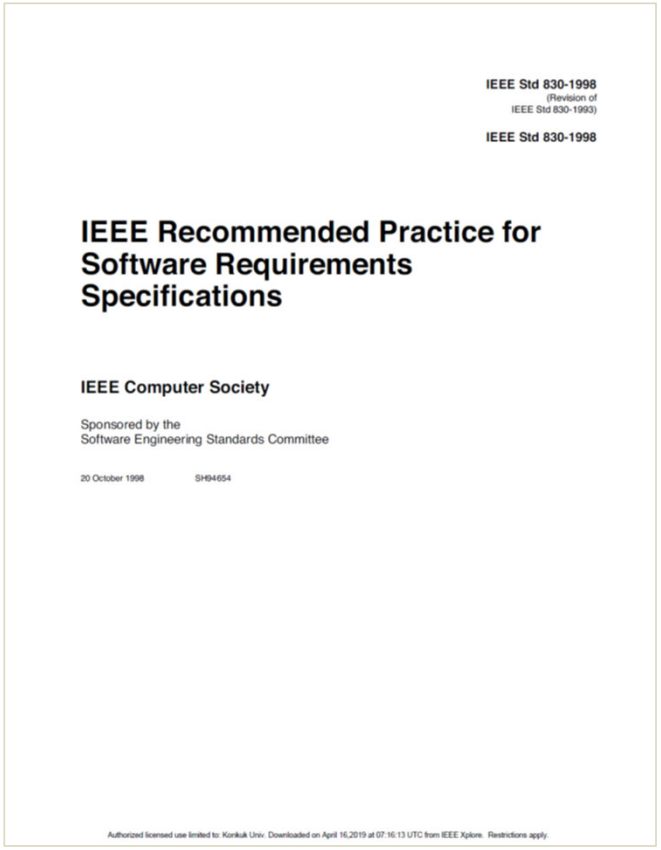
- Should involve various stakeholders (**Stakeholder Analysis**)
  - end-users, managers, engineers, domain experts, trade unions, etc.

- 4 activities performed iteratively
  - **Requirements Discovery**
  - **Requirements Classification and Organization**
  - **Requirements Negotiation and Prioritization**
  - **Requirements Documentation**



3. Requirements Specification

- Write elicited, analyzed, negotiated, prioritized and selected requirements into **documents** according to the **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

Table of Contents

Figure 1—Prototype SRS outline

4. Requirements Validation

- Demonstrate whether the requirements we defined are **what the customer really wants**

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

- **Requirements validation tools:**
  - Requirements reviews
  - Prototyping
  - Test case generation

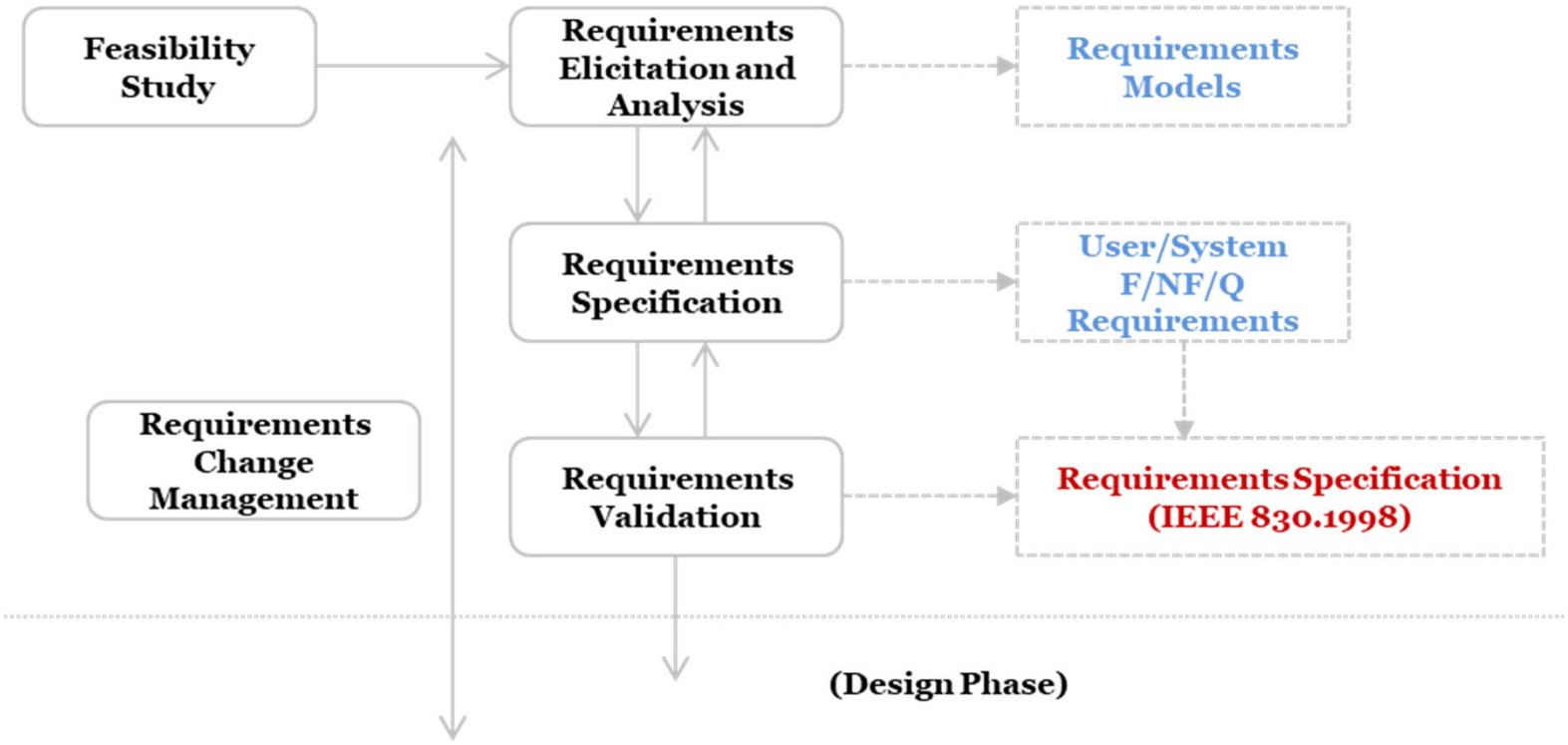
5. Requirements Change Management

- The process of **managing requirements change** during the RE process and the overall system development
  - Requirements are inevitably incomplete and inconsistent.
  - New requirements emerge during the process, as business needs change and a better understanding of the system is developed.

- **Traceability** is the heart of requirements management and all functional safety standards.
  - Source ↔ Requirements ↔ Design ↔ Code ↔ Test

IEC 61508
ISO 26262
EN 50128
DO 178B, 178C
IEC 60880, 62138

Requirements Engineering Process

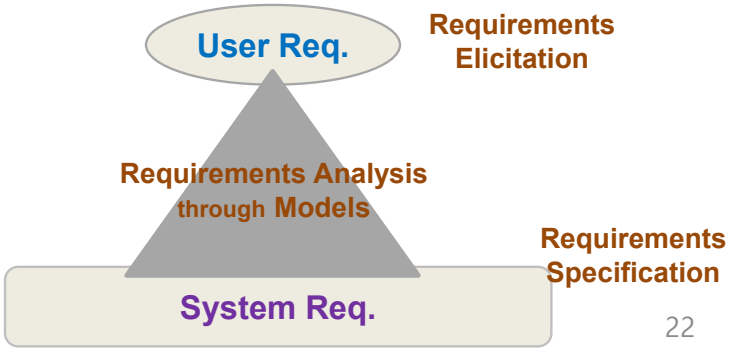


2. Requirements

Requirements

- **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
    - Specified



User and System Requirements

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
    - Constraints on the **development process** or **in operation**
    - Complying to standards (MISRA C, ISO 25010, ISO 26262, etc.)
  - **Quality Attributes**
    - Timing constraints, Performance, Safety, Security, Reliability, etc.
    - **ISO/IEC 25010 (9126)**

Non-Functional Requirements (Quality)

- **The challenge of NFRs**
  - Hard to model
  - Usually stated informally
  - Hard to make them measurable requirements

- Often called **Quality Attributes** or **Quality Requirements**

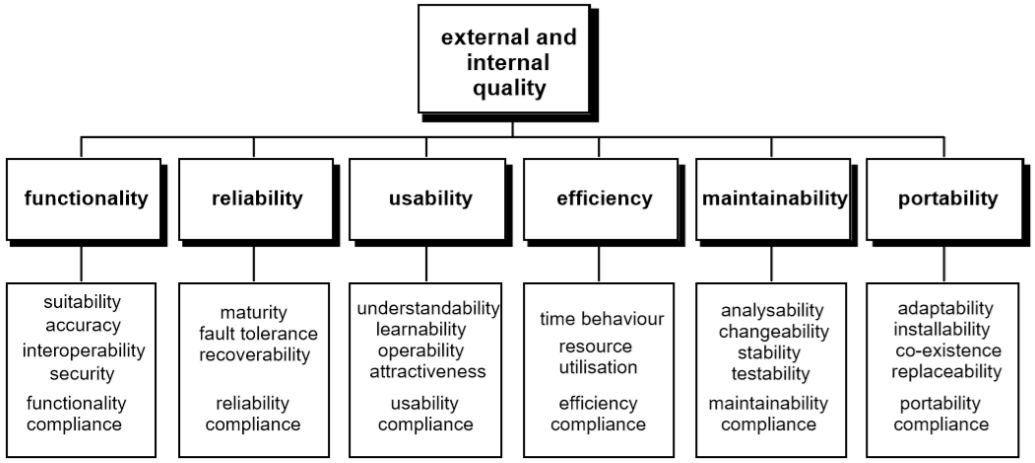
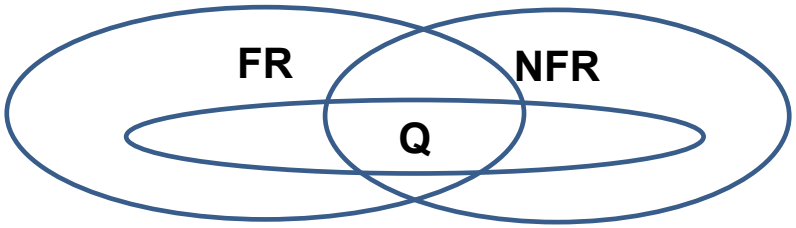


Figure 4 – Quality model for external and internal quality

- Non-functional requirements may be more critical than functional requirements.
  - If these are not met, the system is totally useless.
  - Safety Critical systems often include non-functional requirements into mandatory (i.e., functional) requirements.
    - IEC-61508, ISO 26262 (Functional Safety)

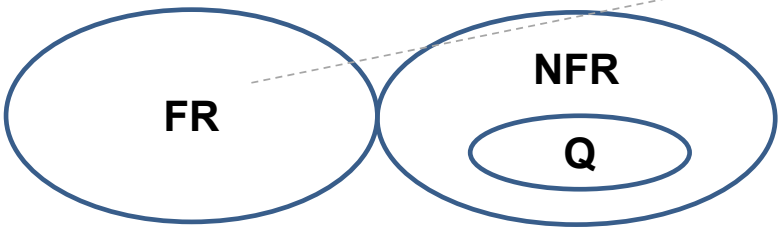
Classifying Functional and Non-Functional Requirements

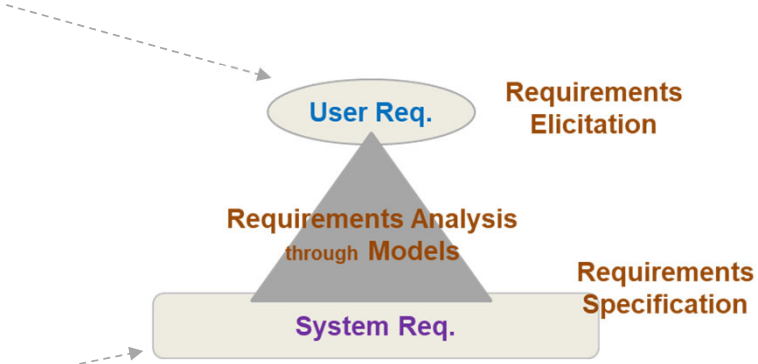
- **At the early phase :**



Several cycles of RE

- **At the end of RE :**





Goals and Verifiable Non-Functional Requirements

- **Non-functional requirements** may be very difficult to state precisely and to verify.
  - Write a "**Goal**" first → transform into "**Verifiable non-functional requirements**"

- **Goal**
  - A general intention of the user
  - Example of QA : *"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.***"

    → by **QAS (Quality Attribute Scenario)**

Requirements Completeness and Consistency

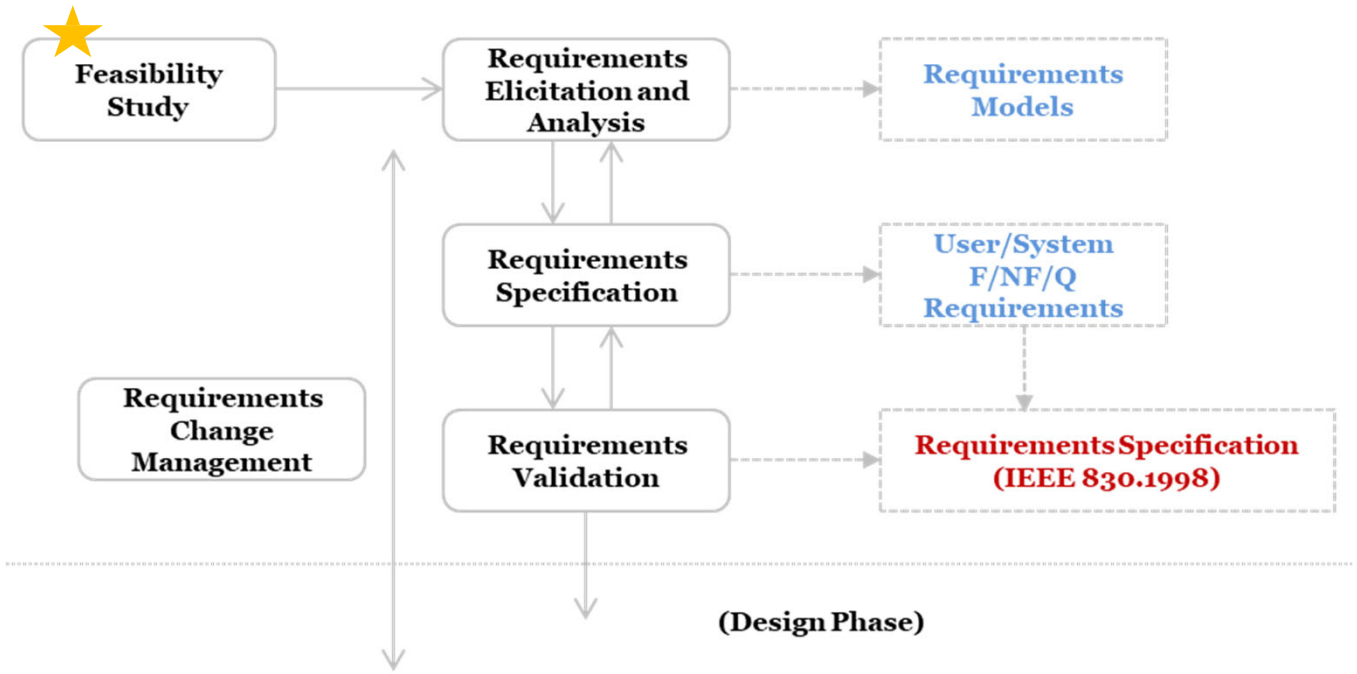
- Problems arise when requirements are not precisely stated.
  - Ambiguous requirements may be interpreted in different ways.

- In principle, requirements should be both **complete and consistent (C&C).**
  - **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.
  - Needs for (formal/informal/semi-formal) requirements models to aid

3. Feasibility Study

Requirements Engineering Process



Why a Feasibility Study?

- **Objectives:**
  - **To find out if** a system development project can be done:
    - "Is it possible?"
    - "Is it justified?"
  - To suggest possible alternative solutions.
  - To provide **management** with enough information to know:
    - Whether the project can be done
    - Whether the final product will benefit its intended users
    - What the alternatives are
    - Whether there is a preferred alternative

- **A management-oriented activity:**
  - After a feasibility study, management makes a "***go/stop***" decision.
  - Need to examine the problem in the context of broader business strategy

Content of Feasibility Study

- **Things to be studied in the feasibility study:**
  - The present (**existing**) organizational system
    - Stakeholders, users, policies, functions, objectives
  - **Problems** with the present system
    - inconsistencies, inadequacies in functionality, performance
  - **Goals** and **other requirements** for the new system
    - Which problems need to be solved?
    - What would the stakeholders like to achieve?
  - **Constraints**
    - Including nonfunctional requirements on the system
  - Possible **alternatives**
    - "Sticking with the current system" is always an alternative
    - Different business processes for solving the problems
    - Different levels/types of computerization for the solutions
  - **Advantages** and **disadvantages** of the alternatives

- **Things to conclude:**
  - Feasibility of the project (**Go** / **Stop**)
  - A preferred alternative

4 Types of Feasibility Study

Technical Feasibility	• ***"Is the project possible with current technology?"***
	• What technical risk is there?
	• Availability of the technology
	  • Is it available locally?
	  • Can it be obtained?
	  • Will it be compatible with other systems?

Economical Feasibility	• ***"Is the project possible, given resource constraints?"***
	• What are the benefits?
	  • Both tangible and intangible
	  • Quantification requires
	• What are the development and operational costs?
	• Are the benefits worth the costs?

Schedule Feasibility	• ***"Is it possible to build a solution in time to be useful?"*** • What are the consequences of delay? • Any constraints on the schedule? • Can these constraints be met?

Operational Feasibility	• ***"If the system is developed, will it be used?"***
	• Human and social issues:
	  • Potential labor objections?
	  • Manager resistance?
	  • Organizational conflicts and policies?
	  • Social acceptability?
	  • Legal aspects and government regulations?

Comparing Alternatives

- **Feasibility Analysis Matrix**
  - Each cells contains the feasibility assessment notes for each candidate.
    - Can be assigned a rank or score for each criterion
  - A final ranking or score is recorded in the last row.

	Candidate 1 Name	Candidate 2 Name	Candidate 3 Name
Description			
Operational Feasibility			
Technical Feasibility			
Schedule Feasibility			
Economic Feasibility			
Ranking			

Feasibility Analysis Matrix

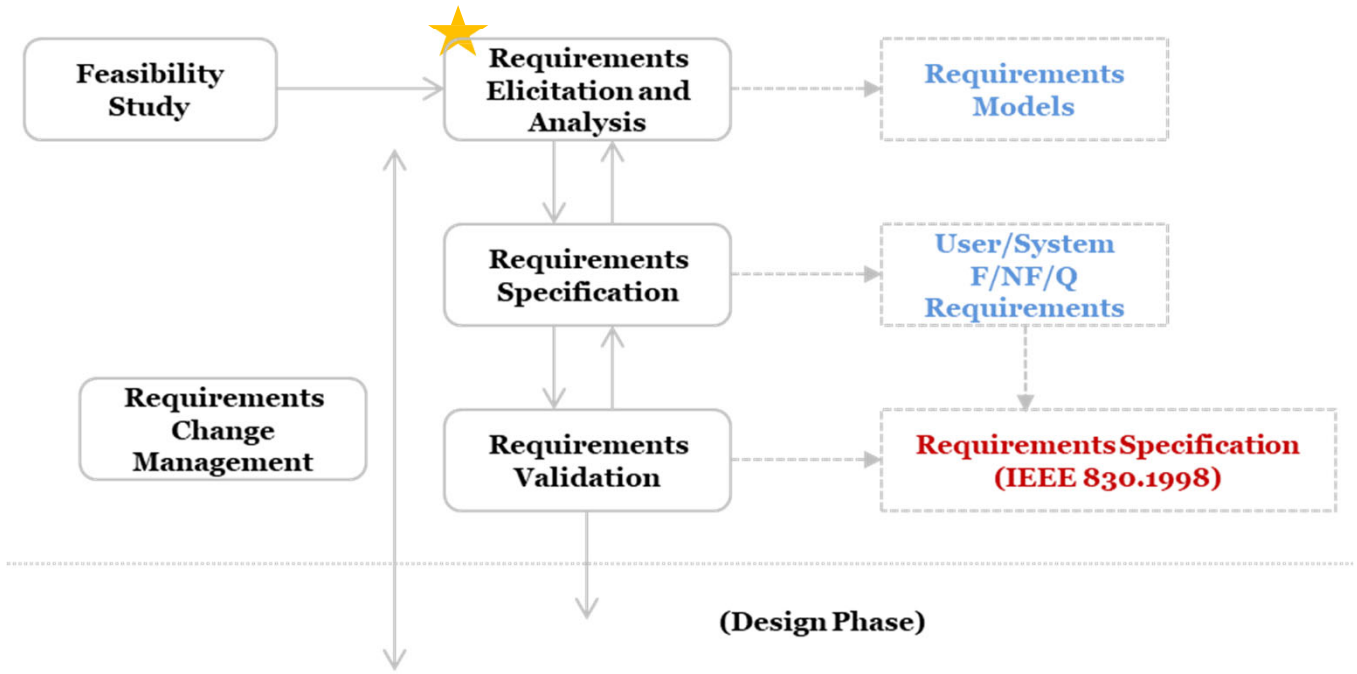
Feasibility Criteria	Wt.	Candidate 1	Candidate 2	Candidate 3	Candidate …
Operational Feasibility **Functionality**. Describes to what degree the alternative would benefit the organization and how well the system would work. **Political**. A description of how well received this solution would be from both user management, user, and organization perspective.	30%	Only supports Member Services requirements and current business processes would have to be modified to take advantage of software functionality **Score: 60**	Fully supports user required functionality. **Score: 100**	Same as candidate 2. **Score: 100**	
Technical Feasibility **Technology**. An assessment of the maturity, availability (or ability to acquire), and desirability of the computer technology needed to support this candidate. **Expertise**. An assessment to the technical expertise needed to develop, operate, and maintain the candidate system.	30%	Current production release of Platinum Plus package is version 1.0 and has only been on the market for 6 weeks. Maturity of product is a risk and company charges an additional monthly fee for technical support. Required to hire or train C++ expertise to perform modifications for integration requirements. **Score: 50**	Although current technical staff has only Powerbuilder experience, the senior analysts who saw the MS Visual Basic demonstration and presentation, has agreed the transition will be simple and finding experienced VB programmers will be easier than finding Powerbuilder programmers and at a much cheaper cost. MS Visual Basic 5.0 is a mature technology based on version number. **Score: 95**	Although current technical staff is comfortable with Powerbuilder, management is concerned with recent acquisition of Powerbuilder by Sybase Inc. MS SQL Server is a current company standard and competes with SYBASE in the Client/Server DBMS market. Because of this we have no guarantee future versions of Powerbuilder will "play well" with our current version SQL Server. **Score: 60**	

Feasibility Analysis Matrix

Feasibility Criteria	Wt.	Candidate 1	Candidate 2	Candidate 3	Candidate …
Operational Feasibility	**30%**	**Score: 60**	**Score: 100**	**Score: 100**	
Technical Feasibility	**30%**	**Score: 50**	**Score: 95**	**Score: 100**	
Economic Feasibility **Cost to develop:** **Payback period (discounted):** **Net present value:** **Detailed calculations:**	**30%**	 Approximately $350,000. Approximately 4.5 years. Approximately $210,000. See Attachment A. **Score: 60**	 Approximately $418,040. Approximately 3.5 years. Approximately $306,748. See Attachment A. **Score: 85**	 Approximately $400,000. Approximately 3.3 years. Approximately $325,500. See Attachment A. **Score: 90**	
Schedule Feasibility An assessment of how long the solution will take to design and implement.	**10%**	Less than 3 months. **Score: 95**	9-12 months **Score: 80**	9 months **Score: 85**	
Ranking	**100%**	**60.5**	**92**	**83.5**	

4. Requirements Elicitation

Requirements Engineering Process

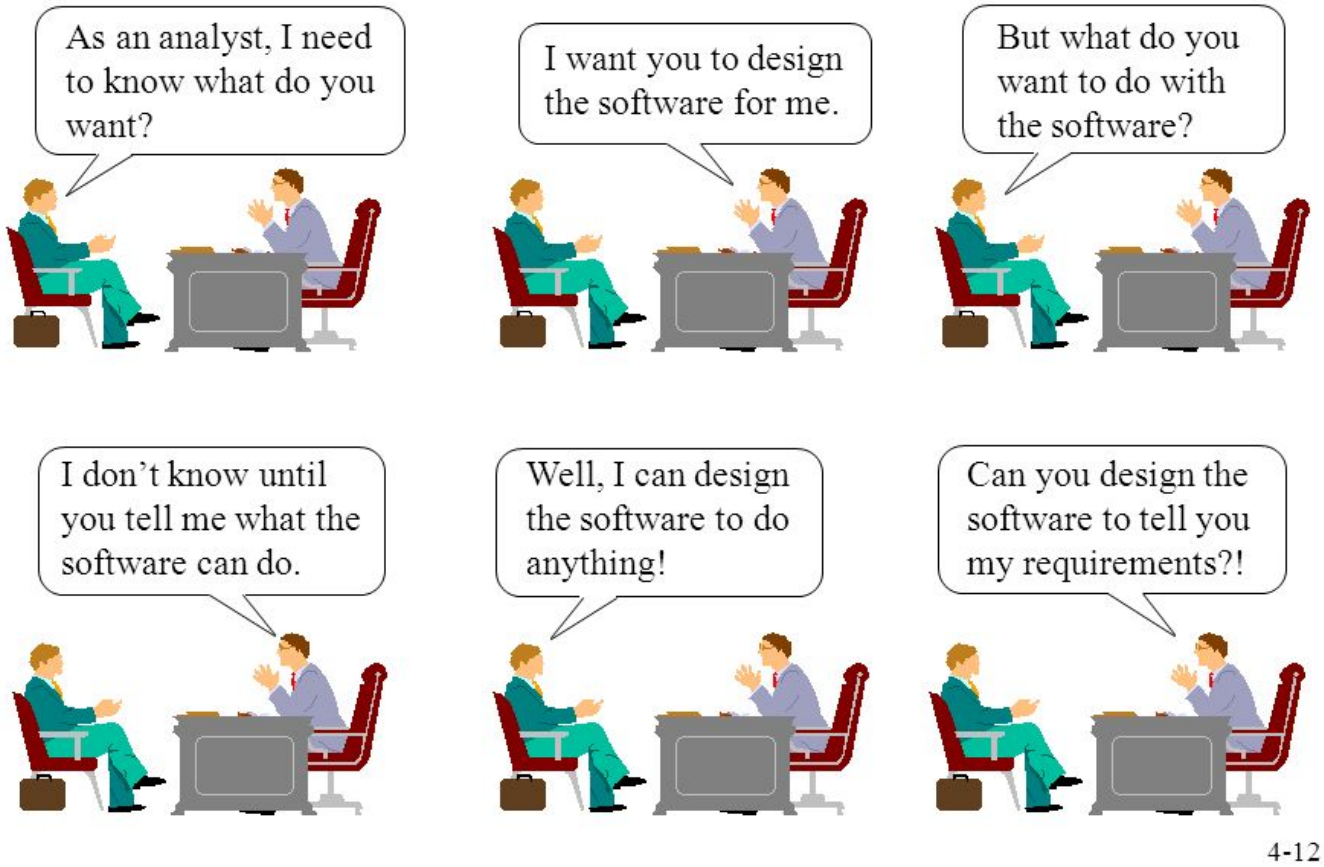


Requirements Elicitation

- There should be a "***problem***" that needs solving.
  - Dissatisfaction with the current state of affairs
  - New business opportunity
  - Potential saving of cost, time, resource usage, etc.

- **Collect enough information to** Identify the "***problem***" and "***opportunity***"
  - Which problem needs to be solved? (identify problem Boundaries)
  - Where is the problem? (understand the Context/Problem Domain)
  - Whose problem is it? (identify Stakeholders)
  - Why does it need solving? (identify the stakeholders' Goals)
  - How might a software system help? (collect some Scenarios)
  - When does it need solving? (identify Development Constraints)
  - What might prevent us solving it? (identify Feasibility and Risk)

Challenges in Requirements Elicitation



Problems of Requirements Elicitation

- **Vague problem stated by the customer (stakeholders)**
  - **Stakeholders** don't know what they really want.
  - Stakeholders express requirements in their own terms.
  - Different stakeholders may have conflicting requirements.
  - New stakeholders may emerge and the business environment changes.
- **Organizational** and **political factors** influence the system requirements.
- The requirements **keep changing** during the analysis process itself.

Stakeholders

- **Stakeholder analysis**
  - **Identify all the people** who must be consulted during information acquisition
  - No specific form of analysis

- **Typical stakeholders :**

User	Concerned with the features and functionality of the new system
Designer	Want to build a perfect system, or reuse existing code
System Analyst	Want to “get the requirements right”
Training and User Support	Want to make sure the new system is usable and manageable
Business Analyst	Want to make sure “we are doing better than the competition”
Technical Author	Will prepare user manuals and other documentation for the new system
Project Manager	Wants to complete the project on time, within budget, with all objectives met.
Customer	Wants to get best value for money invested

The Requirements Elicitation Activities

1. Requirements Discovery

- Interacting with stakeholders to discover their requirements
- Domain requirements are also discovered at this stage.

2. Requirements Classification and Organization

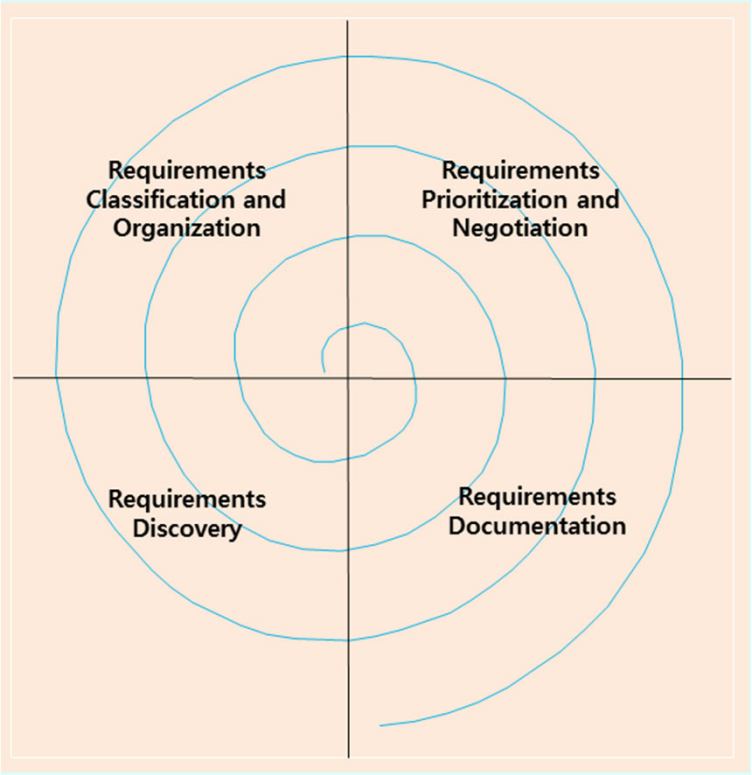
- Groups related requirements and organizes them into coherent clusters

3. Negotiation and Prioritization

- Resolving requirements conflicts (for user requirements)
- Prioritizing requirements (for system requirements, actually)

4. Requirements Documentation

- Document requirements in a form of annotated requirements list
- Input it into the next round of the spiral

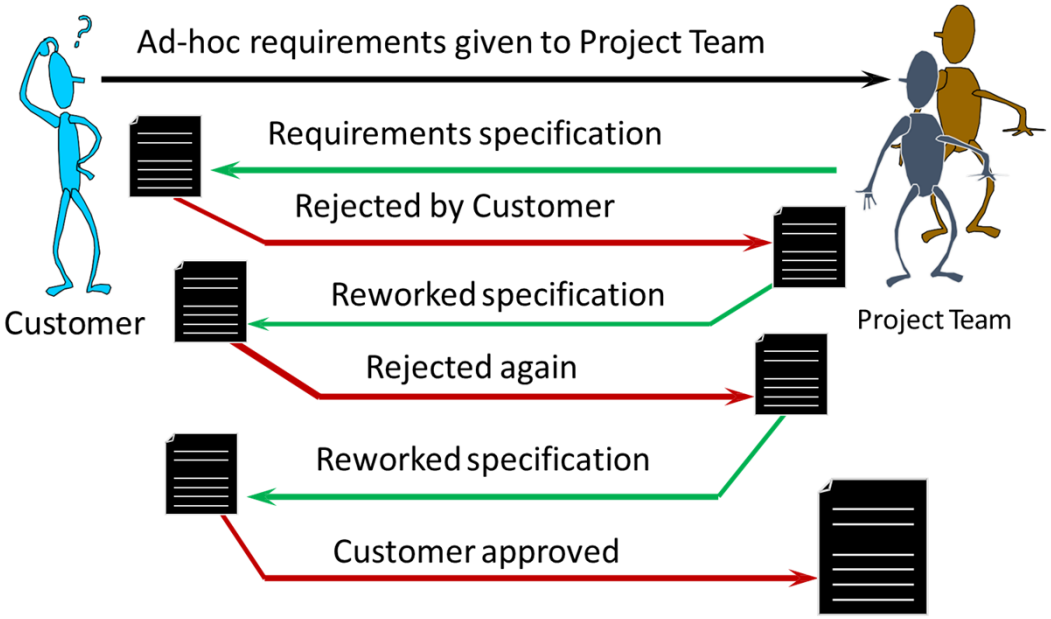


Things to Remember When Eliciting Requirements

1. **Don’t Lose Sight of the Goal**
2. **Think Who’s Smart**
3. **A Single Stakeholder Can’t Speak for All**
4. **Use Appropriate Elicitation Methods**
5. **Accept Requirements Changes**
6. **Manage Elicited Requirements**

1. Don't Lose Sight of the Goal

- Establish the system's vision and scope to reduce the risk of building the wrong system
- Try to obtain early commitment from stakeholders



2. Think Who's Smart

- Don't try to convince stakeholders that YOU are smart.
- Instead take everybody to show you think the STAKEHOLDER is smart

- Contrast these 2 cases:

1. My Elevators Are Too Slow!

2-1. I See. Tell Me Why You Feel They Are Too Slow.

2-2. I Don't Think So. I Think You Have an Elevator Throughput Problem, not a Speed Problem.

3. A Single Stakeholder Can't Speak for All

Stakeholder	Role
User	• Users of the system and the results of the system • ALWAYS included • Often many classes - make sure all are represented
Customer	• People with decision making authority • ALWAYS included; no project otherwise! • Often many classes - make sure all are represented • Closely aligned with marketing function
Marketing	• ESSENTIAL; The experts in the "market" • Too easy for development to dismiss them • In a commercial setting, they know the pulse of customers
Subject Matter Experts (SME)	• Helpful to learn foundation requirements • Helpful to alleviate disagreements among stakeholders
Developer	• Helpful to learn system implications • Helpful to learn evolution / maintenance requirements
Development Managers	• Knows the development capability and resources
Tester	• Useful a bit later in project • Knows which requirements are testable
Loser Users	• People who loses power as a result of the project • Useful if a system has "loser users"
Technical Writers Trainers / Customer Support	• Can also help • Experts in making the system easy to use/teach/explain

4. Use Appropriate Elicitation Methods

- **Methods** for requirements elicitation:
  - **Interviews**
  - **Role Playing**
  - **Brainstorming**
  - **Requirements Workshop**
  - **Prototyping**
  - **Storyboard**
  - **Survey/Questionnaire**
  - **Use Case**

- **<u>A single method may not be sufficient</u>**.
  - Consider requirements' size, complexity, etc., and select several ones.

5. Accept Requirements Changes

- **Requirements change is inevitable.**
  - Clients have right to change requirements.
  - The more features the product has, the more customers want.

- Don’t ever ask “***Okay, is that your final requirement?***”

- Change is not a threat, it’s an opportunity.

6. Manage Elicited Requirements

- Record the **rationale** of each requirement
  - Reason why requirement is necessary
  - Assumptions on the requirement

- Managing the rationale with **annotated requirements lists**
  - Do not simply rewrite the requirement
  - Make it unique for each requirement
  - Keep it simple

- Example:
  - Requirements : "***The truck shall have a height of no more than 14 feet.***"
  - Rationale : "***99% of all U.S. Interstate highway overpasses have a 14-foot or greater clearance.***"

Techniques for Eliciting Stakeholder Needs

- **Requirements Elicitation Methods**
  1. Requirements workshop
  2. Brainstorming
  3. Storyboards
  4. Interviews
  5. Survey/Questionnaires
  6. Role playing
  7. Prototypes
  8. Use-Case

1. Requirements Workshop

- **Gather all stakeholders together** for an intensive and focused period
  - Create consensus on the scope, risks and key features of the software system
  - Results immediately available
  - Outputs:
    - Problem statement , Key features , Initial business object model, Use-case diagram , Prioritized risk list, etc.

- Provide a **framework** for applying other elicitation techniques such as
  - Brainstorming, use-case workshops, storyboarding, etc.

2. Brainstorming

- **Rules for Brainstorming**
  - Clearly state the objective of the session
  - Generate as many ideas as possible
  - Let your imagination soar
  - Do not allow criticism or debate
  - Even the impractical, absurd ideas should not be neglected
  - Merge the various ideas to create new ideas

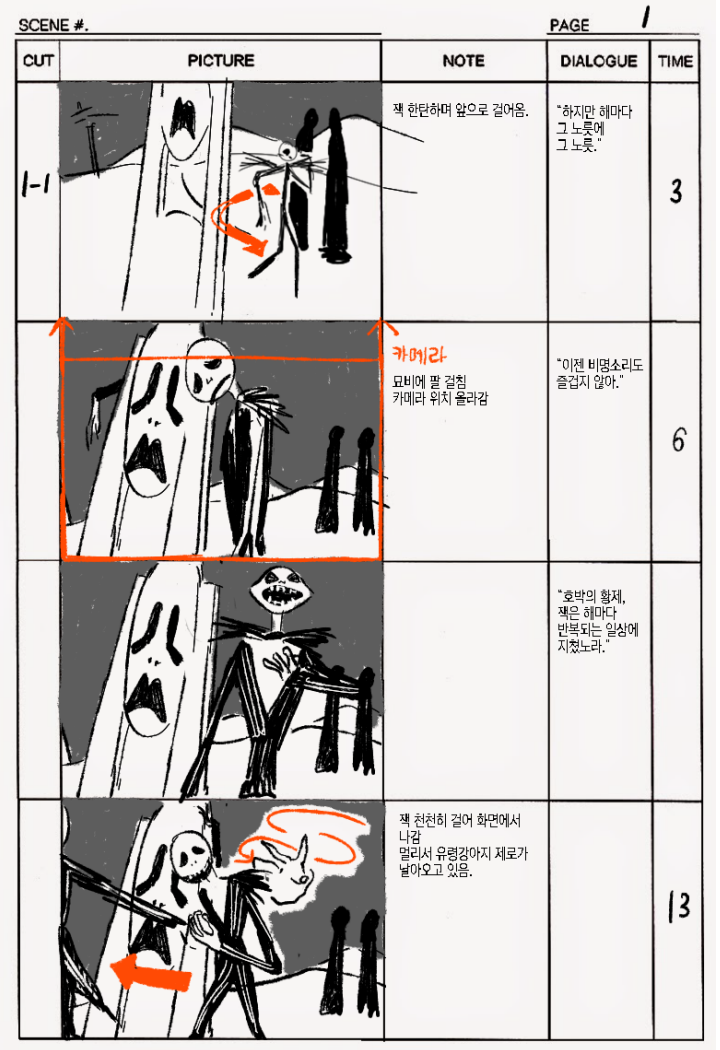
- **Express freely**
  - Do not explain or specify the ideas
  - Do not evaluate or argue about the ideas
  - Do not put names on the ideas
  - Encourage the unexpected and imaginative

- **Put up ideas openly**
  - Ideas should be put up on a whiteboard where all can see
  - Participants themselves may put up ideas on the board
  - Put tabs of Post-Its on the center table

3. Storyboards

- Visually tell and show:
  - Who/what the players are (actors)
  - What happens to them
  - When it happens

- Benefits
  - Help gather and refine customer requirements
  - Encourage creative and innovative solutions
  - Encourage team review
  - Prevent features that no one wants
  - Ensure that features are implemented in an accessible and intuitive way
  - Ease the interviewing process
  - Help to avoid blank-page syndrome



4. Interviews

- Provide **a simple and direct technique** to gain understanding of problems and solutions

- Types of interviews
  - **Open interview**
    - No pre-set agenda
    - Irrelevant data can be gathered
    - Needs time and training
  - **Closed interview**
    - Fairly open-questions agenda
    - Needs extended preparation
    - Prevents biases

- Interview tips
  - Avoid asking people to describe things they don't usually describe
    - Example: Describe how to tie your shoes
  - Avoid "Why…?" questions
  - Ask open-ended (context-free) questions
    - High-level abstract questions

5. Survey/Questionnaires

- Give access to a wide audience
  - Apply to broad markets where questions are well-defined
- **Statistical analysis** is applicable.
  - Powerful, but not a substitute for an interview
- Assumptions:
  - Relevant questions can be decided in advance
  - Questions phrased, so reader hears as intended

고 객 명		소 속		조사일자	20 년 월 일

■ 설문조사내용

구 분	평 가 내 용	평 가	점 수
품질	1. 고객께서 알고 있는 회사명의 이미지는 어떠하십니까?	A : 좋 다 B : 보 통 C : 나쁘다	
	2. 고객께서는 회사명으로 전화를 하셨을 때 친절 하게 응대를 하였습니까?	A : 좋 다 B : 보 통 C : 나쁘다	
	3. 고객께서는 회사명의 제품에 만족 하십니까?	A : 만 족 B : 보 통 C : 불만족	
	4. 당사의 제품에 이상 발생 시 고객께서 요구했을 때 신속한 대응이 있었습니까?	A : 그렇다 B : 보 통 C : 아니다	
	5. 품질관련해서 동종업계화의 수준에 있어 어느 정도의 수준이라고 생각하십니까?	A : 상 위 B : 중 위 C : 하 위	
	평 가 점 수 A : 20 B : 15 C : 10	소 계	
생산기술	1. 회사명을 과거에 방문하신 적이 있으면 어떤 느낌을 받았습니까?	A : 좋 다 B : 보 통 C : 나쁘다	
	2. 회사명의 생산기술에 있어 동종 업계와의 기술수준은 어느 정도라고 생각하십니까?	A : 상 위 B : 중 위 C : 하 위	
	3. 제품에 대한 상호간 정보교환은 잘 이루어지고 있습니까?	A : 그렇다 B : 보 통 C : 아니다	
	4. 신규 제품 개발 시 경쟁업체와 비교했을 때 개발일정 및 진행이 잘 되고 있습니까?	A : 그렇다 B : 보 통 C : 아니다	
	평 가 점 수 A : 25 B : 20 C : 15	소 계	
영업	1. 당사의 납입 준수율은 어느 정도라고 생각하십니까?	A : 80%이상 B : 50%이상 C : 50%미만	
	2. 당사의 납품수량은 정확하다고 생각하십니까?	A : 정 확 B : 보 통 C : 미 흡	
	3. 회사명의 서비스(친절함) 수준은 어느 정도라고 생각하십니까?	A : 좋 다 B : 보 통 C : 나쁘다	
	4. 포장 및 제품에 청결함과 더불어 사용하는데 불편함이 있었습니까?	A : 그렇다 B : 보 통 C : 아니다	
	평 가 점 수 A : 25 B : 20 C : 15	소 계	
고객의 견		평 점	

6. Role Playing

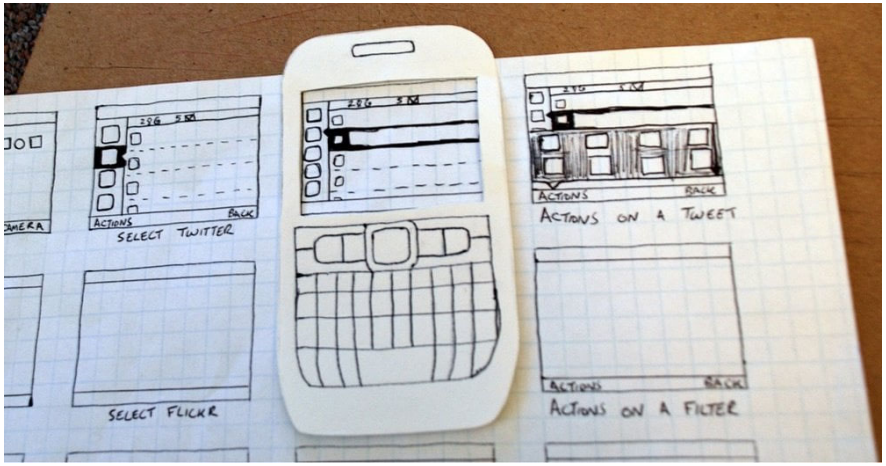
- Perform requirements elicitation from the viewpoint of the roles
  - Learns and performs user's job
  - Performs a scripted walkthrough

- Advantages
  - Gain real insights into the problem domain
  - Understand problems that users may face

7. Prototypes

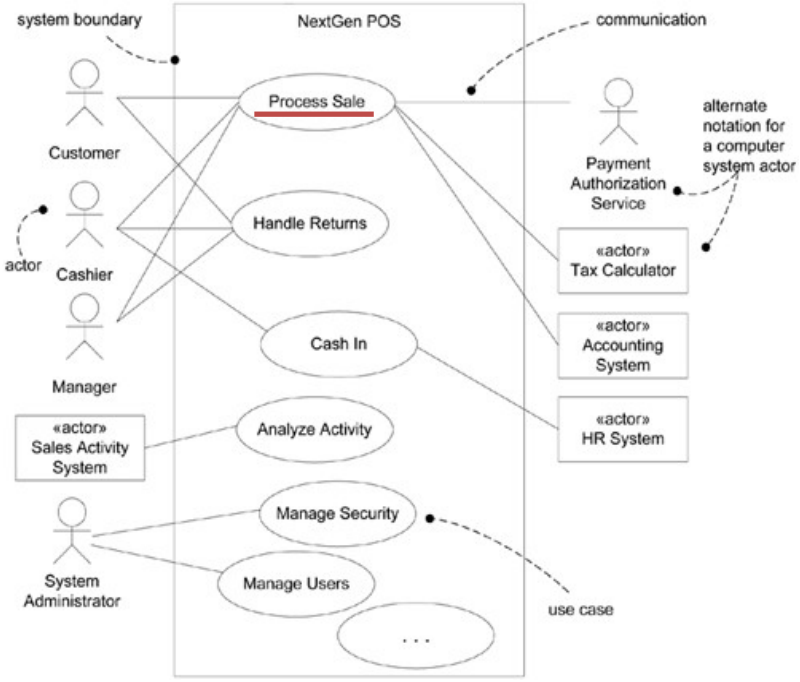
- Demonstrate some or all externally observable behaviors of a system through building prototypes quickly

- Used to:
  - Demonstrate understanding of the problem domain
  - Gain feedback on proposed solution
  - Validate known requirements
  - Discover unknown requirements
  - Create simulations
  - Elicit and understand requirements
  - Prove and understand technology
  - Reduce risk
  - Enhance shared understanding
  - Improve cost and schedule estimates and feature definitions



8. Use-Case

- **Text stories** of some actors using a system to meet goals
  - A mechanism to capture and analyze requirements (from elicitation to analysis)
  - Use case is not a diagram, but a text.
  - Use cases are requirements, primarily **functional (behavioral)** requirements.



Use Case : Process Sale

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

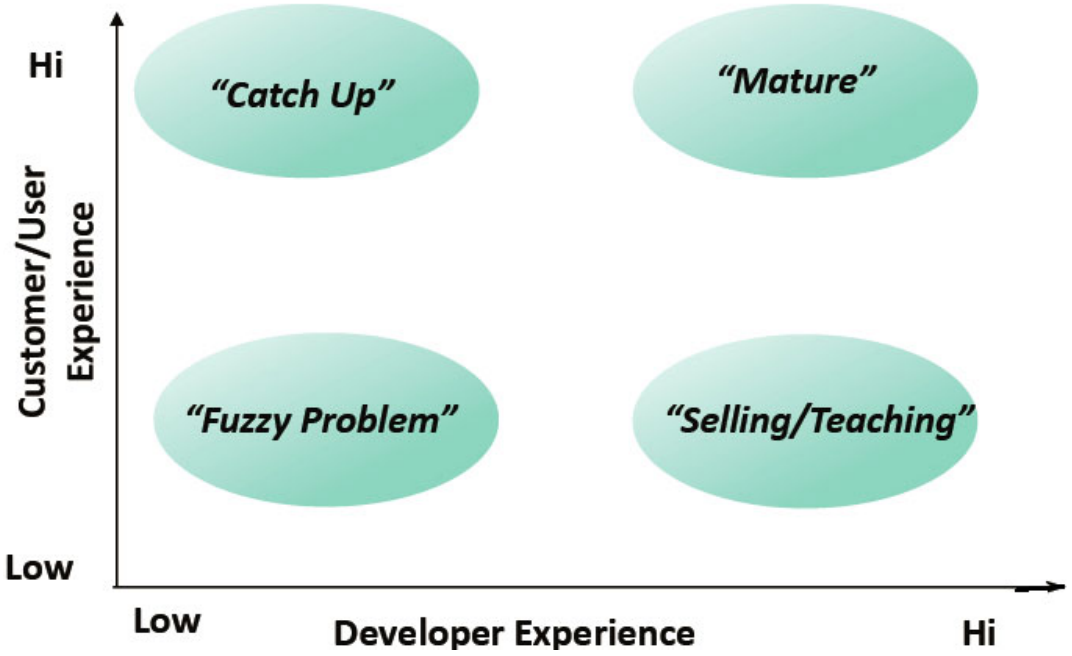
Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Which Techniques to Use?

- No single technique is sufficient for realistic projects.

  - **Catch Up**
    - Role Playing
    - Interview
  - **Fuzzy Problem**
    - Requirements Workshops
    - Brainstorming
    - Storyboards
  - **Selling / Teaching**
    - Use-Case
  - **Mature**
    - Questionnaires
    - Prototyping



What to Do with Elicited Requirements?

- **Maintain requirements in lists**
  - Maintaining **a list of requirements** can support all activities of requirements.

- Enables you to **answer questions** such as:
  - *How many requirements do you have?*
  - *How many high priority requirements do you have?*
  - *What percentage of the requirements deemed high priority by customer X are you satisfying with?*
  - *What percentage of the candidate requirements have you chose to satisfy in your next release? (Actually later)*

Example of Elicitation Results : A list of (annotated) requirements

ID	Requirement Text
961	No formal training shall be required to operate the RLM.
955	Any new releases or versions of the software shall be sold as new products. Users must p...
954	User software will not be modified or upgraded.
512	The RLM shall return to the refuel location or dump area to within 10 cm of the user-define...
432	Pressing the screen in an area without a command shall make no sound nor shall it be int...
415	The screen shall be capable of displaying alphanumeric data in blocked, uppercase char...
500	The RLM shall accept lawn and obstacle programming from the user. During programming, th...
321	The RLM shall initiate communications with the GPS through external interface EL-GPS
300	The RLM shall interface with two different external systems, The GPS and the Electronically S...
310	External interfaces include the receipt of location data from GPS and detection of obstacles...
511	The RLM shall not overcut or undercut the border and user defined obstacles by more tha...
510	The RLM shall cut the lawn only within the area defined by the user during the programming.
550	Border programming shall be required to be completed by the user prior to accepting the oth...
411	The Screen shall be 16.25 mm (high) by 105 mm (wide) and capable of displaying two row...
446	Serious errors (for example, blade fouling, Requirement 179) shall not have a button on th...
553	Programming border data shall be terminated by a user request, or when the RLM returns t...
554	After the termination, the RLM shall be ready to receive another command.
418	The screen shall be used to display information from the RLM to the user and accept dire...
561	User shall guide the RLM to the obstacle and indicated that the boundary of obstacle will ...
562	RLM shall record sufficient data (e.g. from GPS) to meet the accuracy requirements stated...
552	RLM shall record sufficient data (e.g. from GPS) to meet accuracy requirements stated in ...
551	User shall guide the RLM to the border of the lawn and indicate that the boundary will be ...

Exercise 1 : Requirements Workshop - Requirements Elicitation

- Let's do **Requirements Workshop** to develop **a new advanced OOO digital watch** **next season**
  - **(1) Stakeholder Analysis** for finding all relevant stakeholders around 8 ~ 15

Stakeholder (Role)	Goals	희망사항
개발팀장	개발노력 ↓ On Time ↑ 퇴사률 ↓	퇴사율이 떨어지면 좋겠다. 야근을 덜 했으면 좋겠다. 나중에 요구사항 변경이 없어야만 한다.
기획/판매팀장(사장)	영업이익 ↑ 회사평판 ↑	
품질팀장	품질 ↑ 테스트노력 ↓	
동호회장(사용자)	사용성 ↑ 편의성 ↑ 가격 ↓	
…		

  - **(2) Brainstorming** for eliciting new requirements with markers and Post-Its®
    - **Role playing**
    - Eliciting about **20~30** ideas
    - Organize into **20** requirements
    - Categorize with different colors (**FR**, **NFR**, **Q**)

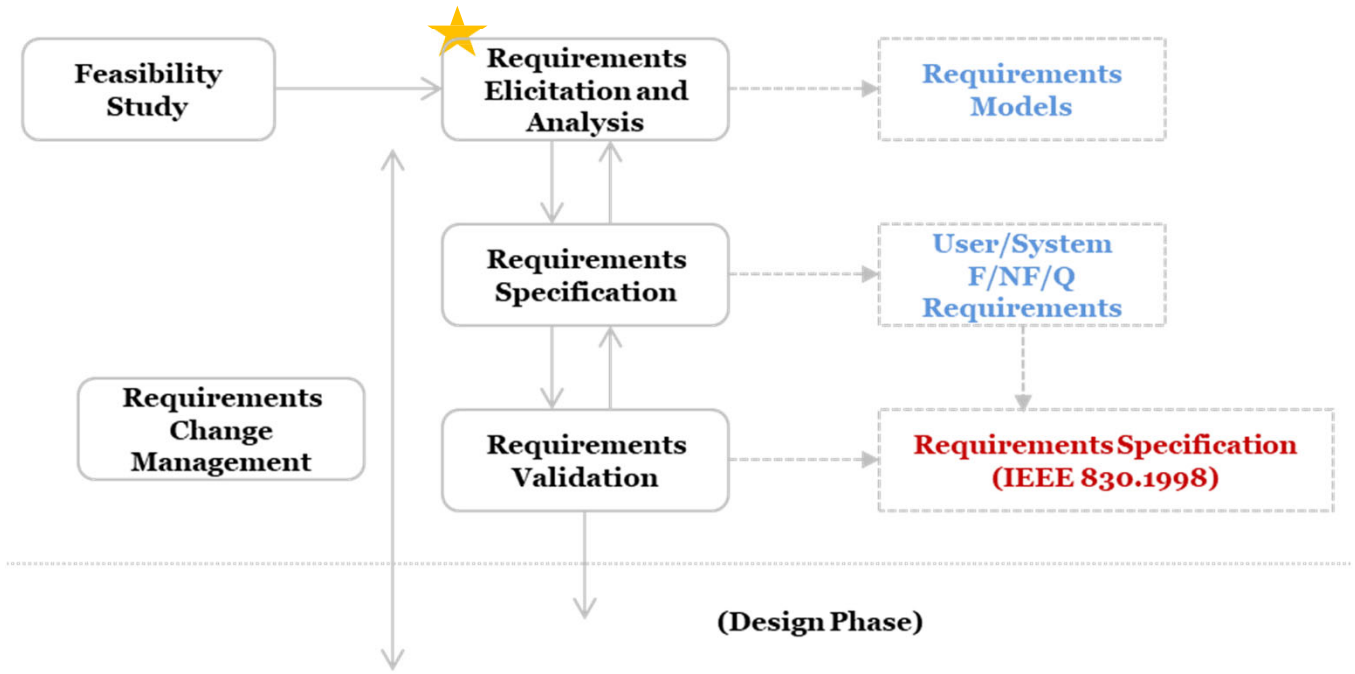
Basic HW features :
- 4 Buttons , 1 Buzzer , 1 LCD , 1 SW downloadable
- GPS , LTE , Wi-Fi , Bluetooth, 5G, Camera, Sensors
- Any addition/extensions are available.

The Brainstorming Result after Consolidation (after discussion)

Stakeholders	Req. ID	Requirements
User (전자시계 동호회 대표)	[1]	시계 사용자 사이에 숫자 야구 게임에 대해 시간/횟수에 대한 스코어링 및 랭킹 시스템 지원
		~~수심 20M 방수 지원~~
	[2]	최소 6개월 이상의 배터리 수명 보장
백화점 판매팀장		~~트렌디한 디자인~~
	[3]	extensible hw 에 대한 시연 지원
	[4]	baseball game 외 시계사용법에 대한 메뉴얼 제공
상품기획팀장		~~Watch face 가 변경 가능해야 함~~
	[5]	상품 가격이 20만원이 넘지 않아야 함
	[6]	다양한 언어(최소 10개국) 를 지원해야 함
HW/UX Designer		~~않아야 함~~
	[7]	메뉴 사용이 편리해야 함
	[8]	스트랩 교체 가능
SW Engineer	[9]	1년에 1초 미만의 오차를 보장하는 H/W clock 및 이를 활용하기 위한 인터페이스 지원 필요
	[10]	7-segment display로 표현 가능한 언어에 대해서만 지원 가능
		~~C++를 활용한 개발이 가능하도록 관련 Tool 및 Cross compiler 지원~~
Product Quality Manager	[11]	5명의 테스터가 제품 개발 단계 진행을 위한 테스트를 일주일 이내에 완료될 수 있어야 함.
	[12]	100,000대당 1대의 불량률을 구현할 수 있어야 함
		~~시장 문제 대응 비용을 최소화할 수 있어야 함~~
CEO	[13]	Smart Watch 시장의 시장 점유율을 20%이상 점유할 수 있어야 함
	[14]	10억 이내의 개발비로 개발이 완료되어야 함
		~~영업 이익률을 30% 이상 가져갈 수 있어야 함~~
HW Engineer	[15]	Additional module 은 3개 이하로 제한
		~~Camera 는 1개만 탑재 가능~~
	[16]	배터리 교체는 불가능
UI Designer	[17]	color 지원
		~~360X360 해상도 이상 지원~~
	[18]	animatable ui 지원
SW Tester		~~hw features 를 테스트하기 위한 환경이 제공되어야함~~
	[19]	시계버튼 반응속도/화면전환 속도가 타사 digital 시계 대비 우수해야함
	[20]	배터리 사용시간이 idle 상태에서 5일이상되어야함

5. Requirements Negotiation

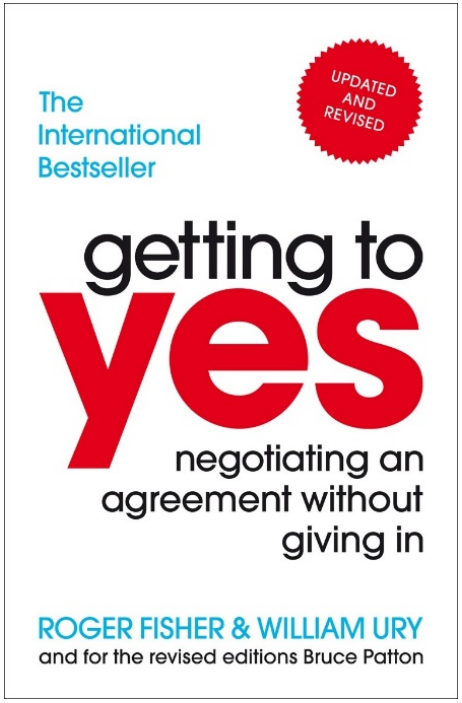
Requirements Engineering Process

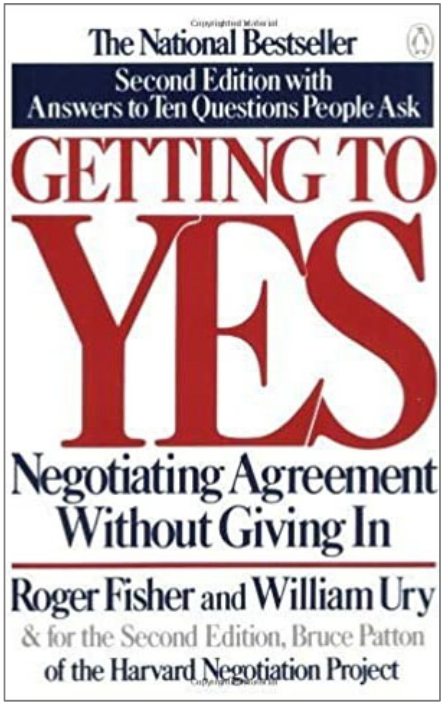


Needs for Requirements Negotiation

- **Requirements are negotiated** to achieve mutually satisfactory agreements.
  - Users, customers, managers, domain experts, and developers share different skills, backgrounds and expectations.
  - Requirements emerge from a process of co-operative learning in which they are explored, prioritized, negotiated, evaluated, and documented.

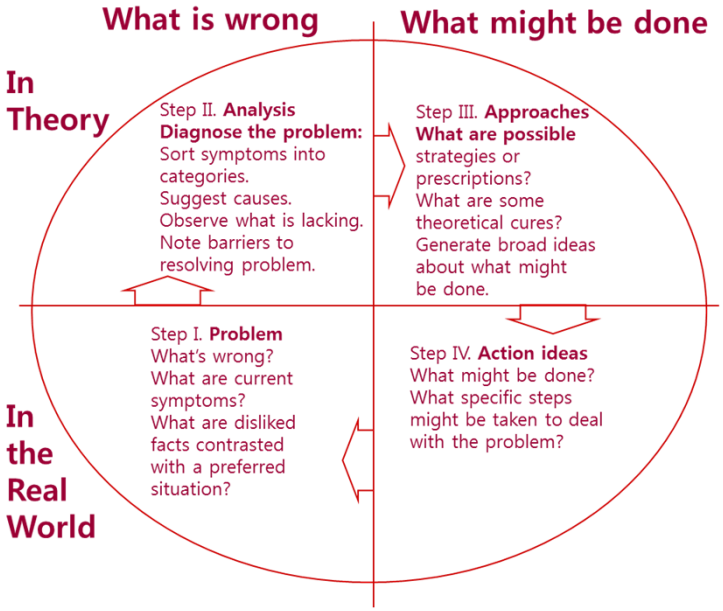
- **[Fisher & Ury, "Getting to Yes," 1981]**
  - "*Negotiating an agreement without giving in*"
  - **4-step solution** approach
    - Separate the people from the problem
    - Focus on interests, not positions
    - Invent options for mutual gain
    - Insist on using objective criteria

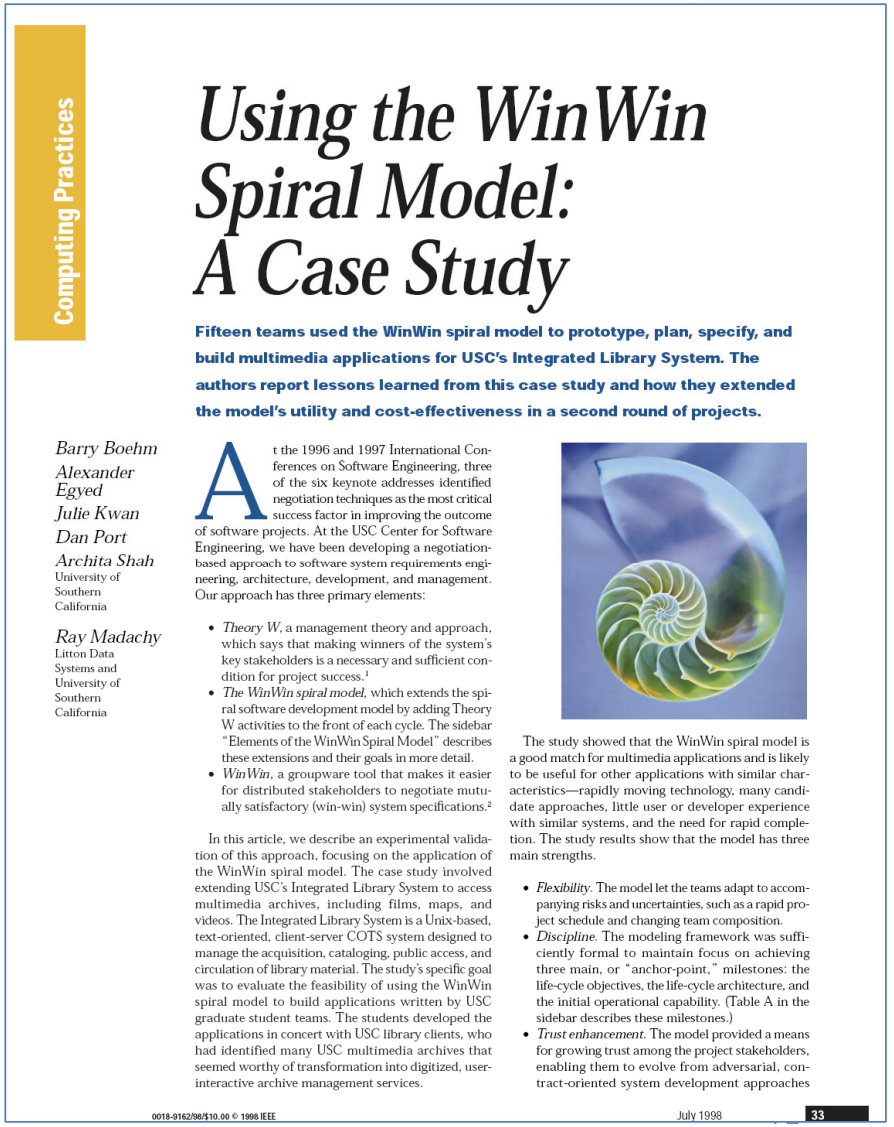




WinWin Negotiation

- **The WinWin approach**
  - A set of principles, practices and tools
  - Enabling a set of interdependent stakeholders to work out a mutually satisfactory (win-win) set of shared commitments
  - **Win-lose generally becomes Lose-lose.**
    - **Nobody** wins in these situations.





Computing Practices

Using the WinWin Spiral Model: A Case Study

Fifteen teams used the WinWin spiral model to prototype, plan, specify, and build multimedia applications for USC's Integrated Library System. The authors report lessons learned from this case study and how they extended the model's utility and cost-effectiveness in a second round of projects.

Barry Boehm
Alexander Egyed
Julie Kwan
Dan Port
Archita Shah
University of Southern California

Ray Madachy
Litton Data Systems and University of Southern California

At the 1996 and 1997 International Conferences on Software Engineering, three of the six keynote addresses identified negotiation techniques as the most critical success factor in improving the outcome of software projects. At the USC Center for Software Engineering, we have been developing a negotiation-based approach to software system requirements engineering, architecture, development, and management. Our approach has three primary elements:

- *Theory W*, a management theory and approach, which says that making winners of the system's key stakeholders is a necessary and sufficient condition for project success.[1]
- *The WinWin spiral model*, which extends the spiral software development model by adding Theory W activities to the front of each cycle. The sidebar "Elements of the WinWin Spiral Model" describes these extensions and their goals in more detail.
- *WinWin*, a groupware tool that makes it easier for distributed stakeholders to negotiate mutually satisfactory (win-win) system specifications.[2]

In this article, we describe an experimental validation of this approach, focusing on the application of the WinWin spiral model. The case study involved extending USC's Integrated Library System to access multimedia archives, including films, maps, and videos. The Integrated Library System is a Unix-based, text-oriented, client-server COTS system designed to manage the acquisition, cataloging, public access, and circulation of library material. The study's specific goal was to evaluate the feasibility of using the WinWin spiral model to build applications written by USC graduate student teams. The students developed the applications in concert with USC library clients, who had identified many USC multimedia archives that seemed worthy of transformation into digitized, user-interactive archive management services.

The study showed that the WinWin spiral model is a good match for multimedia applications and is likely to be useful for other applications with similar characteristics—rapidly moving technology, many candidate approaches, little user or developer experience with similar systems, and the need for rapid completion. The study results show that the model has three main strengths.

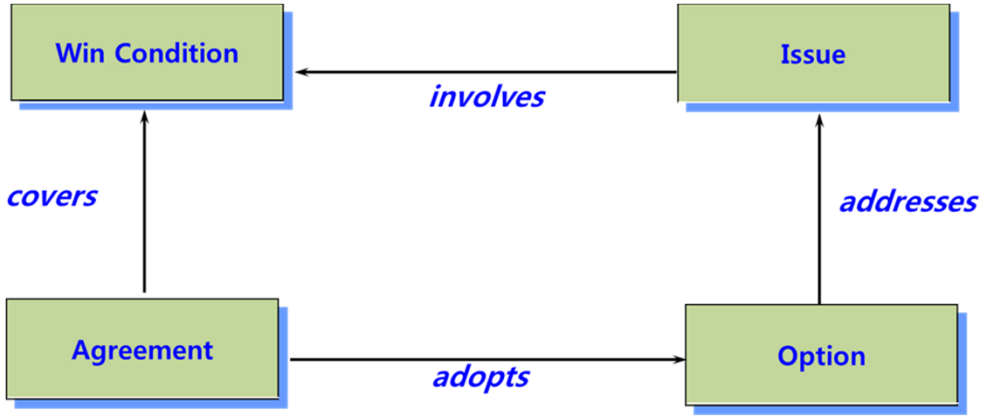
- *Flexibility.* The model let the teams adapt to accompanying risks and uncertainties, such as a rapid project schedule and changing team composition.
- *Discipline.* The modeling framework was sufficiently formal to maintain focus on achieving three main, or "anchor-point," milestones: the life-cycle objectives, the life-cycle architecture, and the initial operational capability. (Table A in the sidebar describes these milestones.)
- *Trust enhancement.* The model provided a means for growing trust among the project stakeholders, enabling them to evolve from adversarial, contract-oriented system development approaches

0018-9162/98/$10.00 © 1998 IEEE

Key Concepts in WinWin

- **Win Condition**: objective which makes a stakeholder feel like a winner
- **Issue**: conflict or constraint on a win condition
- **Option**: a way of overcoming an issue
- **Agreement**: mutual commitment to an option or win condition

- **WinWin Equilibrium State**
  - All Win Conditions are covered by Agreements
  - No outstanding Issues



Steps of WinWin

1. Identify success-critical stakeholders
2. Identify stakeholders' win conditions
3. Identify issues conflicting win conditions
4. Negotiate top-level win-win agreements
   - Invent options for mutual gain
   - Explore option tradeoffs
   - Manage expectations
5. Embody win-win agreements into specs and plans
6. Elaborate steps 1-5 until product is fully developed
   - Confront, resolve new win-lose, lose-lose risk items

http://csse.usc.edu/csse/research/easy_win_win/

Exercise 2 : Requirements Workshop – Requirements Negotiation

- Let's do **Requirements Workshop**
  - To develop **a new advanced OOO digital watch next season**

  - **(1) Stakeholder Analysis**
  - **(2) Brainstorming**

  - **(3) Negotiating Requirements (WinWin Negotiation)**
    - **Role playing**
    - Preparation for the Workshop :
      - List-up the requirements derived from the previous requirements workshop
      - Vote for each requirement : O, X
    - Preparation for the WinWin negotiation :
      - For each, clarify the reason (issue) and possible solutions (option) for your "X"
    - Negotiation :
      - Discuss to find out what others think about the requirements I proposed
      - Revise requirements by agreement or discard to reach WinWin equilibrium state

The Brainstorming Result after Consolidation (Discussion)

Stakeholders	Req. ID	Requirements
User (전자시계 동호회 대표)	[1]	시계 사용자 사이에 숫자 야구 게임에 대해 시간/횟수에 대한 스코어링 및 랭킹 시스템 지원
		~~수심 20M 방수 지원~~
	[2]	최소 6개월 이상의 배터리 수명 보장
백화점 판매팀장		~~트렌디한 디자인~~
	[3]	extensible hw 에 대한 시연 지원
	[4]	baseball game 외 시계사용법에 대한 메뉴얼 제공
상품기획팀장		~~Watch face 가 변경 가능해야 함~~
	[5]	상품 가격이 20만원이 넘지 않아야 함
	[6]	다양한 언어(최소 10개국) 를 지원해야 함
HW/UX Designer		~~않아야 함~~
	[7]	메뉴 사용이 편리해야 함
	[8]	스트랩 교체 가능
SW Engineer	[9]	1년에 1초 미만의 오차를 보장하는 H/W clock 및 이를 활용하기 위한 인터페이스 지원 필요
	[10]	7-segment display로 표현 가능한 언어에 대해서만 지원 가능
		~~C++를 활용한 개발이 가능하도록 관련 Tool 및 Cross compiler 지원~~
Product Quality Manager	[11]	5명의 테스터가 제품 개발 단계 진행을 위한 테스트를 일주일 이내에 완료될 수 있어야 함.
	[12]	100,000대당 1대의 불량률을 구현할 수 있어야 함
		~~시장 문제 대응 비용을 최소화할 수 있어야 함~~
CEO	[13]	Smart Watch 시장의 시장 점유율을 20%이상 점유할 수 있어야 함
	[14]	10억 이내의 개발비로 개발이 완료되어야 함
		~~영업 이익률을 30% 이상 가져갈 수 있어야 함~~
HW Engineer	[15]	Additional module 은 3개 이하로 제한
		~~Camera 는 1개만 탑재 가능~~
	[16]	배터리 교체는 불가능
UI Designer	[17]	color 지원
		~~360X360 해상도 이상 지원~~
	[18]	animatable ui 지원
SW Tester		~~hw features 를 테스트하기 위한 환경이 제공되어야함~~
	[19]	시계버튼 반응속도/화면전환 속도가 타사 digital 시계 대비 우수해야함
	[20]	배터리 사용시간이 idle 상태에서 5일이상되어야함

The WinWin Analysis for Each Requirement for Each

Stakeholders	Candidate Requirements								
	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
CEO	O								
Marketing Manager	O								
Project Team Leader	X								
SW Engineer	X								
HW Engineer	△								
Designer	△								
Others, if any									

for each requirement,

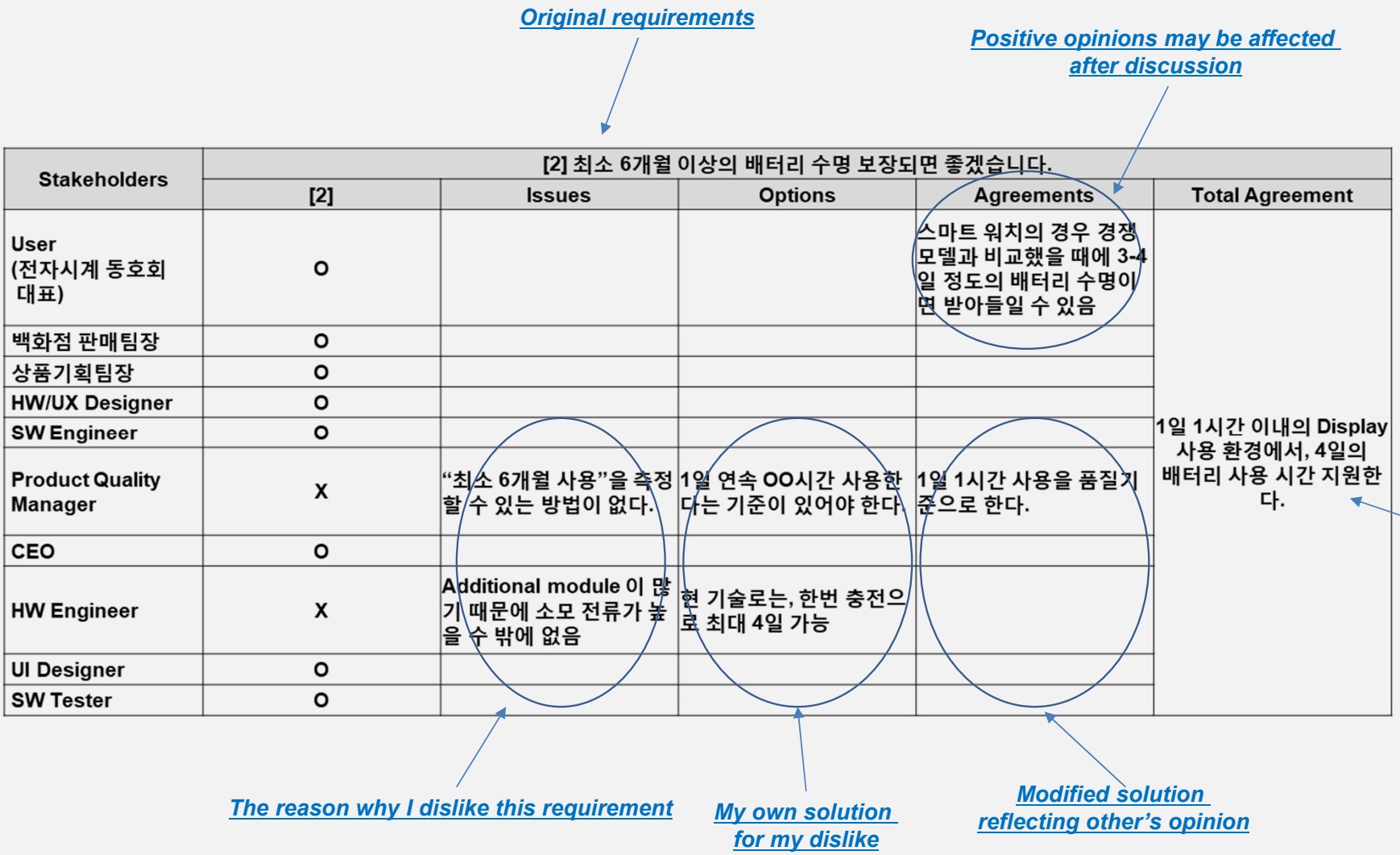
Stakeholders	Candidate Requirement [1]				
	[1]	Issues	Options	Agreements	Total Agreement
CEO	O				Revised Requirement [1]
Marketing Manager	O				
Project Team Leader	X				
SW Engineer	X				
HW Engineer	△				
Designer	△				
Others, if any					

The WinWin Analysis for Each Requirement for Each

Stakeholders	Candidate Requirements																			
	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]
User (전자시계 동호회 대표)	O	O	O	O	O	O	O	O	O	X	O	O	O	O	X	X	O	O	O	O
백화점 판매팀장	O	O	O	O	O	O	O	O	O	X	O	O	O	O	X	X	O	O	O	O
상품기획팀장	O	O	O	X	O	O	O	O	O	X	O	O	X	O	X	X	O	O	O	O
HW/UX Designer	O	O	O	O	O	O	O	O	O	O	O	O	O	X	O	O	O	O	O	O
SW Engineer	X	O	O	O	O	X	O	O	O	O	X	X	O	X	O	O	O	X	X	X
Product Quality Manager	O	x	X	O	O	X	O	O	O	O	O	O	O	O	O	O	O	O	X	O
CEO	O	O	X	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
HW Engineer	O	X	O	O	X	O	O	X	O	O	O	X	O	X	O	O	X	O	O	X
UI Designer	X	O	O	X	O	X	O	O	O	X	O	O	O	O	O	O	O	O	O	O
SW Tester	X	O	O	O	O	X	O	O	O	O	X	O	O	X	O	O	X	X	O	O

Stakeholders	[2] 최소 6개월 이상의 배터리 수명 보장되면 좋겠습니다.				
	[2]	Issues	Options	Agreements	Total Agreement
User (전자시계 동호회 대표)	O			스마트 워치의 경우 경쟁 모델과 비교했을 때에 3-4일 정도의 배터리 수명이면 받아들일 수 있음	1일 1시간 이내의 Display 사용 환경에서, 4일의 배터리 사용 시간 지원한다.
백화점 판매팀장	O				
상품기획팀장	O				
HW/UX Designer	O				
SW Engineer	O				
Product Quality Manager	X	“최소 6개월 사용”을 측정할 수 있는 방법이 없다.	1일 연속 OO시간 사용한다는 기준이 있어야 한다.	1일 1시간 사용을 품질기준으로 한다.	
CEO	O				
HW Engineer	X	Additional module 이 많기 때문에 소모 전류가 높을 수 밖에 없음	현 기술로는, 한번 충전으로 최대 4일 가능		
UI Designer	O				
SW Tester	O				

The WinWin Negotiation Table



Stakeholders	[2] 최소 6개월 이상의 배터리 수명 보장되면 좋겠습니다.				
	[2]	Issues	Options	Agreements	Total Agreement
User (전자시계 동호회 대표)	O			스마트 워치의 경우 경쟁 모델과 비교했을 때에 3-4일 정도의 배터리 수명이면 받아들일 수 있음	1일 1시간 이내의 Display 사용 환경에서, 4일의 배터리 사용 시간 지원한다.
백화점 판매팀장	O				
상품기획팀장	O				
HW/UX Designer	O				
SW Engineer	O				
Product Quality Manager	X	"최소 6개월 사용"을 측정할 수 있는 방법이 없다.	1일 연속 OO시간 사용한다는 기준이 있어야 한다	1일 1시간 사용을 품질기준으로 한다.	
CEO	O				
HW Engineer	X	Additional module 이 많기 때문에 소모 전류가 높을 수 밖에 없음	현 기술로는, 한번 충전으로 최대 4일 가능		
UI Designer	O				
SW Tester	O				

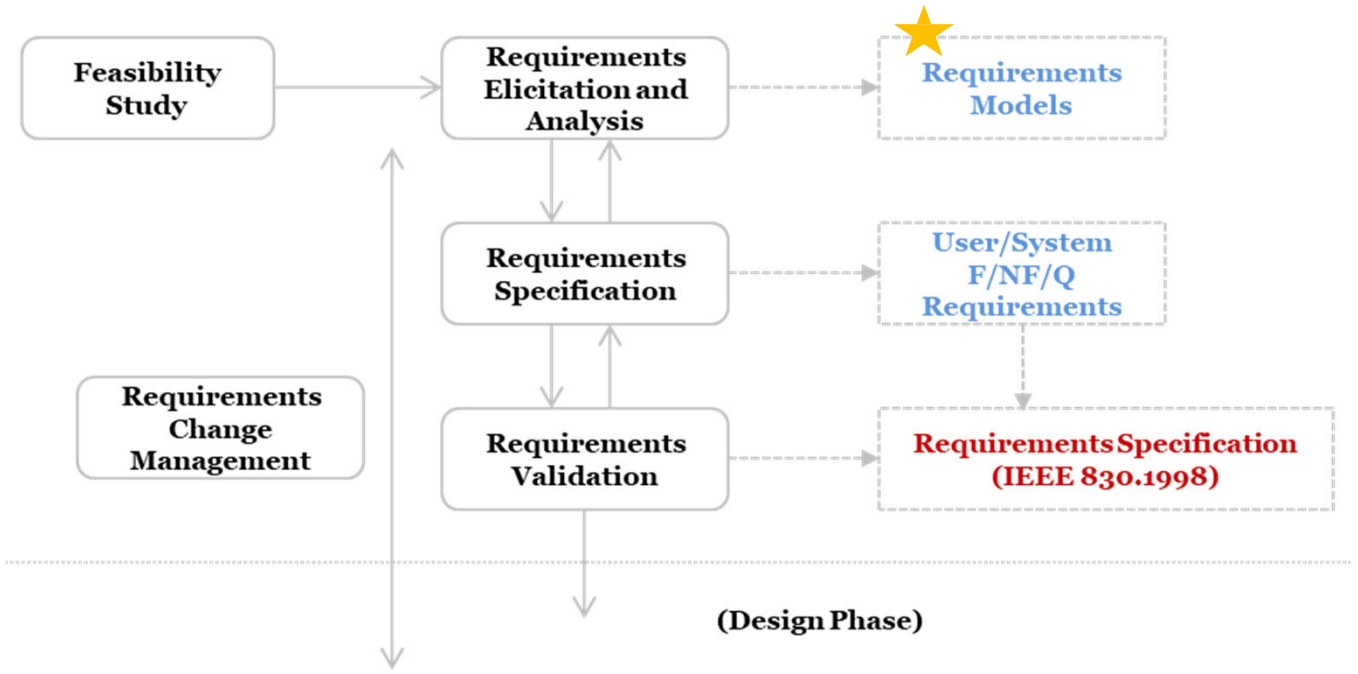
The Negotiation Result (for reference, but not correct)

Stakeholders	Req. ID	Requirements
User (전자시계 동호회 대표)	[1]	숫자 야구 게임에 대해 시간/횟수에 대한 개인 스코어링 및 랭킹 시스템 지원
	[2]	1일 1시간 이내의 Display 사용 환경에서 4일의 베터리 사용 시간 지원
백화점 판매팀장	[3]	전국 주요 대도시 및 10대 백화점 위주 extensible hw 에 대한 시연 지원 (사용자는 관람만 가능)
	[4]	baseball game 외 시계사용법에 대한 온라인 메뉴얼 제공
상품기획팀장	[5]	GPT, LTE, WIfi를 필수로 포함하는 상품 가격이 33만원이 넘지 않아야 함
	[6]	SW 개발 기간을 1개월 연장하는 조건으로, 다양한 언어(최소 10개국) 를 지원해야 함. 4개국 언어 품질 확보 후 이후 지원국가 언어 품질 확보.
HW/UX Designer	[7]	메뉴 사용이 편리해야 함
	[8]	교체 가능한 스트랩을 2개 선출시한다. 선 출시 후 신규 스트랩을 지속적으로 개발한다.
SW Engineer	[9]	1년에 1초 미만의 오차를 보장하는 H/W clock 및 이를 활용하기 위한 인터페이스 지원 필요
	[10]	(Drop. 사유: LCD 필수 요구사항) 7-segment display로 표현 가능한 언어에 대해서만 지원 가능
Product Quality Manager	[11]	5명의 테스터가 제품 개발 단계 진행을 위한 테스트를 8일 이내에 완료될 수 있어야 함. SW 개발자는 11일 내에 버그를 해결해야 한다.
	[12]	HW 불량률은 80,000대당 1대를 구현한다. SW 불량률은 별도로 산정하며 불량 발생시 패치를 통하여 업데이트를 지원한다.
CEO	[13]	Smart Watch 시장의 시장 점유율을 20%이상 점유할 수 있어야 함
	[14]	재료비 기준으로 10억 이내의 개발비로 개발이 완료되어야 함
HW Engineer	[15]	Additional module 은 4개 이하로 제한
	[16]	탈착식 배터리 교환 미지원. 고속 충전 지원 및 서비스 센터를 통한 노후 베터리 교체 지원
UI Designer	[17]	color 지원. (개발 비용 1억 증가, HW 개발 일정 4주 증가, 테스트 일정 1주 증가)
	[18]	animatable ui 지원 (SW 개발자 2명 추가 및 개발 일정 2개월 연장, 테스트 일정 1주 증가)
SW Tester	[19]	시계버튼 반응속도/화면전환 속도가 현존하는 타사 digital 시계 대비 동등 수준 달성
	[20]	배터리 사용시간이 idle 상태에서 5일이상되어야함

6. Requirements Analysis

Requirements Engineering Process

- Requirements analysis through requirements models
- Requirements prioritization
- Requirements selection (Triage)



Requirements Modeling

- **Requirement models** to understand the requirements well
  - Help stakeholders to understand the requirements
  - Guide **elicitation**
  - Provide **a measure of progress**
  - Help to **uncover problems**
  - Help us check our **understanding**

- **Features of good requirements models**
  - **Complete**
    - Modeling guides elicitation
    - Completeness of the model leads to completeness of elicitation
  - **Consistency**
    - Modeling uncovers problems
    - Inconsistency in modeling implies omission, conflict, disagreement and ambiguity
  - **Testability**
    - Modeling checks for expected qualities and predicts end result

A Traditional Survey on Modelling Techniques

- **Modelling Enterprises**
  - Goals & objectives
  - Organizational structure
  - Tasks & dependencies
  - Agents, roles, intentionality

- **Modelling Information & Behavior**
  - Information Structure
  - Behavioral views
    - Scenarios and Use Cases
    - State machine models
    - Information flow
  - Timing/Sequencing requirements

- **Modelling System Qualities (NFRs)**
  - All the 'ilities':
    - usability, reliability, evolvability, safety, security, performance, interoperability,

Organization modelling:
i*, SSM, ISAC
Goal modelling:
KAOS, Korea

Information modelling:
E-R, Class Diagrams
Structured Analysis:
SADT, SSADM, JSD → SASD
Object Oriented Analysis:
OOA, OOSE, OMT → OOAD & UML
Formal Methods:
SCR, NuSCR, Statecharts, MSC, SDL, Z, Larch, VDM…

Quality tradeoffs:
QFD, win-win, AHP
Specific NFRs:
Timed Petri nets (performance)
Task models (usability)
Probabilistic MTTF (reliability)

The State-of-the-Art Requirements Modeling Methods

1. Structured analysis

- Data Flow Diagram (**DFD**)
  + Finite State Machine (**FSM**)
- Entity-Relation Diagram (**ERD**)

FR - Procedural System

Software

2. Use-Case analysis

- Use-Case Modeling (**UC**)
  + Sequence Diagram (**SD**)

FR - Procedural System

FR - Object Oriented System

FR - Business Things

Software

System

3. Goal and Scenario based analysis

- Goal-Scenario Modeling (**GS**)
- Goal-Tree Analysis

FR - All Systems

NFR - Quality

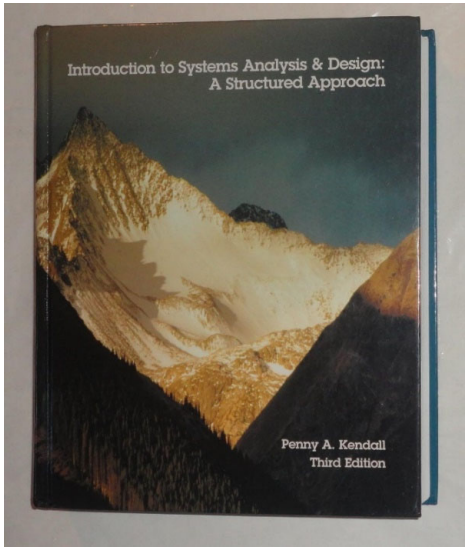
Software

System

1. Structured Analysis

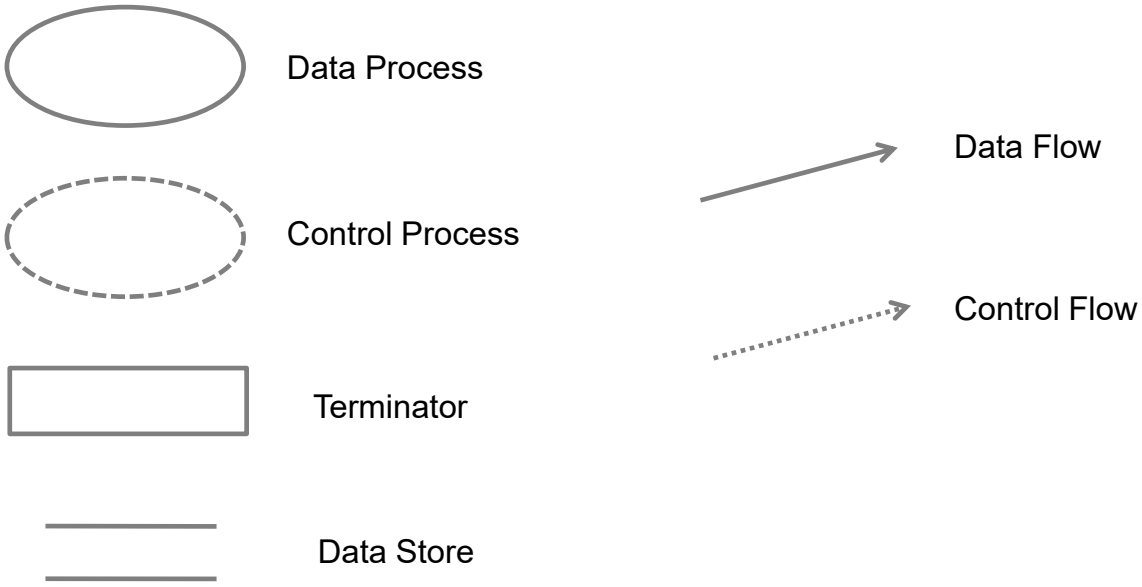
- **Structured analysis** [Kendall 1996]
  - A set of **techniques** and **graphical** tools
    - Allowing the analysts to develop a new kind of system specification that are easily understandable to the users.
  - Data/Functional modeling: **DFD**, **ERD**
  - State-oriented modeling: **STD (FSM)**

- Analysts attempt to divide large, complex problems into smaller, more easily handled ones.
  - **Top-Down Divide and Conquer** approach



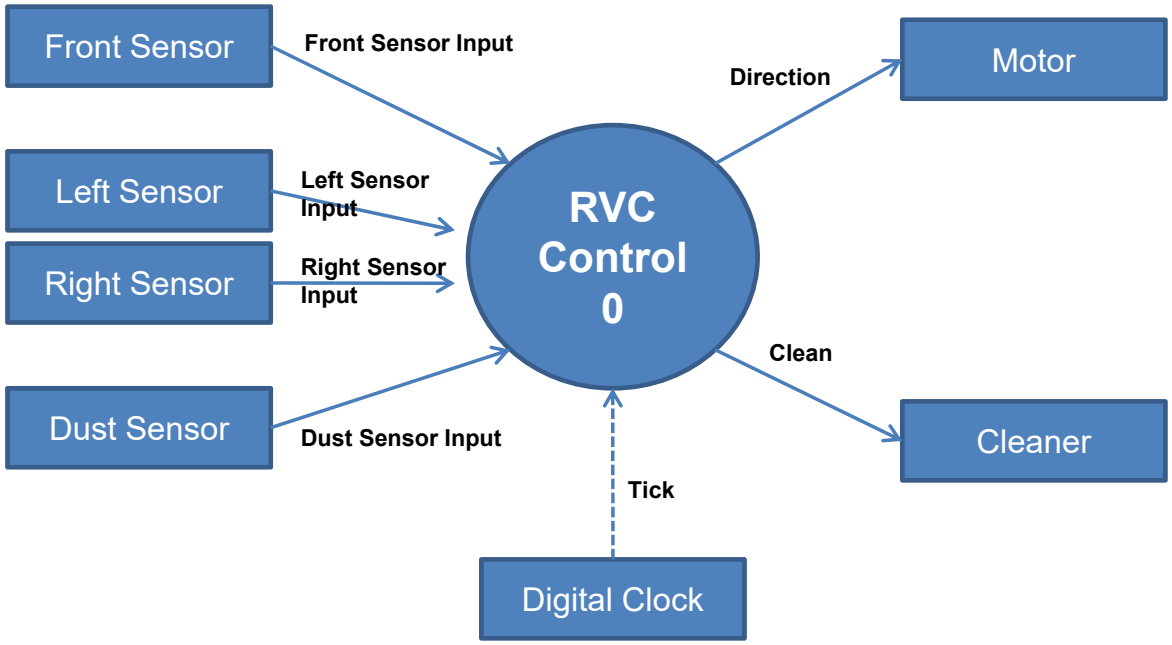
Data Flow Diagram (DFD)

- Provides a means for **functional decomposition**
  - Composed of hierarchies (levels) of DFDs
- Model Elements



DFD Level 0 - RVC Example

- System context diagram

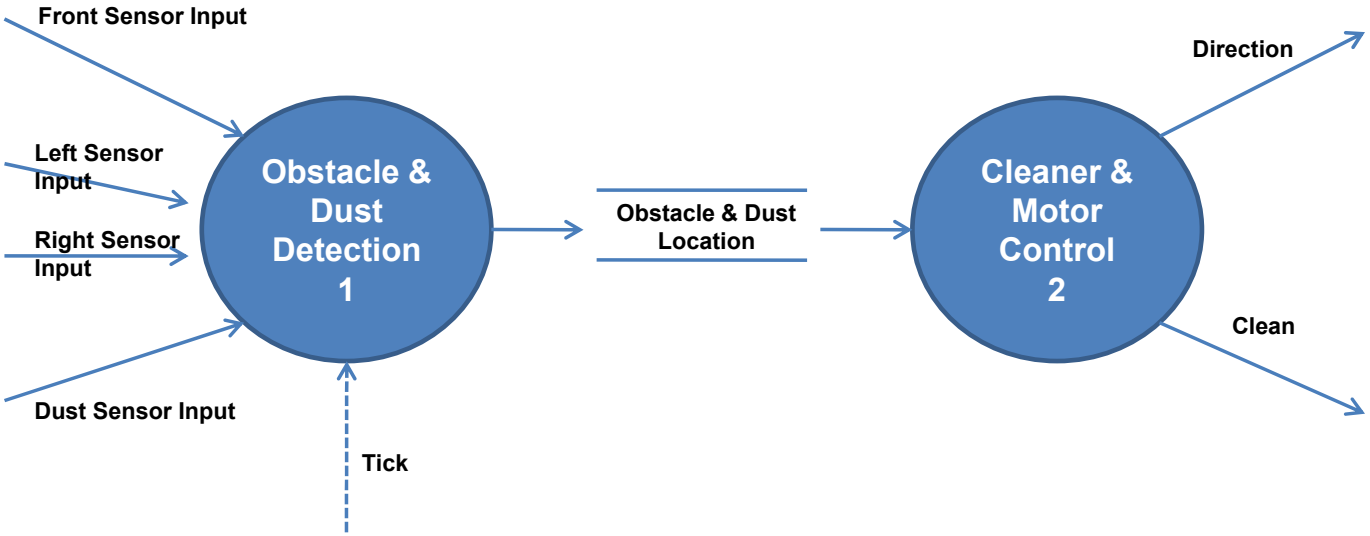


DFD Level 0 - RVC Example

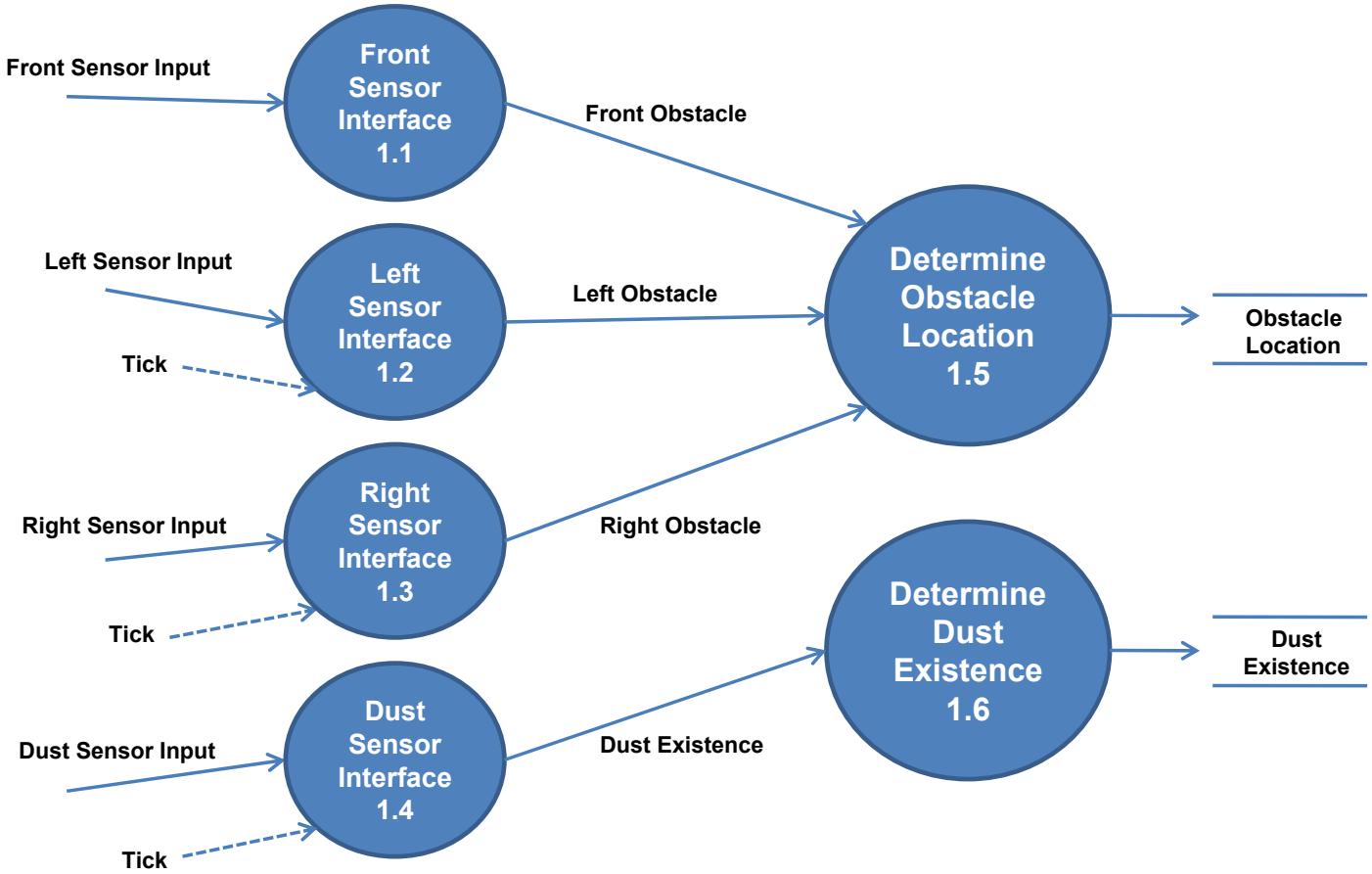
- (A kind of) Data Dictionary

Input/ Output Event	Description	Format / Type
Front Sensor Input	Detects obstacles in front of the RVC	True / False , Interrupt
Left Sensor Input	Detects obstacles in the left side of the RVC periodically	True / False , Periodic
Right Sensor Input	Detects obstacles in the right side of the RVC periodically	True / False , Periodic
Dust Sensor Input	Detects dust on the floor periodically	True / False , Periodic
Direction	Direction commands to the motor (go forward / turn left with an angle / turn right with an angle)	Forward / Left / Right / Stop
Clean	Turn off / Turn on / Power-Up	On / Off / Up

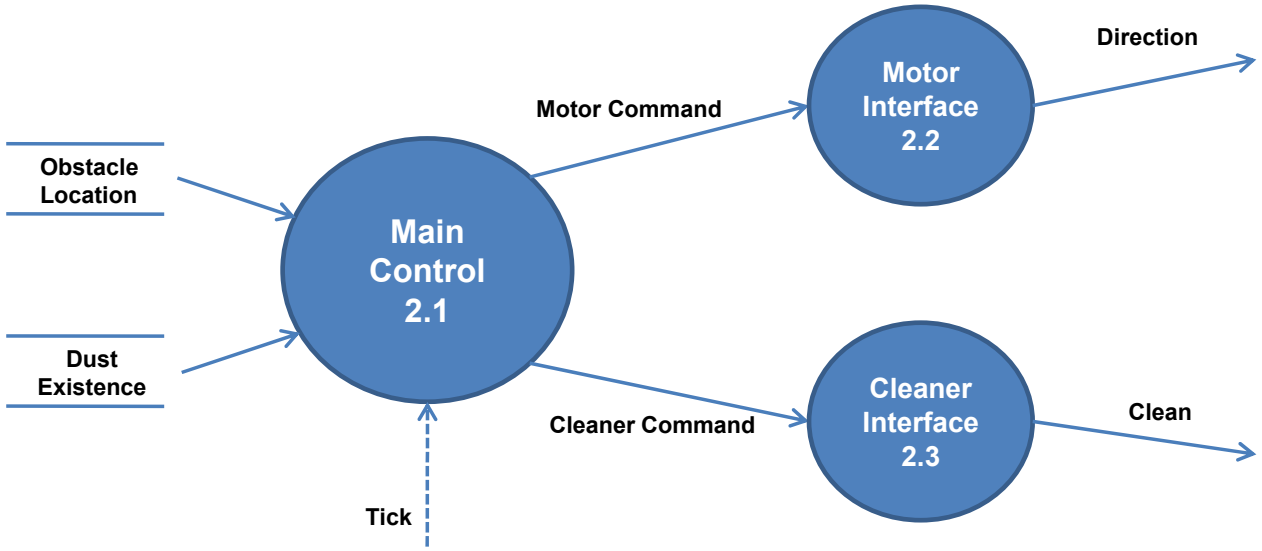
DFD Level 1 - RVC Example



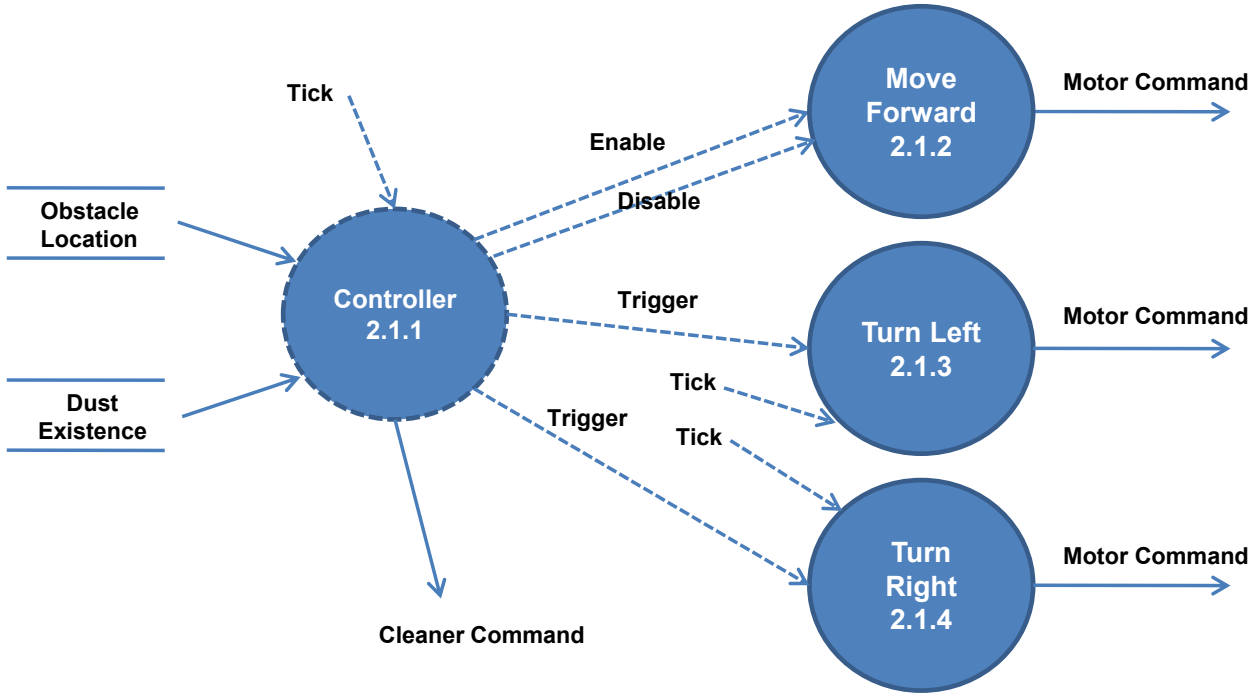
DFD Level 2 - RVC Example



DFD Level 2 - RVC Example

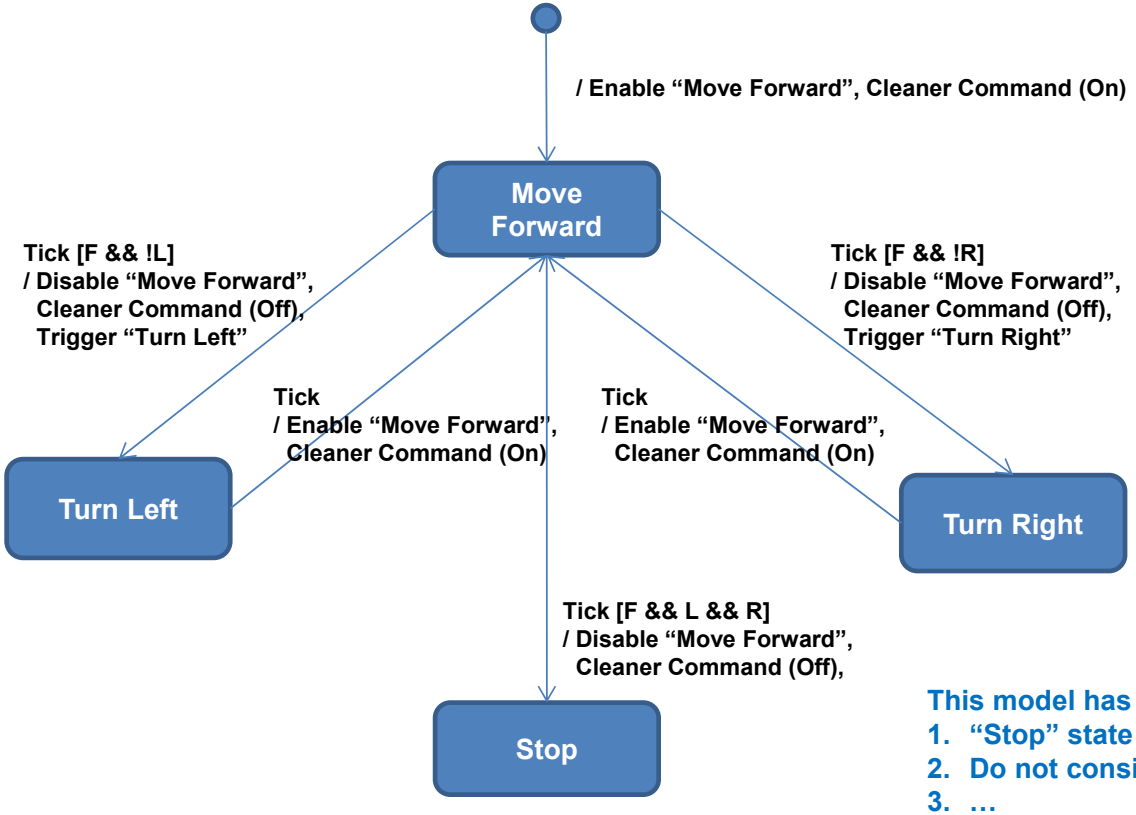


DFD Level 3 - RVC Example



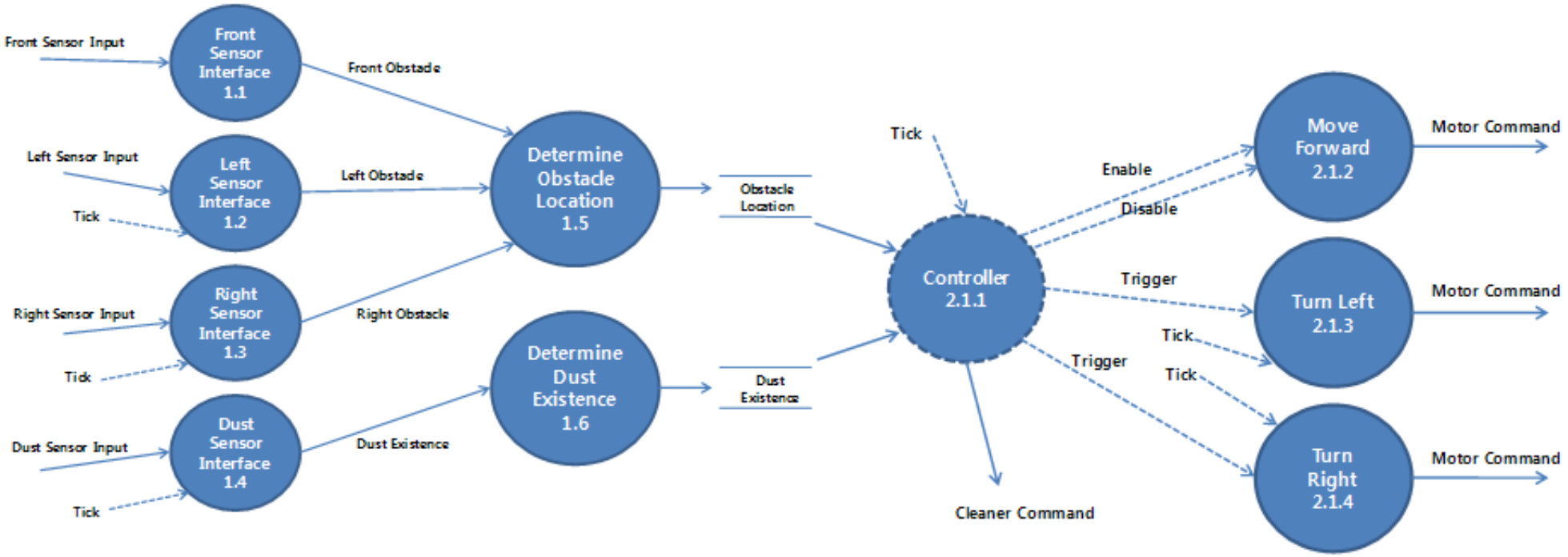
DFD Level 4 - RVC Example

- FSM for Controller 2.1.1



This model has many seeded problems
1. “Stop” state is not normal
2. Do not consider “Dust”
3. …

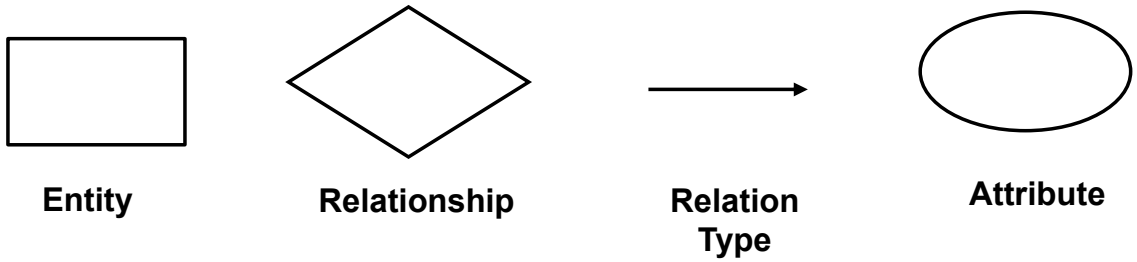
DFD - RVC Example



E-R Modeling

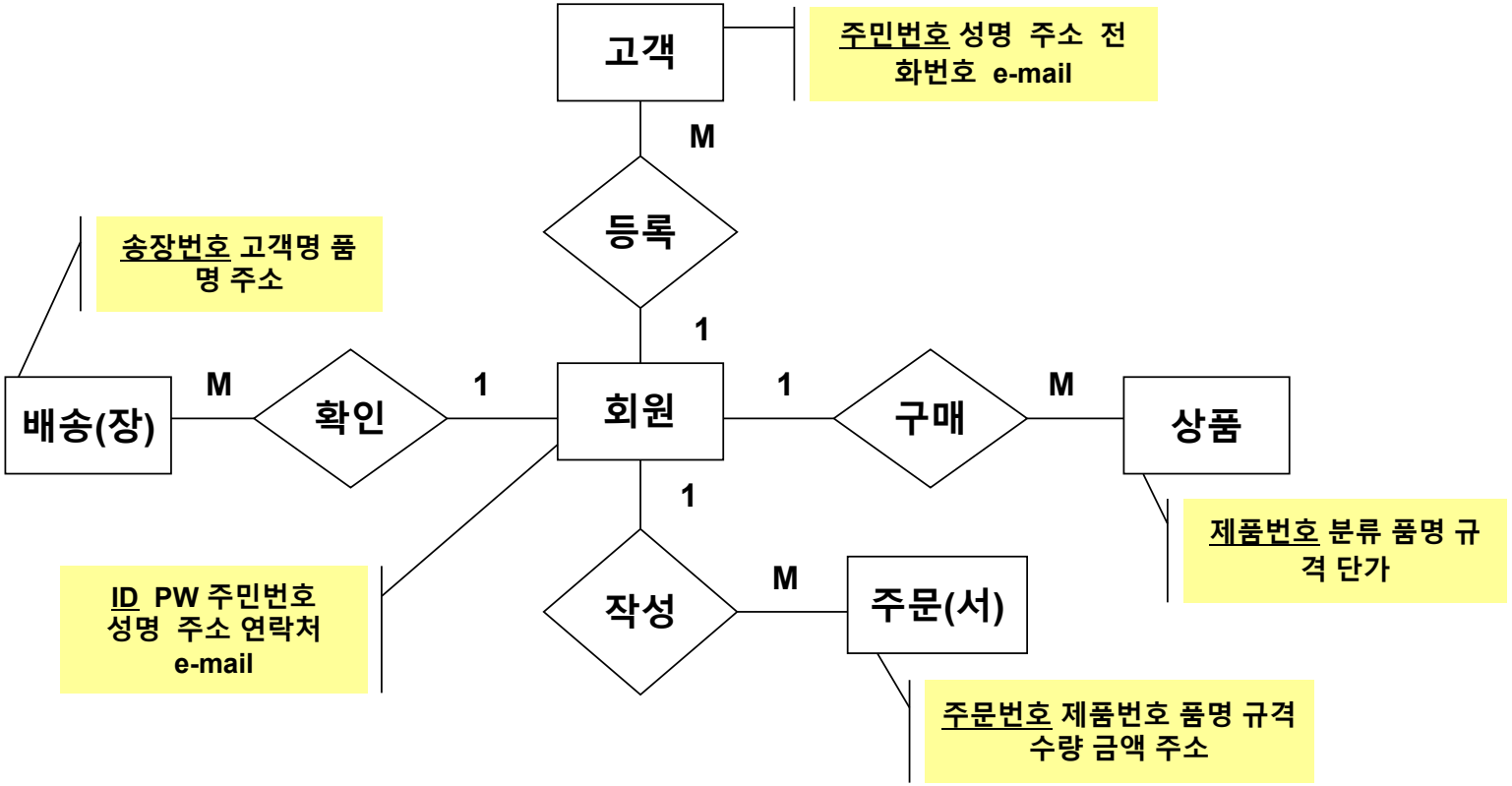
- A graphical representation of the **data layout** of a system at a high level of abstraction
  - Defines data elements and their inter-relationships in the system.
  - Similar with the class diagram in UML.

- Model Elements



E-R Modeling

- Shopping process at Malls



2. Use Case Analysis

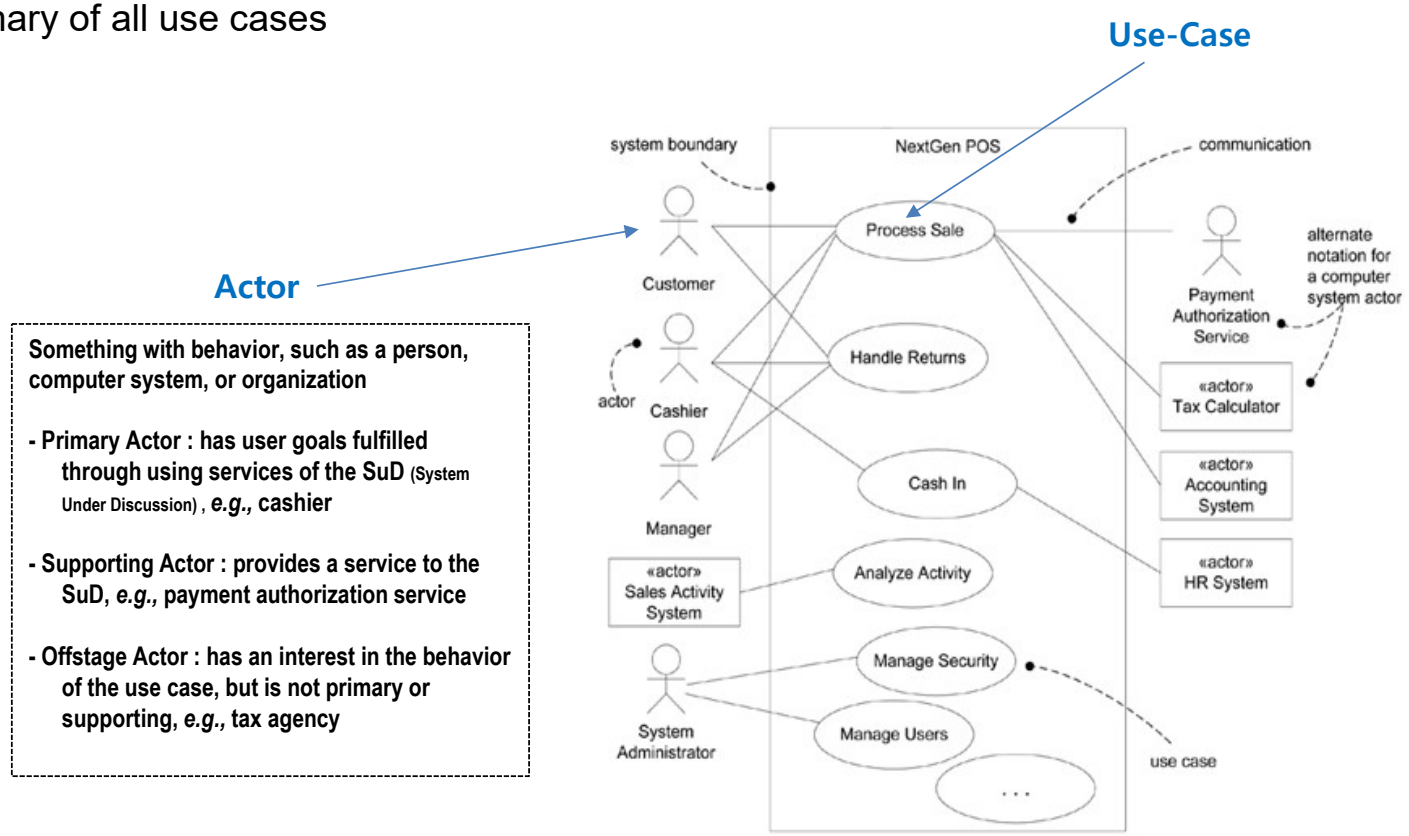
- **Use cases** are **text stories** of some actors using a system to meet goals.
  - A mechanism to capture (analyzes) requirements
  - Use case is not a diagram, but a text.

- Use cases are requirements, primarily **functional (behavioral)** requirements.

Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	"user-goal" or "subfunction"
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, *and* worth telling the reader?
Success Guarantee	What must be true on successful completion, *and* worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Technology and Data Variations List	Varying I/O methods and data formats.
Frequency of Occurrence	Influences investigation, testing, and timing of implementation.
Miscellaneous	Such as open issues.

Use Case Diagram

- **Use case diagram** illustrates the name of use cases and actors, and the relationships between them.
  - System context diagram
  - A summary of all use cases



Three Common Use Case Formats

- **Brief** :
  - Terse one paragraph summary, usually the main success scenario or a happy path

- **Casual** :
  - Informal paragraph format.
  - Multiple paragraphs that cover various scenarios.
    - Main
    - Alternatives
    - Exceptional

Handle Returns

Main Success Scenario: A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item ...

Alternate Scenarios:

If the customer paid by credit, and the reimbursement transaction to their credit account is rejected, inform the customer and pay them with cash.

If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps it is corrupted).

If the system detects failure to communicate with the external accounting system, ...

- **Fully Dressed** :
  - Includes all steps, variations and supporting sections (preconditions, postconditions)

Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	"user-goal" or "subfunction"
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, *and* worth telling the reader?
Success Guarantee	What must be true on successful completion, *and* worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Technology and Data Variations List	Varying I/O methods and data formats.
Frequency of Occurrence	Influences investigation, testing, and timing of implementation.
Miscellaneous	Such as open issues.

Example: Process Sale, Fully Dressed Style

Use Case UC1: Process Sale

Scope: NextGen POS application
Level: user goal
Primary Actor: Cashier
Stakeholders and Interests:
– Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.
– Salesperson: Wants sales commissions updated.
– Customer: Wants purchase and fast service with minimal effort. Wants easily visible display of entered items and prices. Wants proof of purchase to support returns.
– Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.
– Manager: Wants to be able to quickly perform override operations, and easily debug Cashier problems.
– Government Tax Agencies: Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.
– Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.
Preconditions: Cashier is identified and authenticated.
Success Guarantee (or Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario (or Basic Flow):
1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

*a. At any time, Manager requests an override operation:
    1. System enters Manager-authorized mode.
    2. Manager or Cashier performs one Manager-mode operation. e.g., cash balance change, resume a suspended sale on another register, void a sale, etc.
    3. System reverts to Cashier-authorized mode.

*b. At any time, System fails:
    To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.
    1. Cashier restarts System, logs in, and requests recovery of prior state.
    2. System reconstructs prior state.
        2a. System detects anomalies preventing recovery:
            1. System signals error to the Cashier, records the error, and enters a clean state.
            2. Cashier starts a new sale.

1a. Customer or Manager indicate to resume a suspended sale.
    1. Cashier performs resume operation, and enters the ID to retrieve the sale.
    2. System displays the state of the resumed sale, with subtotal.
        2a. Sale not found.
            1. System signals error to the Cashier.
            2. Cashier probably starts new sale and re-enters all items.
    3. Cashier continues with sale (probably entering more items or handling payment).

2-4a. Customer tells Cashier they have a tax-exempt status (e.g., seniors, native peoples)
    1. Cashier verifies, and then enters tax-exempt status code.
    2. System records status (which it will use during tax calculations)

3a. Invalid item ID (not found in system):
    1. System signals error and rejects entry.
    2. Cashier responds to the error:
        2a. There is a human-readable item ID (e.g., a numeric UPC):
            1. Cashier manually enters the item ID.
            2. System displays description and price.
                2a. Invalid item ID: System signals error. Cashier tries alternate method.
        2b. There is no item ID, but there is a price on the tag:
            1. Cashier asks Manager to perform an override operation.
            2. Managers performs override.
            3. Cashier indicates manual price entry, enters price, and requests standard taxation for this amount (because there is no product information, the tax engine can't otherwise deduce how to tax it)
        2c. Cashier performs Find Product Help to obtain true item ID and price.
        2d. Otherwise, Cashier asks an employee for the true item ID or price, and does either manual ID or manual price entry (see above).

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):
    1. Cashier can enter item category identifier and the quantity.

3c. Item requires manual category and price entry (such as flowers or cards with a price on them):
    1. Cashier enters special manual category code, plus the price.

3-6a: Customer asks Cashier to remove (i.e., void) an item from the purchase:
    This is only legal if the item value is less than the void limit for Cashiers, otherwise a Manager override is needed.
    1. Cashier enters item identifier for removal from sale.
    2. System removes item and displays updated running total.
        2a. Item price exceeds void limit for Cashiers:
            1. System signals error, and suggests Manager override.
            2. Cashier requests Manager override, gets it, and repeats operation.

3-6b. Customer tells Cashier to cancel sale:
    1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:
    1. System records sale so that it is available for retrieval on any POS register.
    2. System presents a "suspend receipt" that includes the line items, and a sale ID used to retrieve and resume the sale.

4a. The system supplied item price is not wanted (e.g., Customer complained about something and is offered a lower price):
    1. Cashier requests approval from Manager.
    2. Manager performs override operation.
    3. Cashier enters manual override price.
    4. System presents new price.

5a. System detects failure to communicate with external tax calculation system service:
    1. System restarts the service on the POS node, and continues.
        1a. System detects that the service does not restart.
            1. System signals error.
            2. Cashier may manually calculate and enter the tax, or cancel the sale.

5b. Customer says they are eligible for a discount (e.g., employee, preferred customer):
    1. Cashier signals discount request.
    2. Cashier enters Customer identification.
    3. System presents discount total, based on discount rules.

5c. Customer says they have credit in their account, to apply to the sale:
    1. Cashier signals credit request.
    2. Cashier enters Customer identification.
    3. Systems applies credit up to price=0, and reduces remaining credit.

6a. Customer says they intended to pay by cash but don't have enough cash:
    1. Cashier asks for alternate payment method.
        1a. Customer tells Cashier to cancel sale. Cashier cancels sale on System.

7a. Paying by cash:
1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:
1. Customer enters their credit account information.
2. System displays their payment for verification.
3. Cashier confirms.
   3a. Cashier cancels payment step:
      1. System reverts to "item entry" mode.
4. System sends payment authorization request to an external Payment Authorization Service System, and requests payment approval.
   4a. System detects failure to collaborate with external system:
      1. System signals error to Cashier.
      2. Cashier asks Customer for alternate payment.
5. System receives payment approval, signals approval to Cashier, and releases cash drawer (to insert signed credit payment receipt).
   5a. System receives payment denial:
      1. System signals denial to Cashier.
      2. Cashier asks Customer for alternate payment.
   5b. Timeout waiting for response.
      1. System signals timeout to Cashier.
      2. Cashier may try again, or ask Customer for alternate payment.
6. System records the credit payment, which includes the payment approval.
7. System presents credit payment signature input mechanism.
8. Cashier asks Customer for a credit payment signature. Customer enters signature.
9. If signature on paper receipt, Cashier places receipt in cash drawer and closes it.

7c. Paying by check…

7d. Paying by debit…

7e. Cashier cancels payment step:
1. System reverts to "item entry" mode.

7f. Customer presents coupons:
1. Before handling payment, Cashier records each coupon and System reduces price as appropriate. System records the used coupons for accounting reasons.
   1a. Coupon entered is not for any purchased item:
      1. System signals error to Cashier.

9a. There are product rebates:
1. System presents the rebate forms and rebate receipts for each item with a rebate.

9b. Customer requests gift receipt (no prices visible):
1. Cashier requests gift receipt and System presents it.

9c. Printer out of paper.
1. If System can detect the fault, will signal the problem.
2. Cashier replaces paper.
3. Cashier requests another receipt.

Special Requirements:
– Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.
– Credit authorization response within 30 seconds 90% of the time.
– Somehow, we want robust recovery when access to remote services such the inventory system is failing.
– Language internationalization on the text displayed.
– Pluggable business rules to be insertable at steps 3 and 7.
– . . .

Technology and Data Variations List:
*a. Manager override entered by swiping an override card through a card reader, or entering an authorization code via the keyboard.
3a. Item identifier entered by bar code laser scanner (if bar code is present) or keyboard.
3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.
7a. Credit account information entered by card reader or keyboard.
7b. Credit payment signature captured on paper receipt. But within two years, we predict many customers will want digital signature capture.

Frequency of Occurrence: Could be nearly continuous.

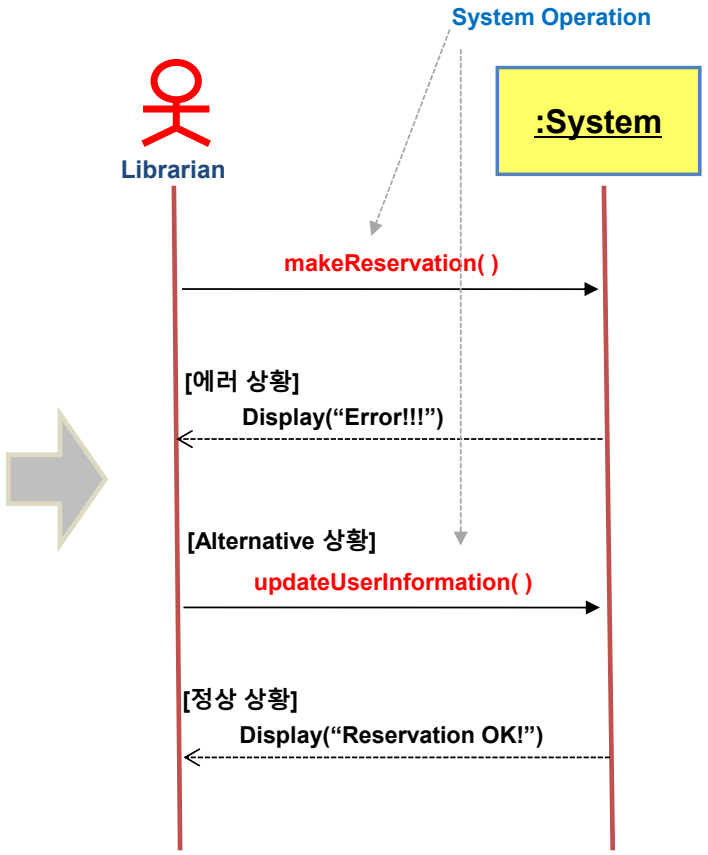
Open Issues:
– What are the tax law variations?
– Explore the remote service recovery issue.
– What customization is needed for different businesses?
– Must a cashier take their cash drawer when they log out?
– Can the customer directly use the card reader, or does the cashier have to do it?

Use-Case (Brief)

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.

Use-Case (Casual)

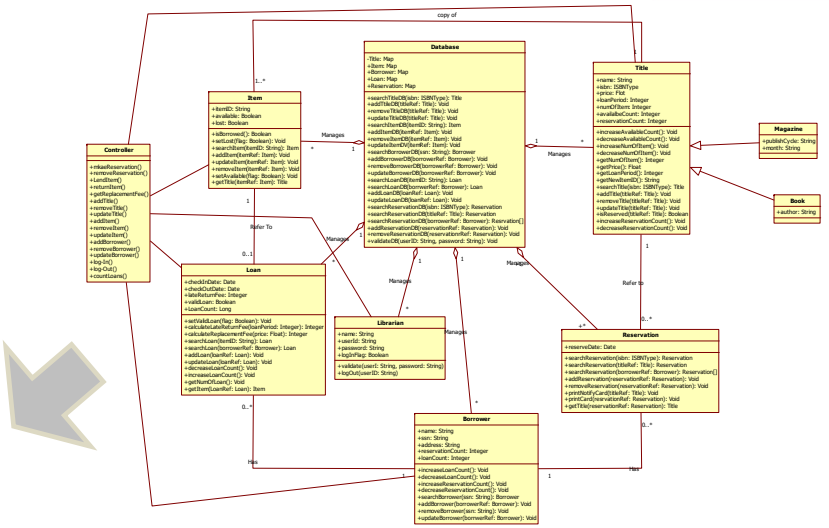
Use Case	1. Make Reservation
Actor	Librarian **(Evident)**
Purpose	(As in the Inception)
Overview	(As in the Inception)
Type	Primary and Casual
Cross Reference	System Functions: R1.1, R3.1 Use Case: “Add Borrower”
Pre-Requisites	Borrower should have an id_card.
Typical Courses of Events	(A) : Actor, (S) : System 1. (A) A librarian requests the reservation of a title 2. (S) Check if a corresponding title exists 3. (S) Check if a corresponding borrower exists 4. (S) Create a reservation information
Alternative Courses of Events	Line 3: (S) If the borrower’s information is out of date, request for the update. (A) A librarian updates up-to-date information of the borrower.
Exceptional Courses of Events	Line 1~3: If invalid reservation information is entered, indicate an error.



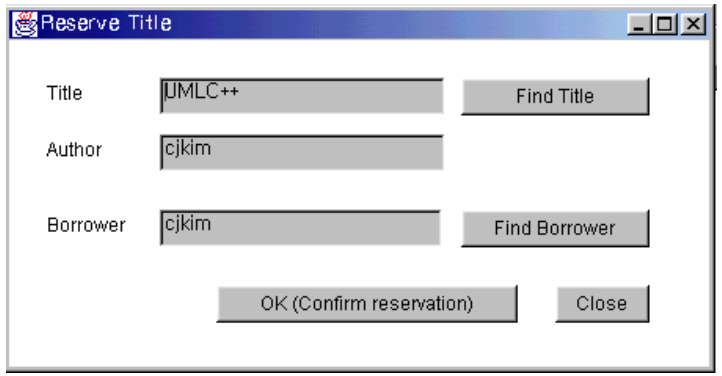
System Sequence Diagram (SSD)

Use-Case (Fully Dressed)

Use Case	1. Make Reservation
Actor	Librarian
Purpose	Create a new reservation
Overview	(As before)
Type	Primary and Fully-Dressed
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	Line 3: (S) If the borrower’s information is out of date, request for the update. (A) A librarian updates up-to-date information of the borrower.
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.

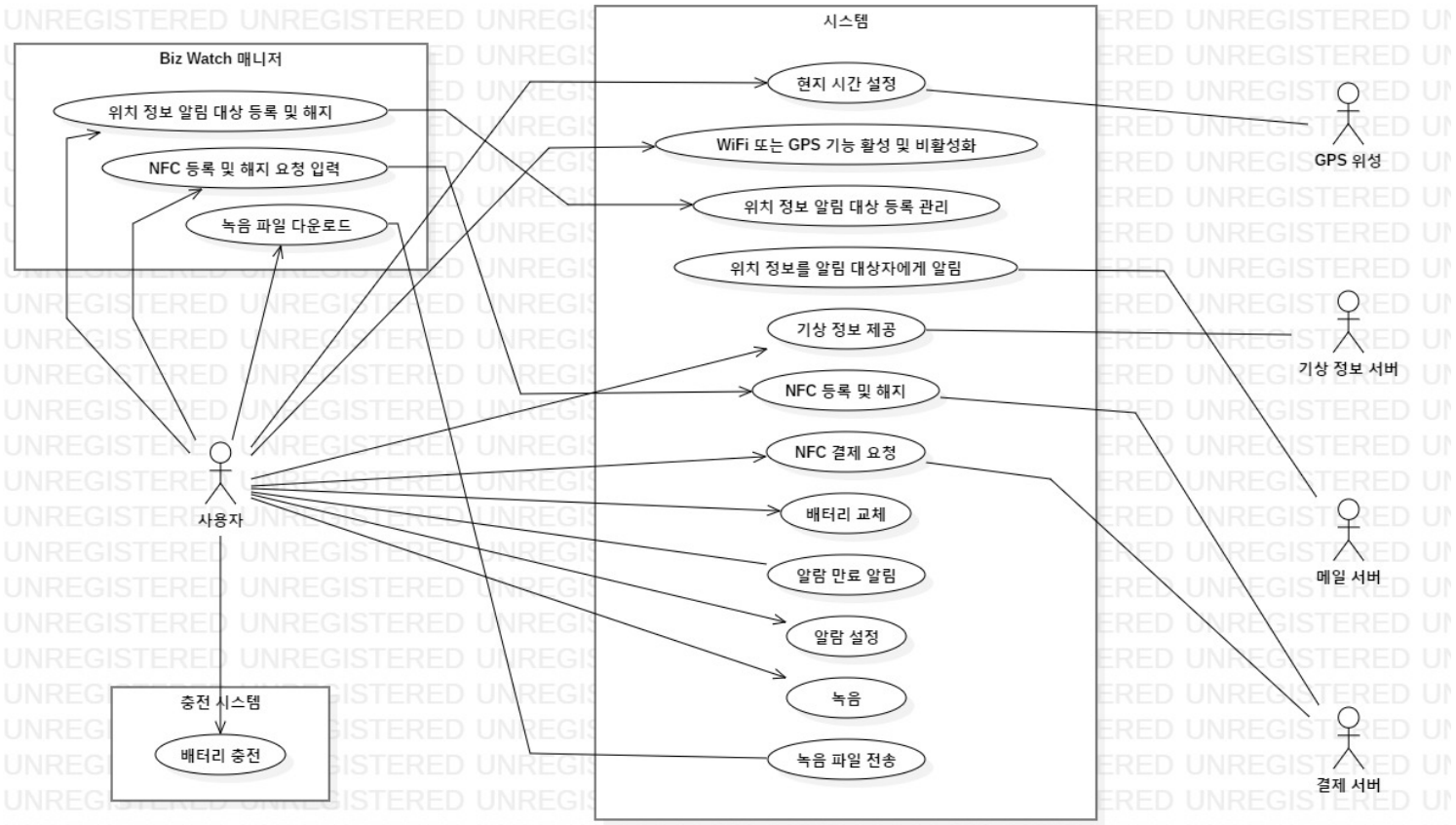


Design Class Diagram



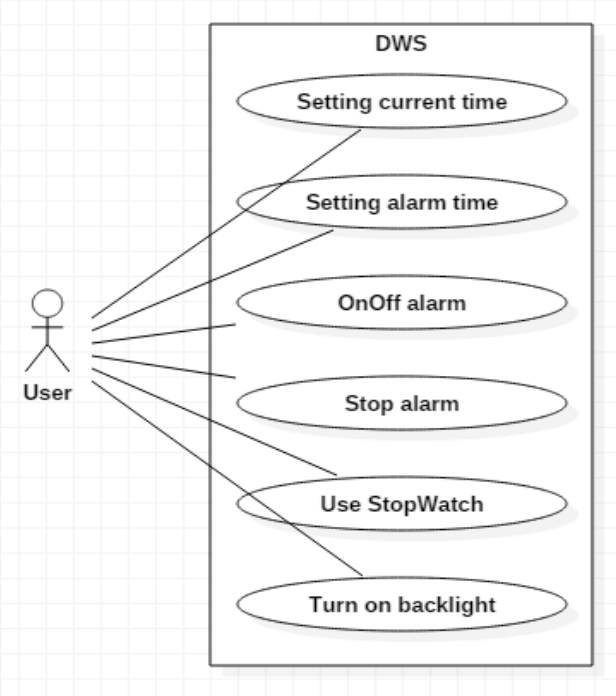
UI (Prototype)

Use-Case Diagrams : An Example with Subsystems



Exercise 3 : Use Case Analysis

- **Identify actors and use cases** for **the new OOO advanced digital watch**
  - Sketch a **use-case diagram** and **descriptions** for some important use cases, as detail as possible (casual format).
  - Use a UML tool
  - Each use case should link to user requirements defined at the Exercise 1 and 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.	

Use-Case (Brief)

Use Case	2. Set Current time
Actors	User
Description	- 연/월/일/요일/시간을 설정한다 - 특정키 입력시 연 > 월 > 일 > 요일 > 시 > 분 > 초 순으로 입력 순서가 변경된다. - 특정키 입력시 현재 선택된 입력 값이 순서대로 증가한다. - 특정키 입력시 현재 편집된 시간을 적용하고 현재 시간 표시 화면으로 전환한다. - 설정 중 일정시간 동안 입력이 없을 경우 설정을 취소한다.

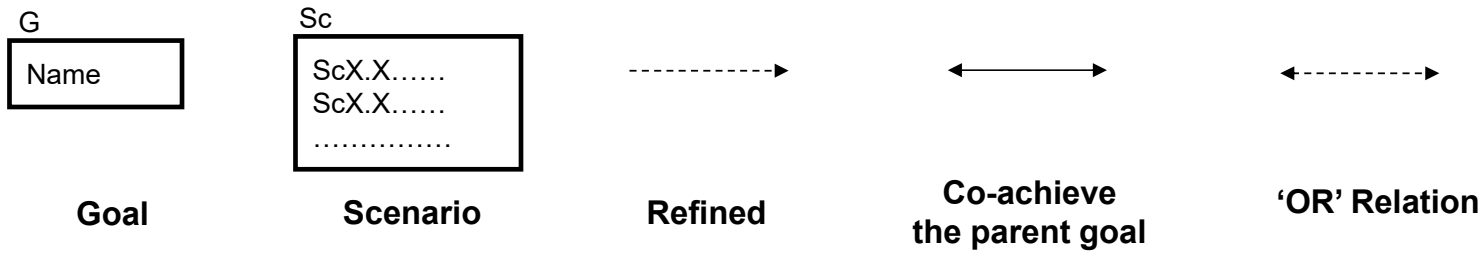
Use-Case (Casual)

Use Case	2. Set Current time
Actors	User
Description	- 연/월/일/요일/시간을 설정한다 - 특정키 입력시 연 > 월 > 일 > 요일 > 시 > 분 > 초 순으로 입력 순서가 변경된다. - 특정키 입력시 현재 선택된 입력 값이 순서대로 증가한다. - 특정키 입력시 현재 편집된 시간을 적용하고 현재 시간 표시 화면으로 전환한다. - 설정 중 일정시간 동안 입력이 없을 경우 설정을 취소한다.
Pre-Requisites	현재 시간 표시 모드일 경우
Typical Courses of Events	1. (A) : 편집모드 버튼을 누른다. 2. (S) : 편집모드 화면으로 전환된다. 3. (S): 커서를 연 위치로 이동한다. 4. (A): 증가 버튼을 눌러 연을 맞춘다. 5. (S) : 증가된 값을 표시한다. 6. (A): 커서 위치를 변경한다. 7. (S) : 커서 위치를 반복한다. 8. 위 내용을 초까지 반복한다. 9. (A) : 저장 버튼을 누룬다. 10. (S) : 시간을 저장하고 현재 시간을 표시한다.
Alternative Courses of Events	Line 1~8 : (S) : 다른모드의 이벤트가 발생함 (A) : 확인 버튼을 누른다. (S) : 설정 화면을 보여준다.
Exceptional Courses of Events	Line 1~8 : (S) : 일정시간 키 입력이 없을 경우 현재 입력된 값을 취소하고 원래 시간을 표시한다. Line 1~8 : (S) : 취소 버튼을 누를 경우 현재 입력된 값을 취소하고 원래 시간을 표시한다.

3. Goal-Scenario Based Analysis

- An analysis using **goal** and **scenario models** to express and refine requirements
  - Provides rationale for the requirements
  - Supports requirements analysis through scenarios
    - Story-line and example-based description
  - Refines goals through scenarios

- Model Elements

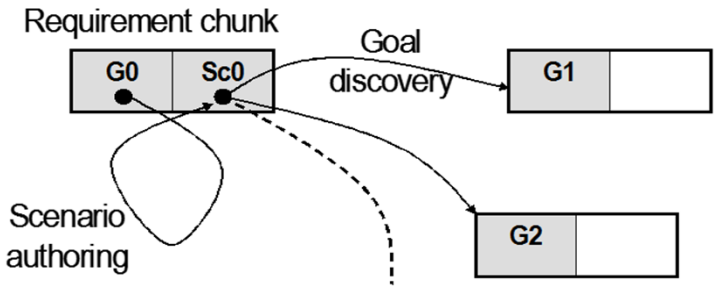


Goal & Scenario

- **Goal**
  - High-level abstraction requirements
  - Example: "*유비쿼터스 기술이 접목된 ATM 서비스를 제공한다.*"

- **Scenario**
  - Purposeful interaction between entities
  - Example: "*사용자는 ATM으로부터 현금을 인출한다.*"

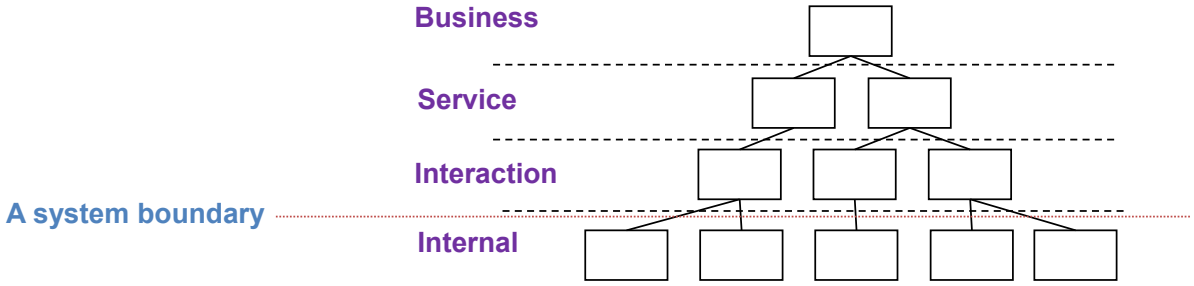
- The relationship between goal and scenario
  - Goals are achieved by scenarios
  - Goals are explained by scenarios
    - Goals are abstract
    - Scenarios are concrete



Goal & Scenario Modeling

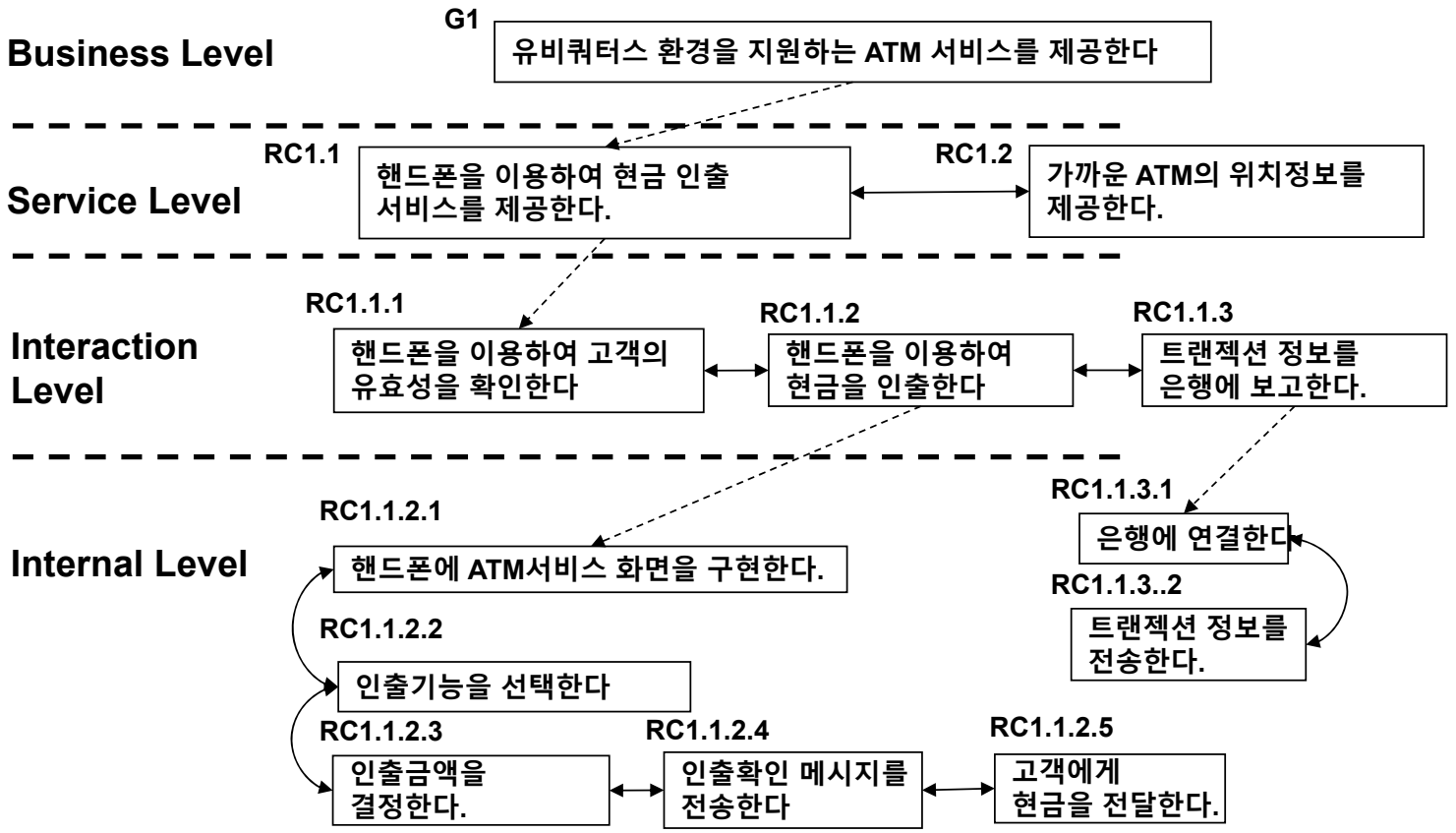
- Inputs: Initial requirements (high-level user requirements)
- Outputs: **Goal-Tree**
  - Abstraction levels provide separation of concern and levels of goal & scenario modeling

- The 4 abstraction levels
  - **Business** : represents the ultimate purpose of a system
  - **Service** : represents the services that a system should provide to an organization and the rationale
  - **Interaction** : represents the interaction between system and external agent (user or external system)
  - **Internal** : represents what the system needs to perform the interactions selected at the user level



Example of G&S Modeling

- A partial example of an ATM system



RC1.1.2

(핸드폰을 이용하여)$_{way}$ (ATM으로부터)$_{direction}$
(현금을)$_{target}$ (인출한다)$_{action}$

SC1.1

1. (ATM이)$_{subject}$ (고객의 핸드폰에)$_{direction}$ (ATM 서비스 화면을)$_{target}$ (전송한다)$_{action}$
2. (핸드폰은)$_{subject}$ (고객에게)$_{direction}$ (ATM서비스 화면을)$_{target}$ (출력한다)$_{action}$
3. (고객이)$_{subject}$ (인출기능을)$_{target}$ (선택한다)$_{action}$
4. (핸드폰이)$_{subject}$ (고객에게)$_{direction}$ (인출금액 입력화면을)$_{target}$ (출력한다)$_{action}$
5. (고객이)$_{subject}$ (핸드폰에)$_{direction}$(인출금액을)$_{target}$ (입력한다)$_{action}$
   만약 인출금액이 올바르면 (state)
6. (핸드폰은)$_{subject}$ (고객에게)$_{direction}$ (인출확인 메세지를)$_{target}$ (전송한다)$_{action}$
7. (ATM이)$_{subject}$ (고객에게)$_{direction}$ (현금을)$_{target}$ (전달한다.)$_{action}$

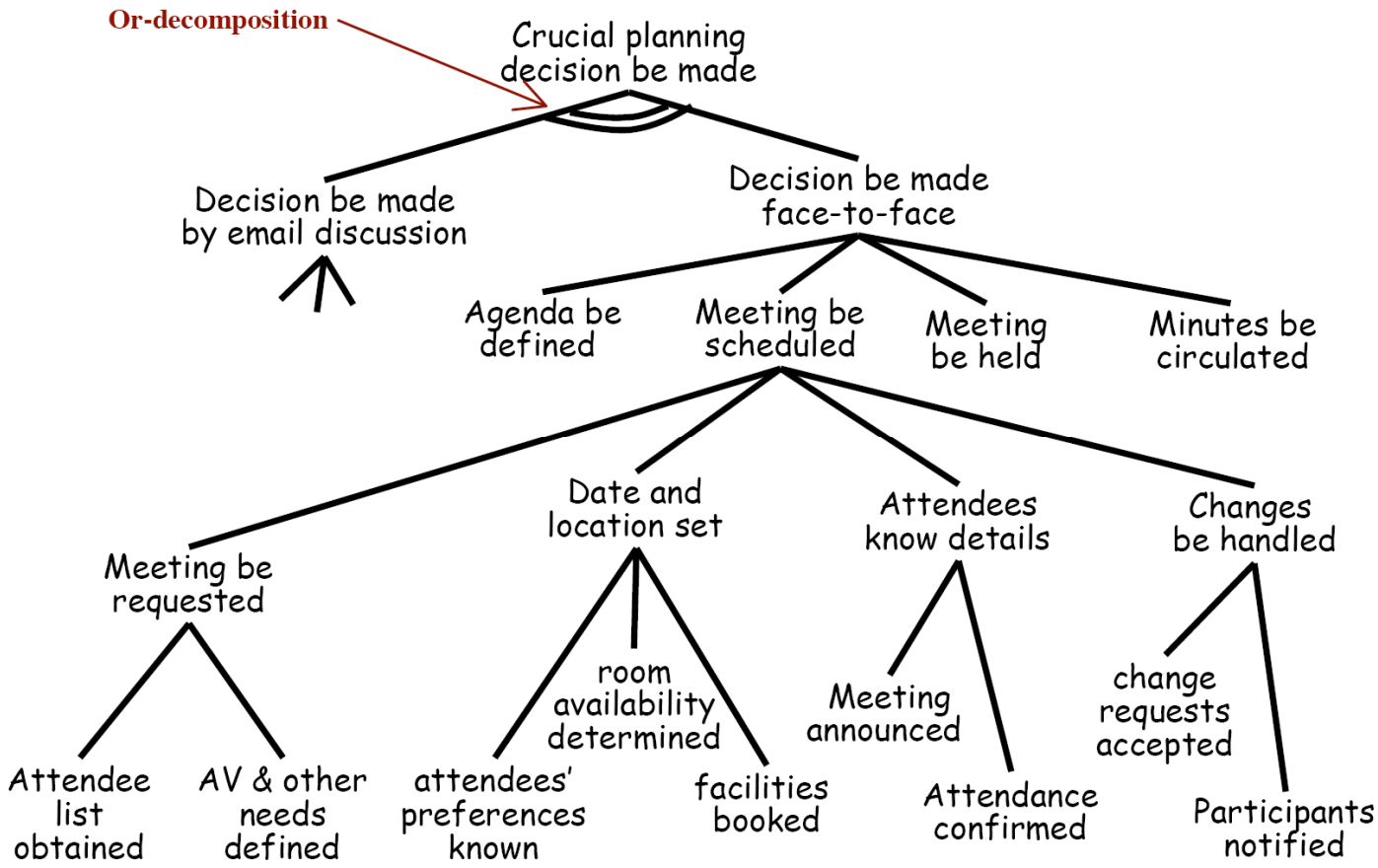
Goal (Goal-Tree) Analysis

- **Goal-Tree Analysis**
  - Focus on *why* a system is required, expressing the 'why' as a set of stakeholder goals
  - Goal refinement to arrive at specific requirements
    - Document, organize and classify goals
  - Goal evolution
    - Refine, elaborate, and operationalize goals
  - Goal hierarchies show refinements and alternatives
  - **Goal-Tree** visualizes goal analysis

- Pros
  - Reasonably intuitive
    - Explicit declaration of goals provides sound basis for conflict resolution

- Cons
  - Captures a static picture - what if goals change over time?
    - Can regress forever up (or down) the goal hierarchy

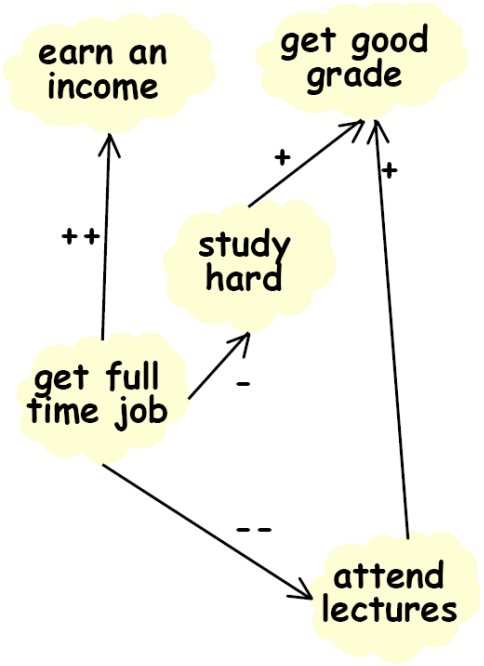
Example : Goa-Tree Modeling



Goal-Tree Analysis

- **Goal Elaboration**
  - "**Why**" questions explore higher goals (context)
  - "**How**" questions explore lower goals (operations)
  - "**How else**" questions explore alternatives

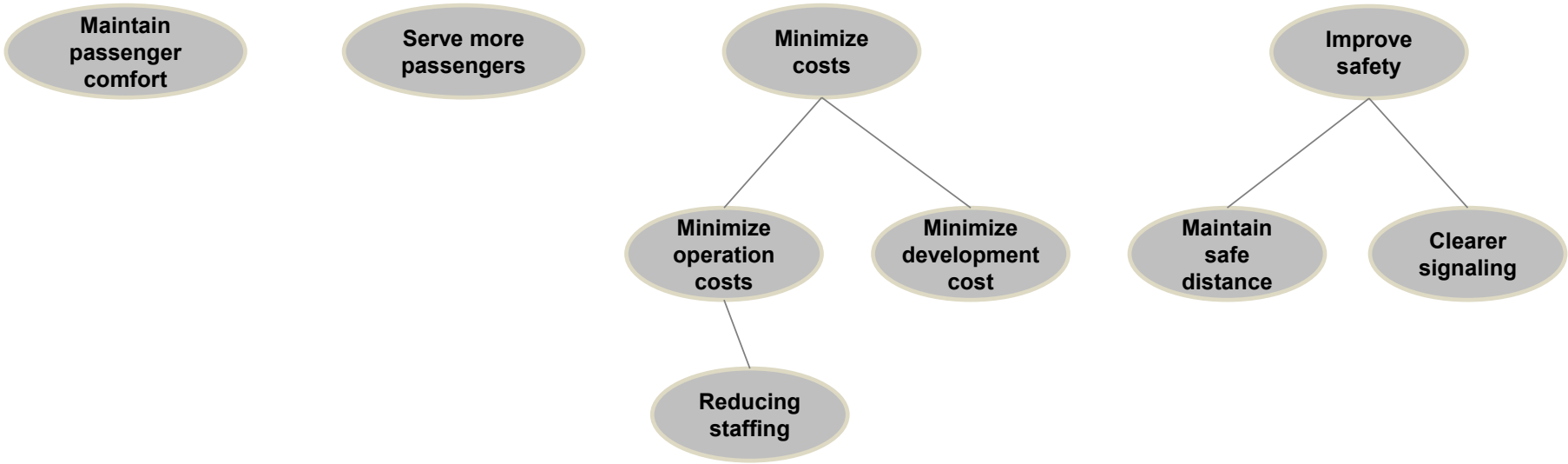
- Relationships between goals
  - One goal helps achieve another (+)
  - One goal hurts achievement of another (-)
  - One goal makes another (++)
    - Achievement of goal A guarantees achievement of goal B
  - One goal break another (--)
    - Achievement of goal A prevents achievement of goal B



Goal-Tree Analysis on Quality Attributes

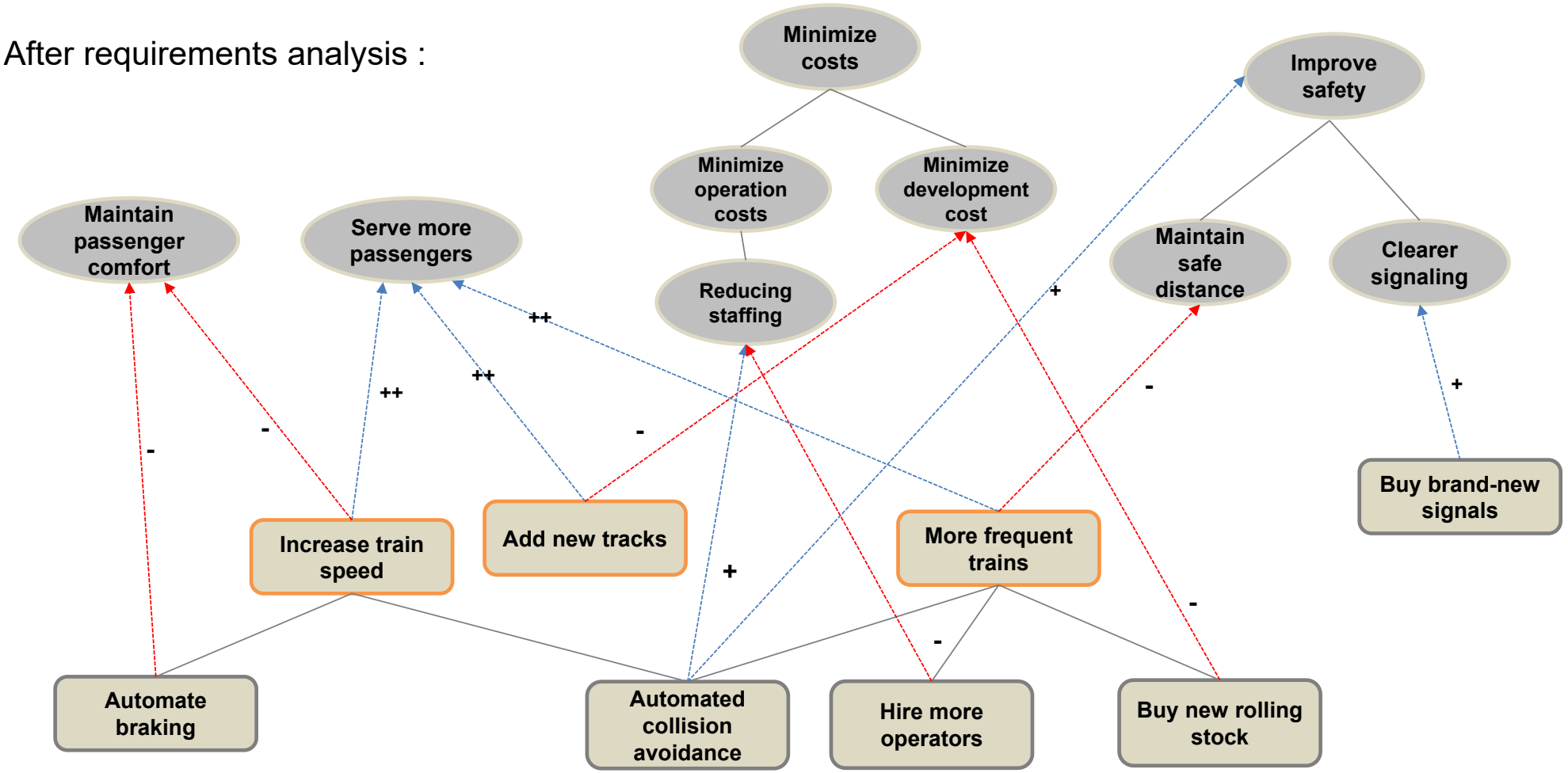
- Quality Attributes for "**Train System**" :
  - Convenience, Benefit, Cost, Safety

- Goals identified from requirements elicitation :



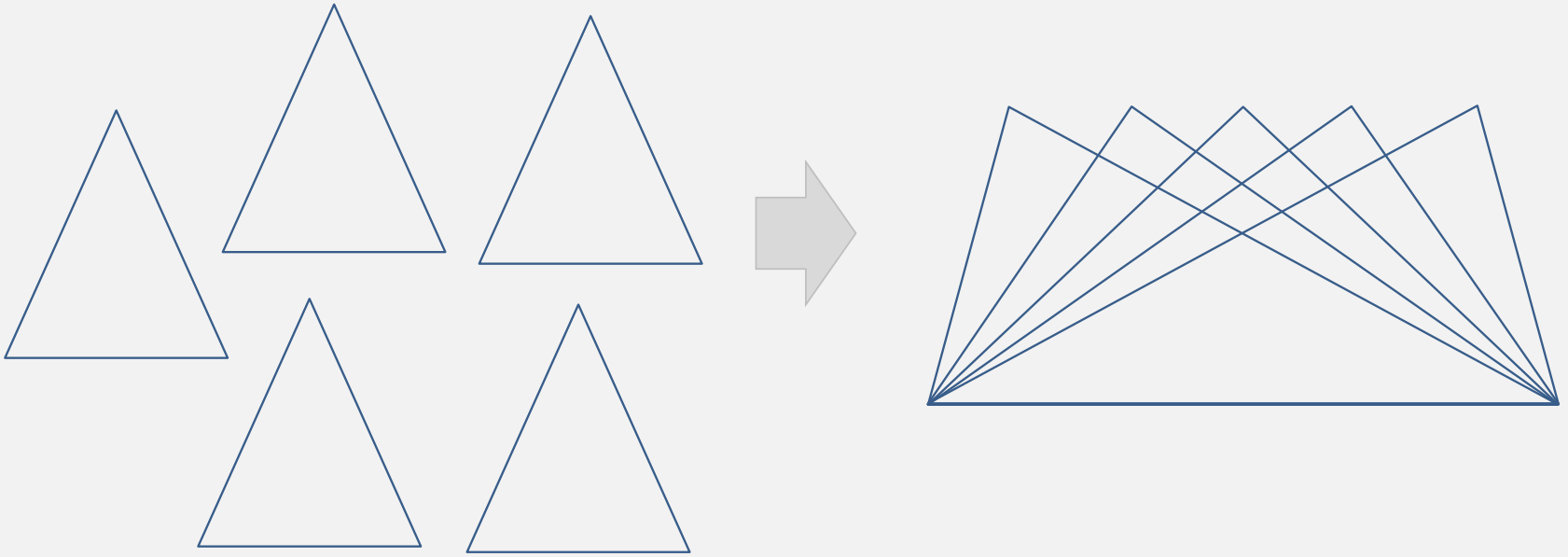
Goal-Tree Analysis

- After requirements analysis :



Exercise 4: Goal-Tree Analysis

- **Perform goal-tree analysis** for the **NEW OOO Advanced Digital Watch**
  - Select 5~6 soft goals(Quality) from the Exercise 1-1 and 2-1
  - Construct a **goal-tree** for each quality goal with the detail **functional requirements** derived from Exercise 3-1
  - Construct a **combined goal-tree** marked with (+ - ++ --)



Requirements Prioritization

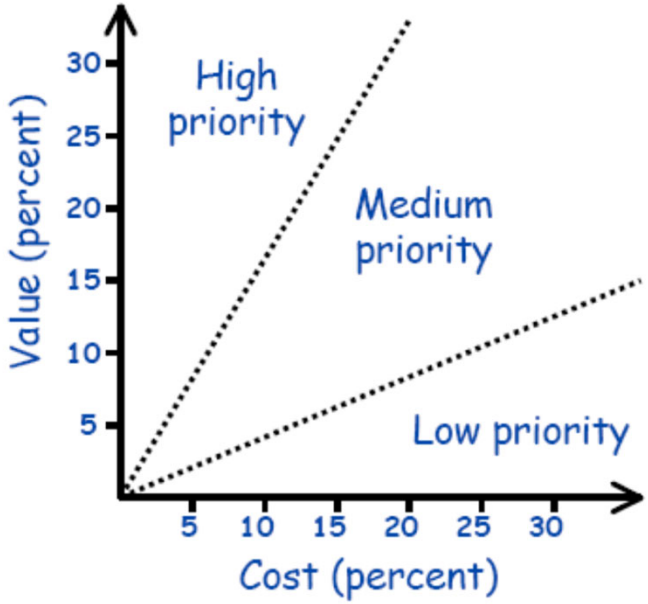
- **Need to select what to implement, after analyzing requirements**
  - Customers (usually) ask for too much
  - Balance time-to-market with amount of functionality
  - Decide which features go into the next release

- For each requirement/feature, ask:
  - How important is this to the customer?
  - How much will it cost to implement?
  - How risky will it be to attempt to build it?

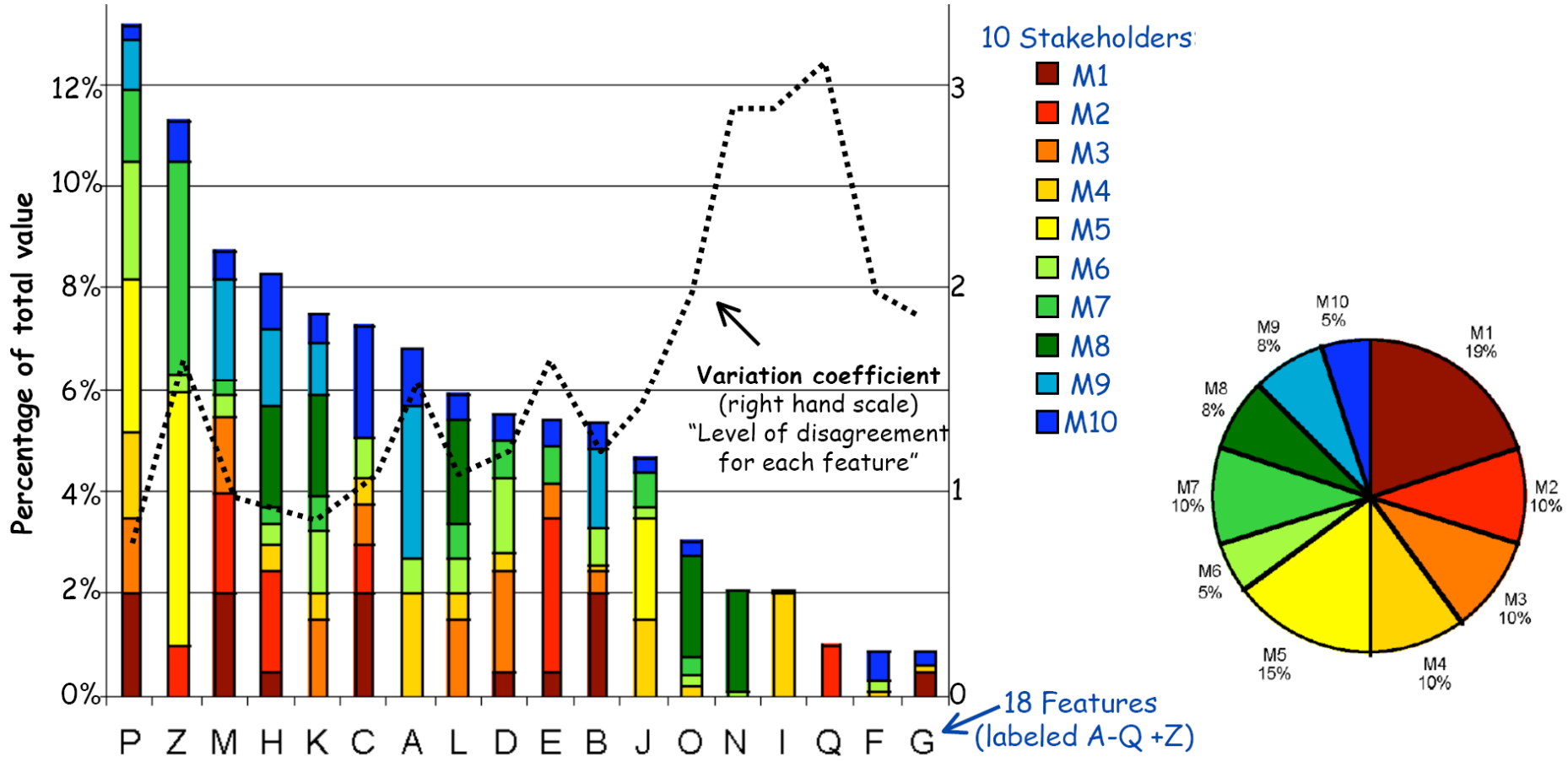
- **Perform Triage:**
  - Some requirements must be included
  - But, some requirements should definitely be excluded

A Cost-Value Approach

- **Calculate return on investment (ROI)**
  1. Assess each requirement's importance (value) to the project as a whole
  2. Assess the relative cost of each requirement
  3. Compute the cost-value trade-off:



Visualizing Value by Stakeholder



Estimating Cost & Value

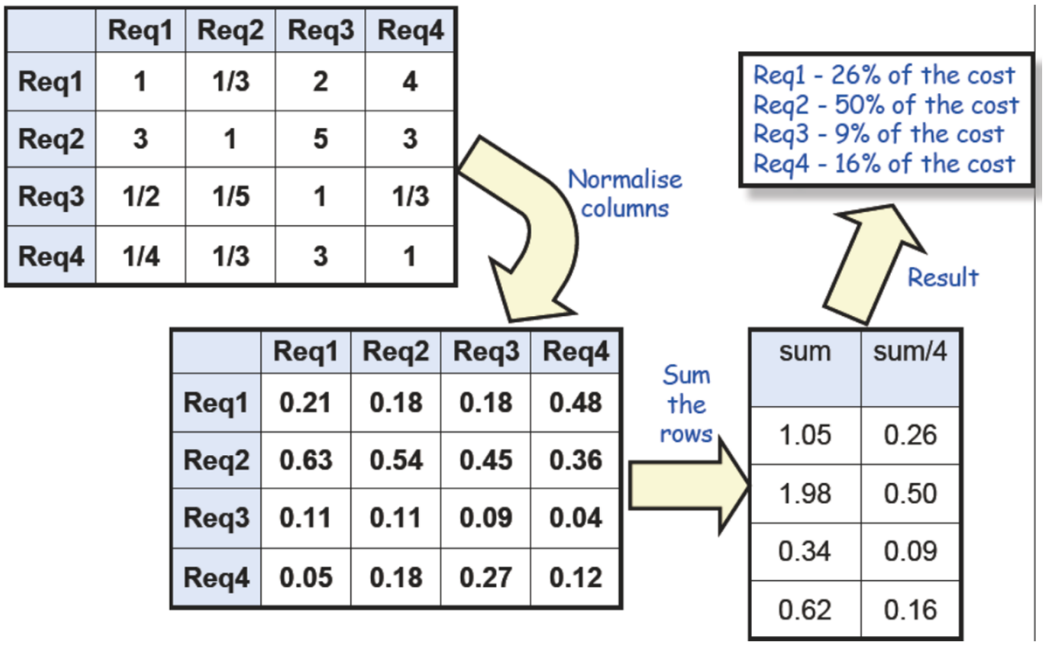
- Two approaches:
  - **Absolute scale** (e.g., dollar values)
    - Requires much domain experience
  - **Relative values** (e.g., less/more; a little, somewhat, very)
    - Much easier
    - Prioritization becomes a sorting problem
      - Bubble sort
      - Binary sort
      - MST (Minimum Spanning Tree)

Complications on Estimation

- **Hard to quantify differences quantitatively**
  - Easier to say “x is more important than y” than to estimate by how much
- **Not all requirements comparable**
  - E.g., different levels of abstraction
  - E.g., core functionality vs. customer enhancements
- **Requirements may not be independent**
  - No point selecting between X and Y if they are mutually dependent
- **Stakeholders may not be consistent**
  - E.g., if X > Y, and Y > Z, then presumably X > Z ?
- **Stakeholders might not agree**
  - Different cost/value assessments for different types of stakeholder

Analytic Hierarchy Process (AHP)

1. **Create n x n matrix for n requirements**
   - For element (x, y) in the matrix enter:
     - 1 : if x and y are of equal value
     - 3 : if x is slightly more preferred than y
     - 5 : if x is strongly more preferred than y
     - 7 : if x is very strongly more preferred than y
     - 9 : if x is extremely more preferred than y
   - For (y, x), enter the reciprocal.
2. **Estimate the eigenvalues:**
   - Use your own approach (strategy, heuristics)
   - E.g., "averaging over normalized columns"
     1. Calculate the sum of each column
     2. Divide each element in the matrix by the sum of its column
     3. Calculate the sum of each row
     4. Divide each row sum by the number of rows
   - This gives a value for each requirement:
     - Giving the estimated percentage of total value of the project



	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

	Req1	Req2	Req3	Req4
Req1	0.21	0.18	0.18	0.48
Req2	0.63	0.54	0.45	0.36
Req3	0.11	0.11	0.09	0.04
Req4	0.05	0.18	0.27	0.12

sum	sum/4
1.05	0.26
1.98	0.50
0.34	0.09
0.62	0.16

Considerations in Requirements Prioritization

- **Find factors that affects priority:**
  - How much does the customer want it?
  - How much cost to develop?
  - How much time to deliver?
  - How technologically difficult?
  - How organizationally difficult?
  - How much will the business benefit?
  - Relevant quality factors

- **Not all factors apply to all projects**
  - Each factor's importance varies from project to project
  - '*Relative*' importance is different to everyone

- **Include all major stakeholders:**
  - We need to prioritize the requirements in collaboration with the customers and developers
  - We must decide on a subset of requirements to be first implemented among various stakeholder interests
  - We need to remember that more influence is exercised by a particular group of stakeholders

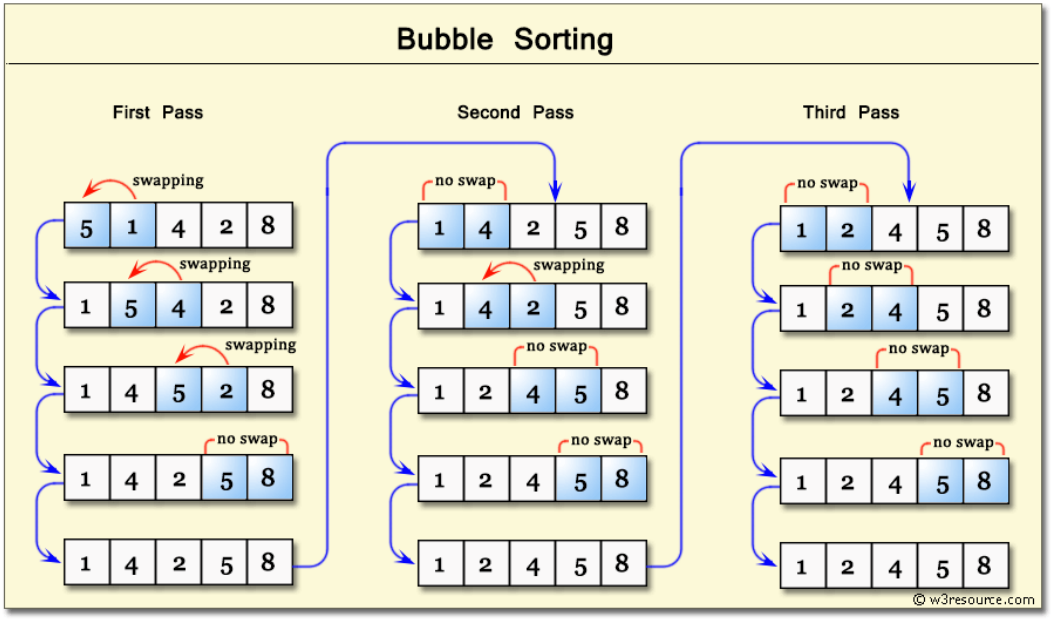
Requirements Prioritization Methods

- **Ranking**
  - When you rank requirements on an ordinal scale, you give each one a different numerical value based on its importance.
    - For example, the number 1 can mean that the requirement is the most important and the number n can be assigned to the least important requirement, n being the total number of requirements.

- **Numerical Assignment (Grouping)**
  - This method is based on grouping requirements into different priority groups with each group representing something stakeholders can relate to.
    - For example, requirements can be grouped into critical priority, moderate priority and optional priority.

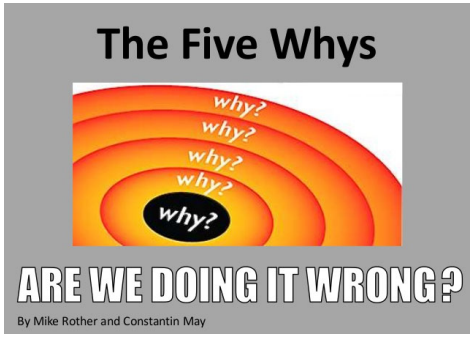
- **MoSCoW Technique**
  - Instead of numbers, this method uses four priority groups:
    - MUST (Mandatory)
    - SHOULD (High priority)
    - COULD (Preferred but not necessary)
    - WOULD (Can be postponed and suggested for future execution)

- **Bubble Sort Technique**
  - To prioritize requirements using bubble sort, you take two requirements and compare them with each other.
  - If you find out that one requirement should have greater priority over the other, you swap them accordingly. You then continue in this fashion until the very last requirement is properly sorted. The result is a list of requirements that are ranked.



- **Hundred Dollar Method**
  - This simple method is useful anywhere multiple stakeholders need to democratically vote on which requirements are the most important.
  - All stakeholders get a conceptual 100 dollars, which they can distribute among the requirements. As such, the stakeholder may choose to give all 100 dollars to a single requirement, or the person may distribute the points more evenly.
  - At the end, the total is counted, and the requirements are sorted based on the number of points received.

- **Five Whys**
  - With five whys, the analyst asks the stakeholder repeatedly (five times or less) why the requirement is necessary until the importance of the requirements is established.
  - The answers reveal whether the requirement is really necessary or can be cancelled/postponed once the priority is determined.

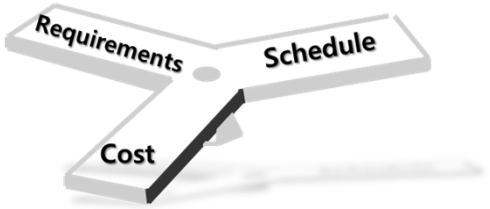


- **Overall AHP** (Analytical Hierarchy Process)
  - Step 1. List all features and use cases that must be prioritized
  - Step 2. Estimate the relative benefit if each feature is included
  - Step 3. Estimate the relative penalty if each feature is not included
  - Step 4. Estimate the relative cost of implementing
  - Step 5. Estimate the relative degree of technical or other risk
  - Step 6. Calculate a priority number for each feature
  - Step 7. Sort the list of features

Relative Weights	2.0	1.0	1.0		0.5		0.5	
Feature	Relative Benefit	Relative Penalty	Relative Cost	Cost %	Relative Risk	Risk %	**Total Value**	**Value %**
Print a material safety data sheet	2	4	1	2.7	1	3.0	**8**	**5.2**
Query status of a vendor order	5	3	2	5.4	1	3.0	**13**	**8.4**
Generate a Chemical Stockroom inventory	9	7	5	13.5	3	9.1	**25**	**16.1**
See history of a specific chemical container	5	5	3	8.1	2	6.1	**15**	**9.7**
Search vendor catalogs for a specific chemical	9	8	3	8.1	8	24.2	**26**	**16.8**
Maintain a list of hazardous	3	9	3	8.1	4	12.1	**15**	**9.7**

Requirements Triage

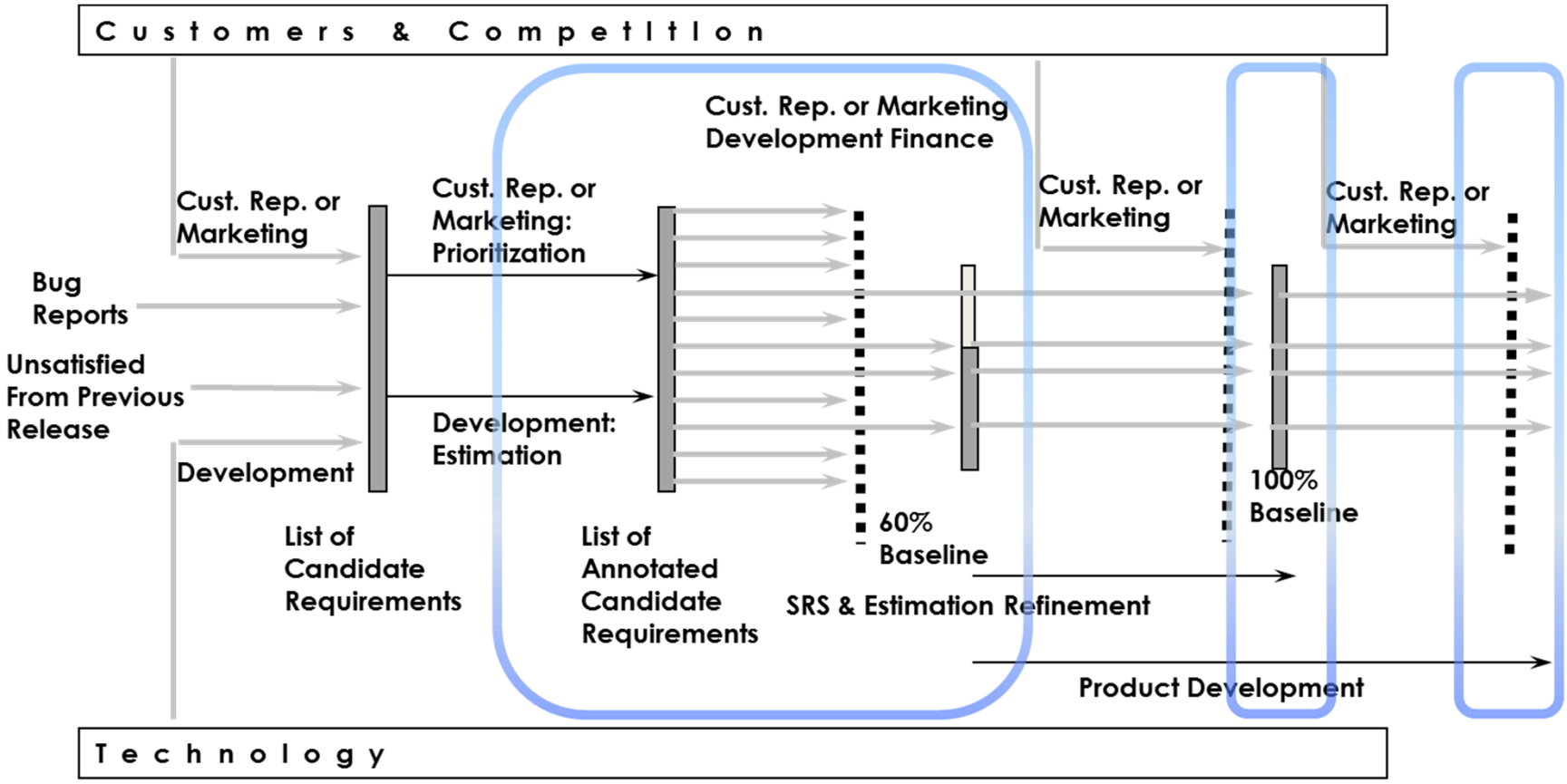
- **Selecting the "right" features to include in next release**
  - Arriving at an answer is not easy.
  - It's either Win-Win or Lose-Lose.



- **Requirements vs. Schedule/Cost Risk**
  - **Basic triage**
    - An Engineering View
    - Balancing between requirements and Cost/Risk/Schedule
  - **Advanced triage**
    - A Business View
    - Balancing between requirements and Cost, Risk, Schedule, Market, Sales, Revenues, Pricing, Profit, and ROI

- **Tips for requirements triage**
  - **Maintain requirements** in **lists**
  - **Annotate requirements** by at least relative priority and cost-to-satisfy
  - Involve representatives from **all key groups (stakeholders)**

Requirements Triage



Annotated Requirements Lists

- **Maintain sound advice to support all activities on requirements**
  - Enables you to answer questions such as:
    - How many requirements do you have?
    - How many high priority requirements do you have?
    - What percentage of the candidate requirements have you chosen to satisfy in your next release?
    - What percentage of the requirements deemed high priority by customer X are you satisfying?
    - If Sally quits, which requirements are affected?
    - What percentage of the requirements for this release have been validated?
    - And so on …
- **Find relevant importance to stakeholders**
  - What should we annotate?
    - Effect and cost
    - In which release?
    - Duration (optional)
    - Technical risk (optional)
- **Requirements should be in a database.**
  - Access, Excel, RequisitePro, CaliberRM, RTM, DOORS, etc.

Annotate Requirements Example

We've ANNOTATED the features.

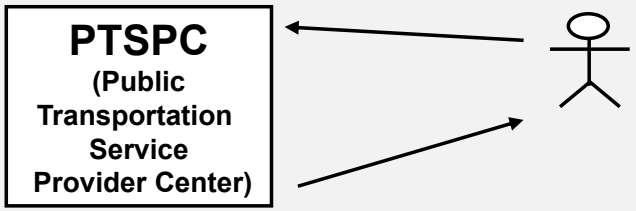
	ID	Requirement Text	Estim... Devel...	Tec... Risk	Priority	Rel...	Relates To	Comments	Child of	Level
...	961	No formal training shall be required to operate the RLM.	0.00	1	High	1.5			960	3
...	955	Any new releases or versions of the software shall be sold as new products. Users must p...	0.00	1	High	1.5			950	3
...	954	User software will not be modified or upgraded.	0.00	1	High	1.5			950	3
...	512	The RLM shall return to the refuel location or dump area to within 10 cm of the user-define...	10.00	6	Medi...	TBD			510	3
...	432	Pressing the screen in an area without a command shall make no sound nor shall it be int...	12.00	4	Medi...	TBD			430	3
...	415	The screen shall be capable of displaying alphanumeric data in blocked, uppercase char...	1.00	1	High	TBD			410	3
...	500	The RLM shall accept lawn and obstacle programming from the user. During programming, th...	35.00	7	High	TBD				1
...	321	The RLM shall initiate communications with the GPS through external interface EL-GPS	22.00	5	High	TBD			320	3
...	300	The RLM shall interface with two different external systems, The GPS and the Electronically S...	120.00	9	High	TBD				1
...	310	External interfaces include the receipt of location data from GPS and detection of obstacles...	22.00	9	High	TBD			300	2
...	511	The RLM shall not overcut or undercut the border and user defined obstacles by more tha...	10.00	4	High	TBD			510	3
...	510	The RLM shall cut the lawn only within the area defined by the user during the programming.	22.00	5	High	TBD			500	2
...	550	Border programming shall be required to be completed by the user prior to accepting the oth...	4.00	3	High	TBD			500	2
...	411	The Screen shall be 16.25 mm (high) by 105 mm (wide) and capable of displaying two row...	3.00	1	High	TBD			410	3
...	446	Serious errors (for example, blade fouling, Requirement 179) shall not have a button on the screen.	00	1	Medi...	TBD	179	Unclear	440	3
...	553	Programming border data shall be terminated by a user request, or when the RLM returns t...	4.00	3	High	TBD			550	3
...	554	After the termination, the RLM shall be ready to receive another command.	4.00	3	High	TBD			550	3
...	418	The screen shall be used to display information from the RLM to the user and accept dire...	5.00	1	High	TBD			410	3
...	561	User shall guide the RLM to the obstacle and indicated that the boundary of obstacle will ...	11.00	6	High	TBD			560	3
...	562	RLM shall record sufficient data (e.g. from GPS) to meet the accuracy requirements stated...	11.00	6	High	TBD	510, 5...		560	3
...	552	RLM shall record sufficient data (e.g. from GPS) to meet accuracy requirements stated in ...	4.00	3	High	TBD	510, 5...		550	3
...	551	User shall guide the RLM to the border of the lawn and indicate that the boundary will be ...	3.00	3	High	TBD			550	3
...	570	Programming refuel location shall be invoked by user during the initial state of programming ...	11.00	5	Medi...	TBD			500	2
...	571	User shall guide the RLM to the refuel location and indicate that the location of RLM is th...	13.00	5	Medi...	TBD			570	3
...	572	RLM shall record sufficient data to meet the accuracy requirements 510, 511, 512, and 5...	5.00	5	Medi...	TBD	510, 5...		570	3
...	573	Programming refuel location shall be terminated after RLM records its location.	6.00	4	Medi...	TBD		Unclear	570	3
...	574	After the termination, the RLM shall be ready to receive another command.	8.00	5	Medi...	TBD			570	3
...	447	In these cases, the RLM must be shut off and the error corrected by the user.	2.00	1	Medi...	TBD			440	3
...	591	User shall guide the RLM to the dump area and indicate that the boundary data of the du...	3.00	6	Medi...	TBD			590	3

Exercise 5: Basic Prioritization & Selection

- **Prioritize the collected stakeholder requirements and select a subset of requirements doable within 3 months**
  - Project Title: "**Custom Mass Transportation System**" in 1990s
  - Purpose: Increase the usage ratio of regional/suburban mass transportation system

- **Justify your selection quantitatively**

Order mass transportation through SMS, call center, internet (Notify departing location, destination, time of departure/arrival, etc.)



Find optimal travel route, fare, ETA, and other traffic information

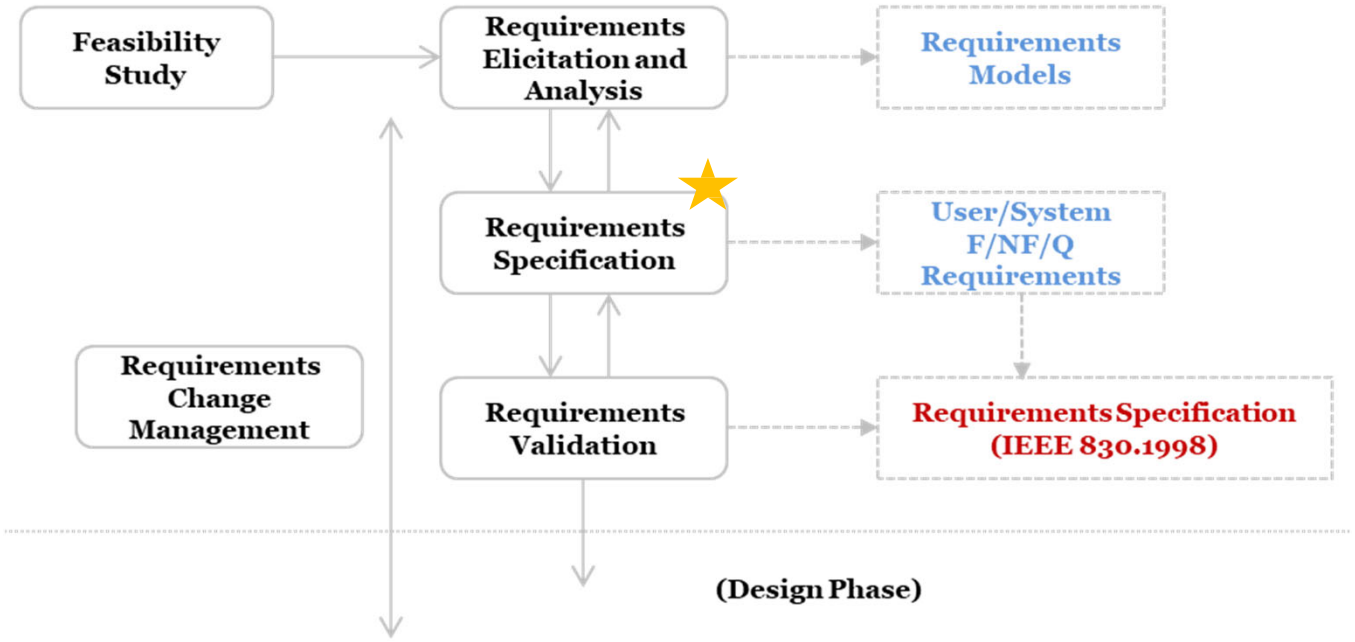
- 5 persons are available as workforce.
  - The project should be **under 15 man-month** if to complete within 3 months.
  - Assume that the Internet was booming.

- Use 4 columns of annotations.
  - Define each column (i.e., evaluation criteria - quality factor) clearly
  - Use 4 different prioritization methods, respectively
  - Keep in mind the purpose of your selection

Requirements	Value (1~10)	B	C	D	Effort (MM)	Total (Rank)	Selected (O / X)
Req. 1. The system should have features such as Register, Sign-in, Sign-out.					1		
Req. 2. The Driver should be able to view Passenger requests.					3		
Req. 3. The system should accept orders through the internet.					2		
Req. 4. The Customer should be able to designate the route in advance.					3		
Req. 5. The system should accept orders through SMS.					2		
Req. 6. The system should accept orders through the call center.					2		
Req. 7. Managers should be able to manage orders through the internet.					1		
Req. 8. Manager should be able to configure User profile through the call center.					2		
Req. 9. Data transfer between a taxi and traffic manager should be possible.					3		
Req. 10. Manager should be able to configure User profile through the internet.					1		

7. Requirements Specification

Requirements Engineering Process



Software Requirements Document

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

- 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*"

SRS Contents

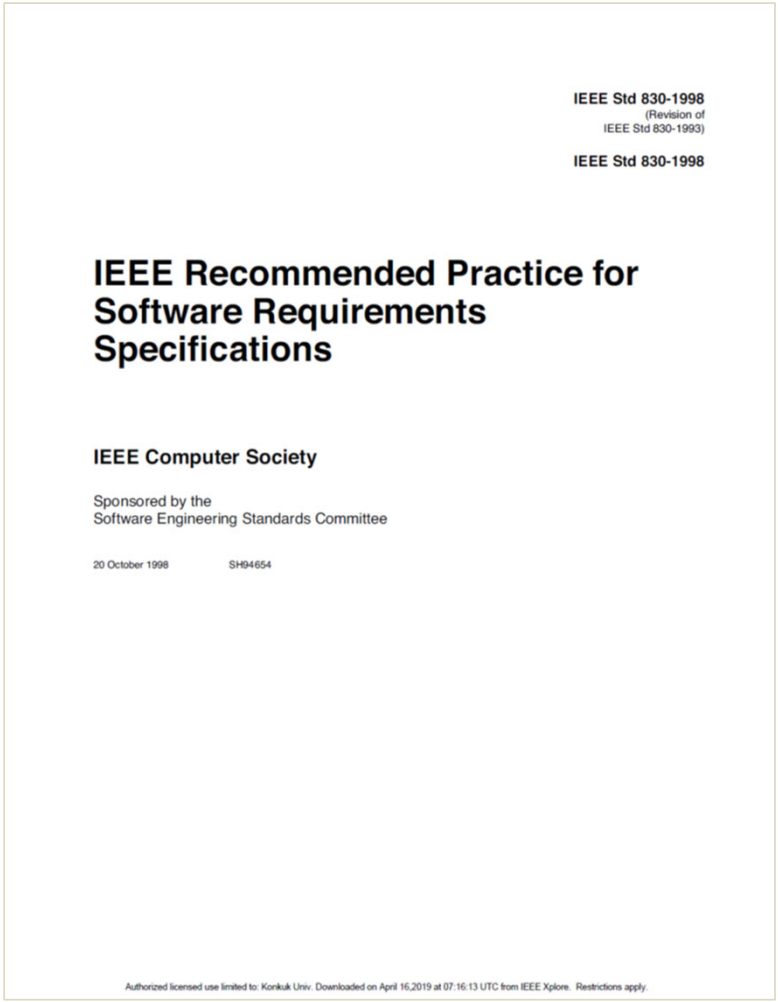
- **Software Requirements Specification** should address:
  - **Functionality**
    - What is the software supposed to do?
  - **External interfaces**
    - How does the software interact with people, the system's hardware, other hardware, and other software?
    - What assumptions can be made about 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?

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

Table of Contents

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

FR

QA

NFR

QA

NFR

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

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

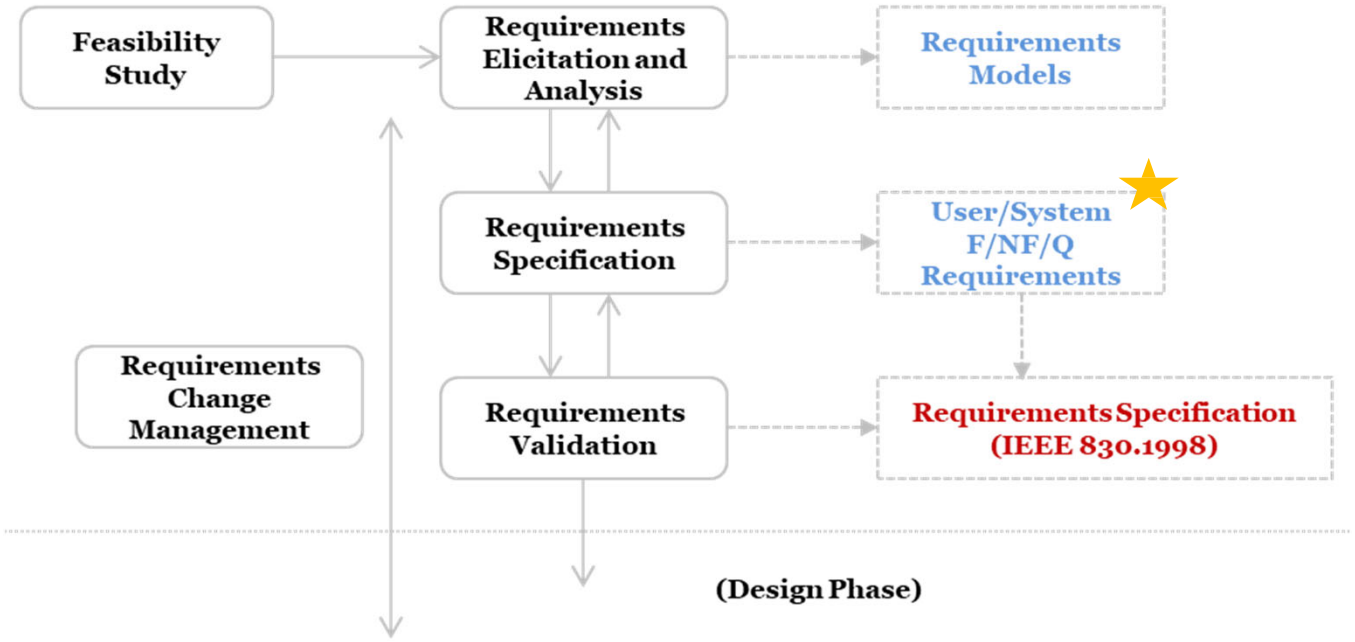
SRS Templates: IEEE STS 830-1998

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

8. Quality Attributes

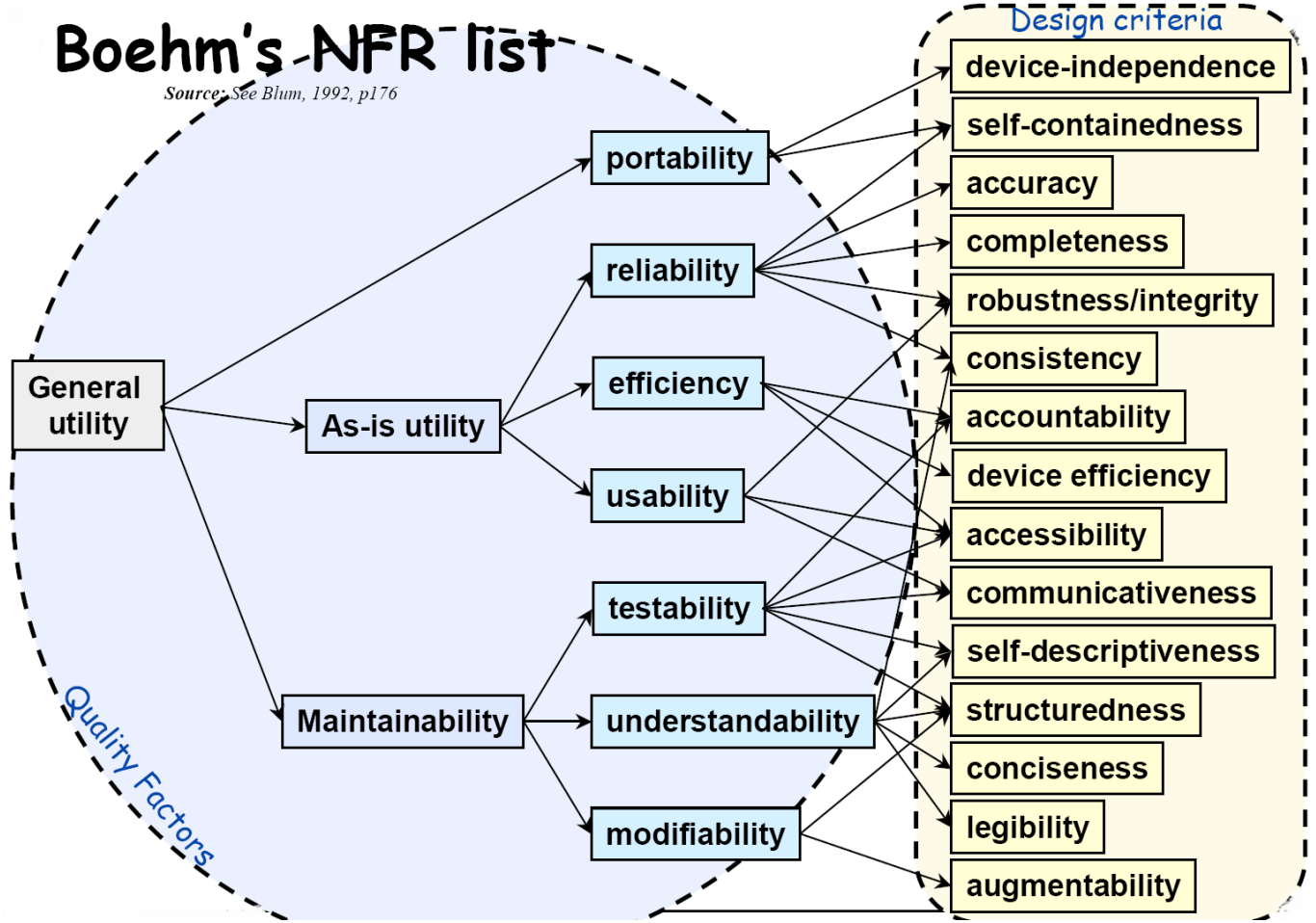
Requirements Engineering Process



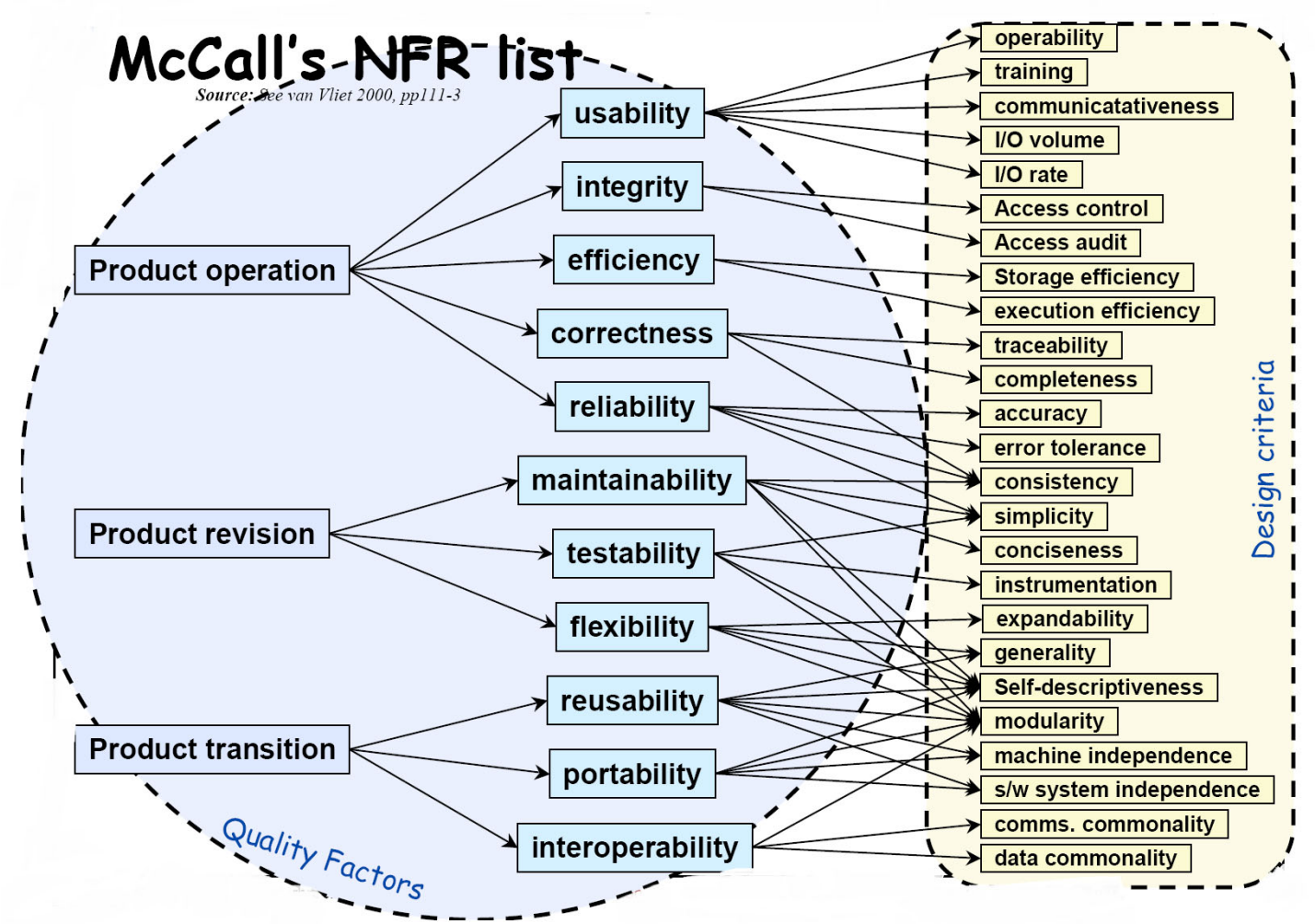
Non-Functional Requirements

- ISO/IEC 9126 / 25010
  - "*A Software requirement that **described not what the software will do, but how the software will do it**, for example, software performance requirements, software external interface requirements, design constraints, and software quality attributes.*"
- Sommerville
  - "***Constraints on the services or functions** offered by the system such as timing constraints, constraints on the development process, standards, etc.*"
- Wikipedia
  - "*A requirement that **specifies criteria that can be used to judge the operation of a system**, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is **detailed in the system architecture**, because they are usually architecturally significant requirements.*"

Boehm's NFR


Boehm's NFR list
Source: See Blum, 1992, p176
General utility
As-is utility
Maintainability
portability
reliability
efficiency
usability
testability
understandability
modifiability
Quality Factors
Design criteria
device-independence
self-containedness
accuracy
completeness
robustness/integrity
consistency
accountability
device efficiency
accessibility
communicativeness
self-descriptiveness
structuredness
conciseness
legibility
augmentability

McCall's NFR

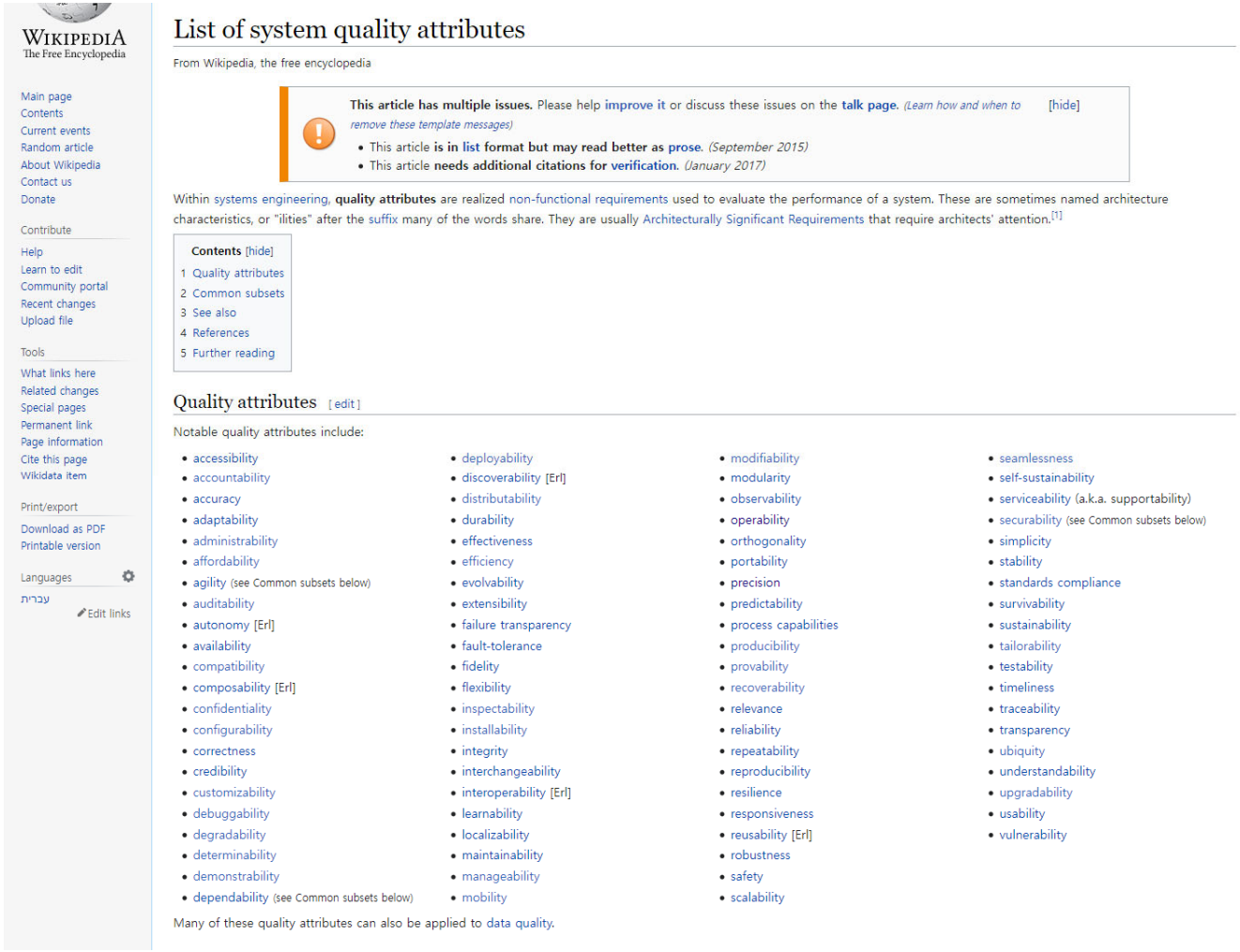

McCall's NFR list
Source: See van Vliet 2000, pp111-3
Product operation
Product revision
Product transition
usability
integrity
efficiency
correctness
reliability
maintainability
testability
flexibility
reusability
portability
interoperability
Quality Factors
operability
training
communicatativeness
I/O volume
I/O rate
Access control
Access audit
Storage efficiency
execution efficiency
traceability
completeness
accuracy
error tolerance
consistency
simplicity
conciseness
instrumentation
expandability
generality
Self-descriptiveness
modularity
machine independence
s/w system independence
comms. commonality
data commonality
Design criteria

Quality Attributes

- Measurable or testable properties of a system
  - Used to indicate how well the system satisfies **the needs of its stakeholders**
    - Availability, configurability, modifiability, performance, reliability, reusability, security, portability, maintainability, efficiency, usability
  - Emergent properties : not a measure of software in isolation
    - Measures the relationship between software and its application domain
    - Cannot measure this until you place the software into its environment
      - Quality will be different in different environments

- Software quality is all about **fitness to purpose** of **stakeholders**.
  - ***"Does it do what is needed?"***
  - ***"Does it do it in the way that its users need it to?"***
  - ***"Does it do it reliably enough? fast enough? safely enough? securely enough?"***
  - ***"Will it be affordable? will it be ready when its users need it?"***
  - ***"Can it be changed as the needs change?"***

Quality Attributes : Taxonomies



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
עברית
Edit links

List of system quality attributes

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. *(Learn how and when to remove these template messages)* [hide]

- This article **is in list format but may read better as prose**. *(September 2015)*
- This article **needs additional citations for verification**. *(January 2017)*

Within systems engineering, **quality attributes** are realized non-functional requirements used to evaluate the performance of a system. These are sometimes named architecture characteristics, or "ilities" after the suffix many of the words share. They are usually Architecturally Significant Requirements that require architects' attention.[1]

Contents [hide]

1 Quality attributes
2 Common subsets
3 See also
4 References
5 Further reading

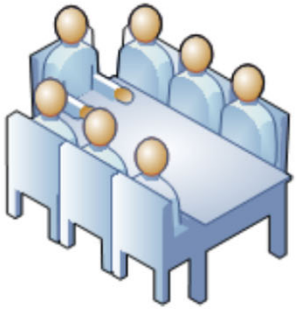
Quality attributes [edit]

Notable quality attributes include:

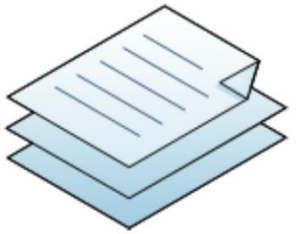
- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composability [Erl]
- confidentiality
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

Many of these quality attributes can also be applied to data quality.

Quality Attributes from Stakeholders



Lead To



Stakeholder Needs

Quality Attribute Requirements

"Increase market share" ----------> Modifiability, Usability

"Maintain a quality reputation" ----------> Performance, Usability, Availability

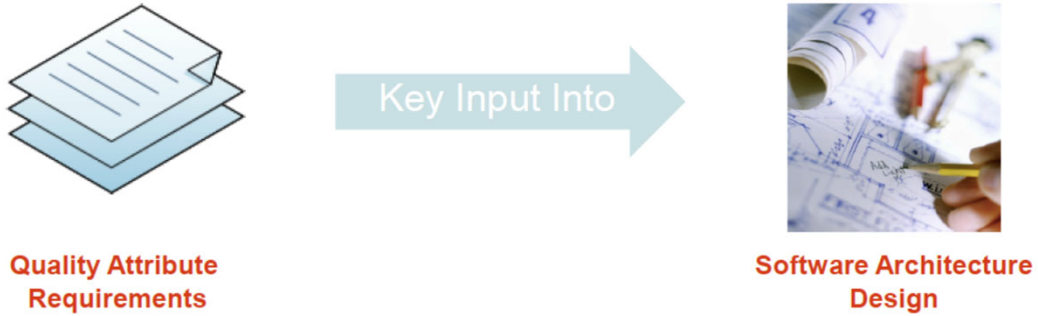
"Introduce new capabilities seamlessly" ----------> Performance, Availability, Modifiability

"Provide a programmer-friendly framework" ----------> Modifiability

"Integrate with other systems easily" ----------> Interoperability, Portability, Modifiability

Quality Attributes to Software Architecture

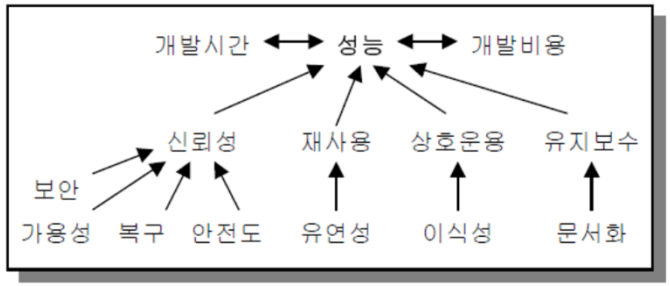
- The degree to which a system satisfies quality attribute requirements is directly dependent on architectural structure.



- **Architects** need to have a solid understanding of the quality attribute requirements for a system, when they are designing the system's software architecture.

Problematic Features of Quality Attribute

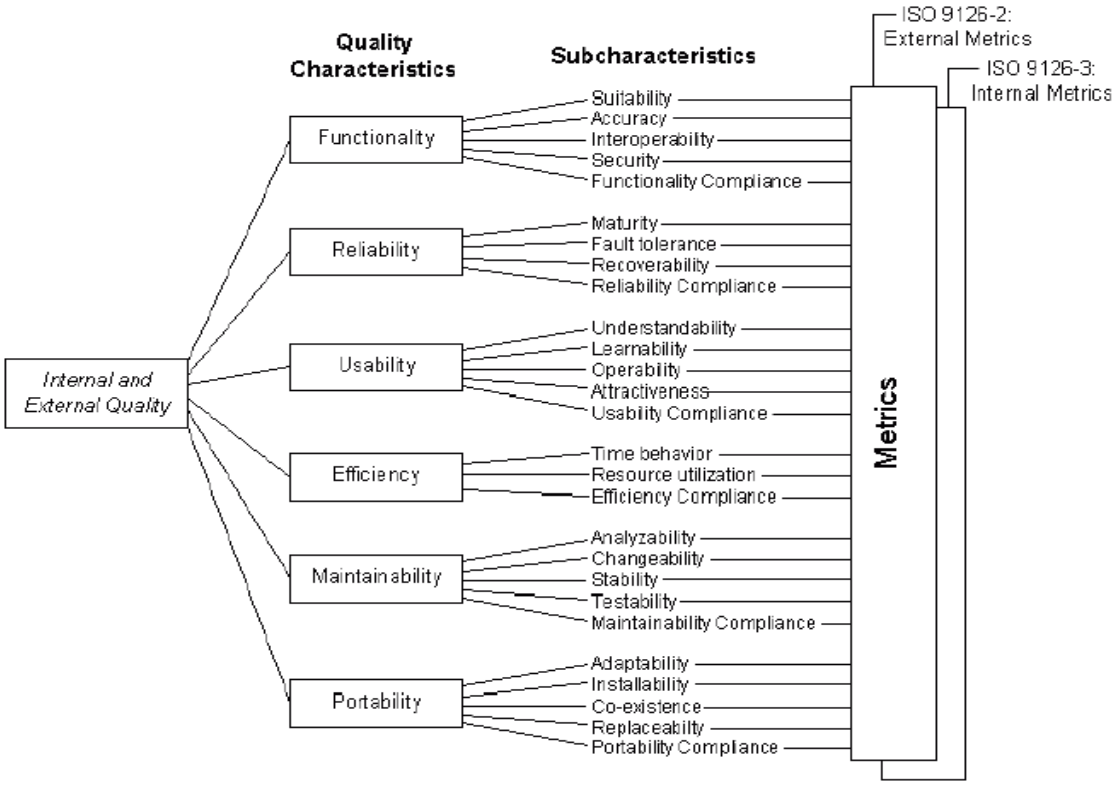
- **Non-Operational requirements**
  - "The system must be easy to use."
  - "The system must have high performance."
  - "The system must be portable."
- **Debating the quality attribute to which a system behavior belongs**
  - "The system must process 10,000 messages per second."
- **Vocabulary variations**
  - Everyone knows what "high performance" means, but different each others.
- **Various inter-dependency among quality attributes**
  - Trade-off
  - No 100% satisfied

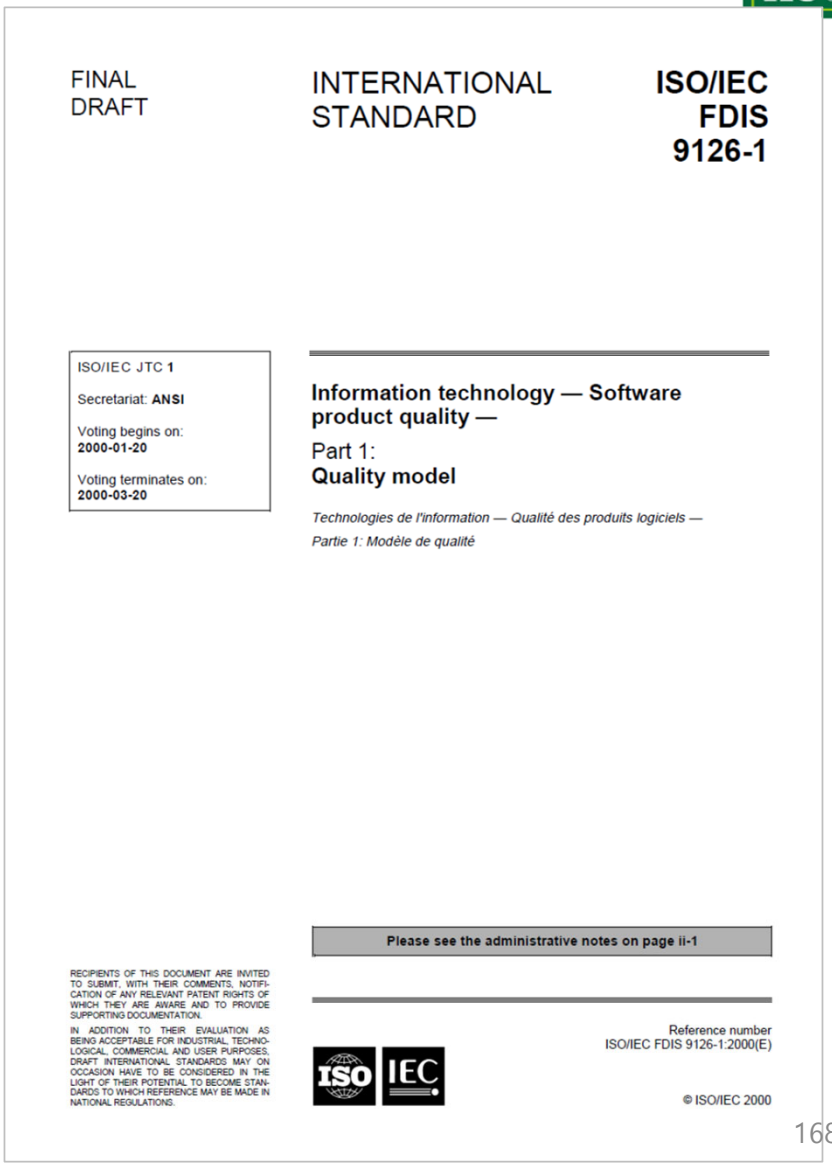


Quality Requirements: Examples

- 응용 프로그램을 위한 프로세서 용량과 RAM 중에서 20%는 최대 부하시점에서도 사용되지 않아야 한다
- 감사접속 권한을 가진 자만이 고객 거래자료를 볼 수 있다
- 사용자가 파일을 저장하기 전 편집기에 에러가 발생하면 편집 중이던 모든 변경내용을 에러발생 5분 전 까지로 복구한다
- 메뉴 파일의 모든 기능은Ctrl+다른 키를 사용하는 단축키가 정의되어야 한다
- 함수 호출은 3 단계 이상 중첩되지 않는다
- 모듈의 Cyclomatic Complexity는 20을 넘지 않는다
- 온도관리 주기는 0.8초 이내에서 수행한다
- 모든 웹 페이지는 10Mbps LAN 접속에서 5초 이내로 다운로드 한다

ISO/IEC 9126





FINAL DRAFT

INTERNATIONAL STANDARD

ISO/IEC FDIS 9126-1

ISO/IEC JTC 1

Secretariat: **ANSI**

Voting begins on: **2000-01-20**

Voting terminates on: **2000-03-20**

Information technology — Software product quality —

Part 1:
Quality model

Technologies de l'information — Qualité des produits logiciels —
Partie 1: Modèle de qualité

Please see the administrative notes on page ii-1

RECIPIENTS OF THIS DOCUMENT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

Reference number
ISO/IEC FDIS 9126-1:2000(E)

© ISO/IEC 2000

ISO 9126-1 : Information Technology – Software Product Quality – Part 1: Quality Model

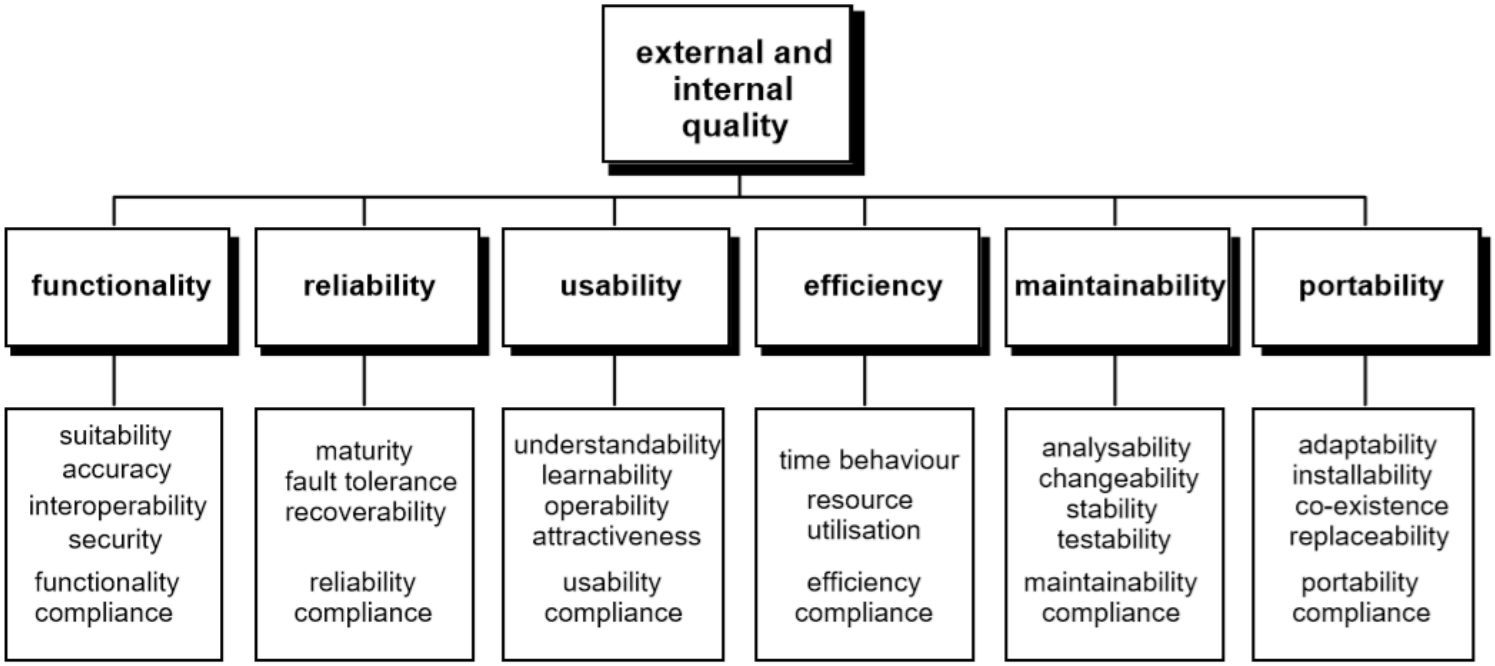


Figure 4 – Quality model for external and internal quality

ISO/IEC 25010

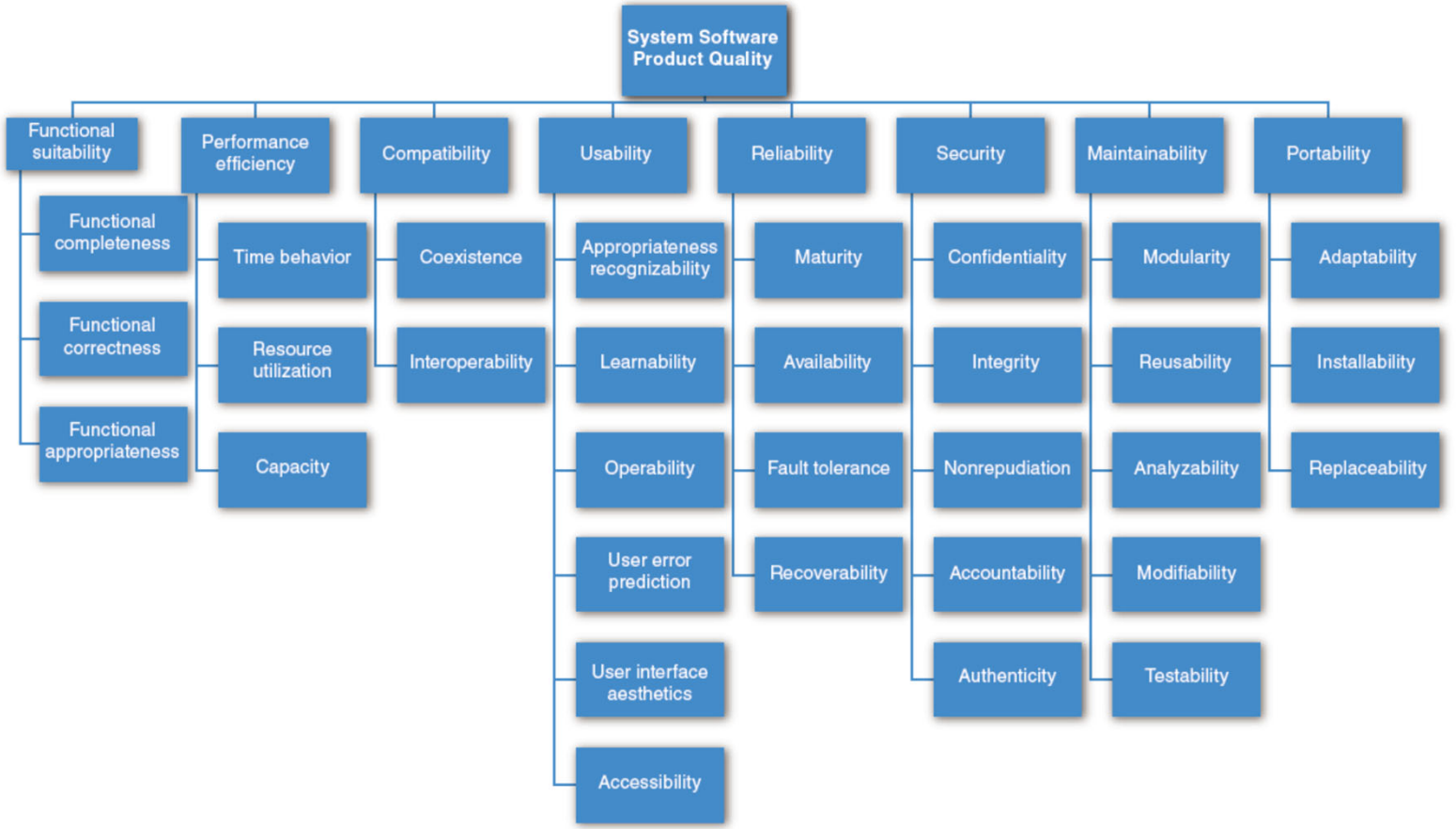
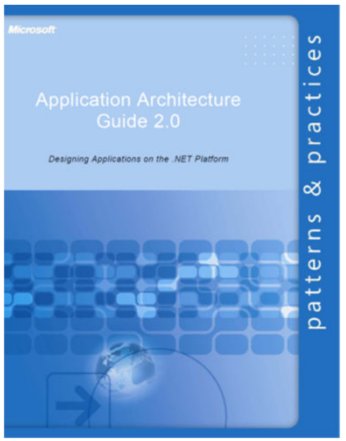


FIGURE 12.1 The ISO/IEC FCD 25010 product quality standard

Microsoft Application Architecture Guide

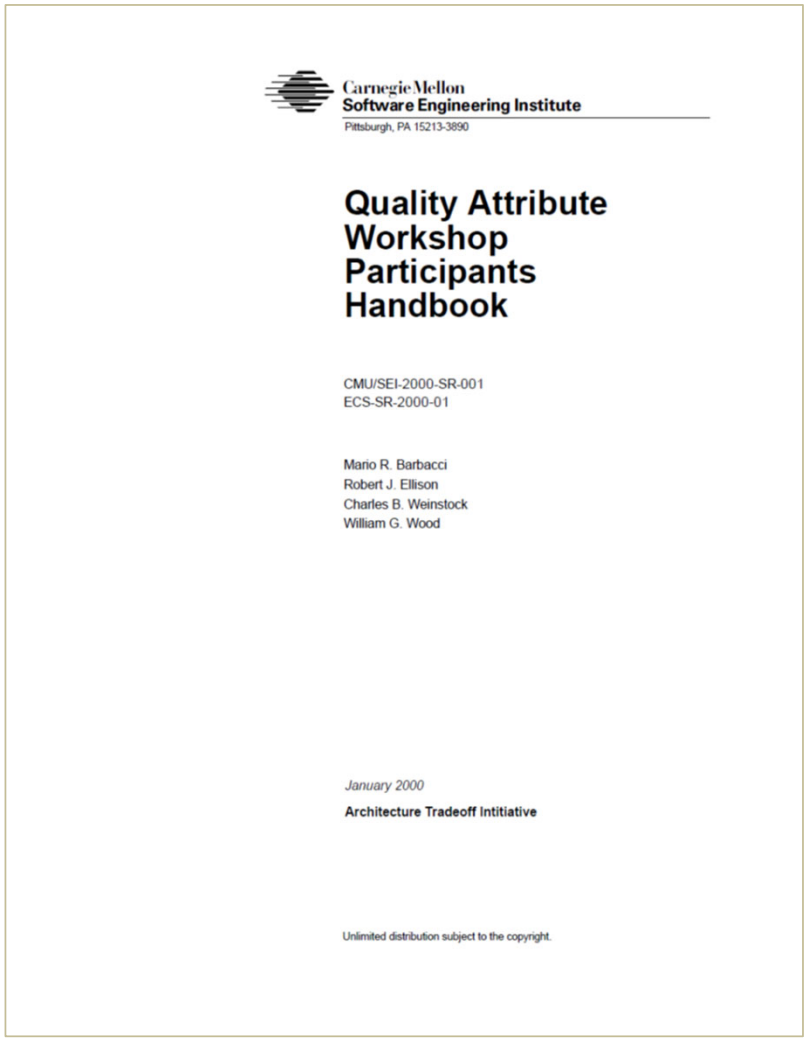
- **Quality attributes** are the overall factors that affect run- time behavior, system design, and user experience.
  - They represent areas of concern that have the potential for application wide impact across layers and tiers
  - When designing applications to meet any of the quality attributes requirements, it is necessary to consider the potential impact on other

- 4 Categories of Quality Attributes
  - **Design Qualities** : Conceptual Integrity, Maintainability, Reusability
  - **Run-time Qualities** : Availability, Interoperability, Manageability, Performance, Reliability, Scalability, Security
  - **System Qualities** : Supportability, Testability
  - **User Qualities** : Usability



CMU SEI Quality Attributes

- Dependability
- Security
- Modifiability
- Interoperability
- Performance



Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Quality Attribute Workshop Participants Handbook

CMU/SEI-2000-SR-001
ECS-SR-2000-01

Mario R. Barbacci
Robert J. Ellison
Charles B. Weinstock
William G. Wood

January 2000

Architecture Tradeoff Intitiative

Unlimited distribution subject to the copyright.

Wikipedia – Quality Attributes

Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
עברית
Edit links

List of system quality attributes

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page.** *(Learn how and when to remove these template messages)* [hide]

- This article **is in list format but may read better as prose.** *(September 2015)*
- This article **needs additional citations for verification.** *(January 2017)*

Within systems engineering, **quality attributes** are realized non-functional requirements used to evaluate the performance of a system. These are sometimes named architecture characteristics, or "ilities" after the suffix many of the words share. They are usually Architecturally Significant Requirements that require architects' attention.[1]

Contents [hide]

1 Quality attributes
2 Common subsets
3 See also
4 References
5 Further reading

Quality attributes [edit]

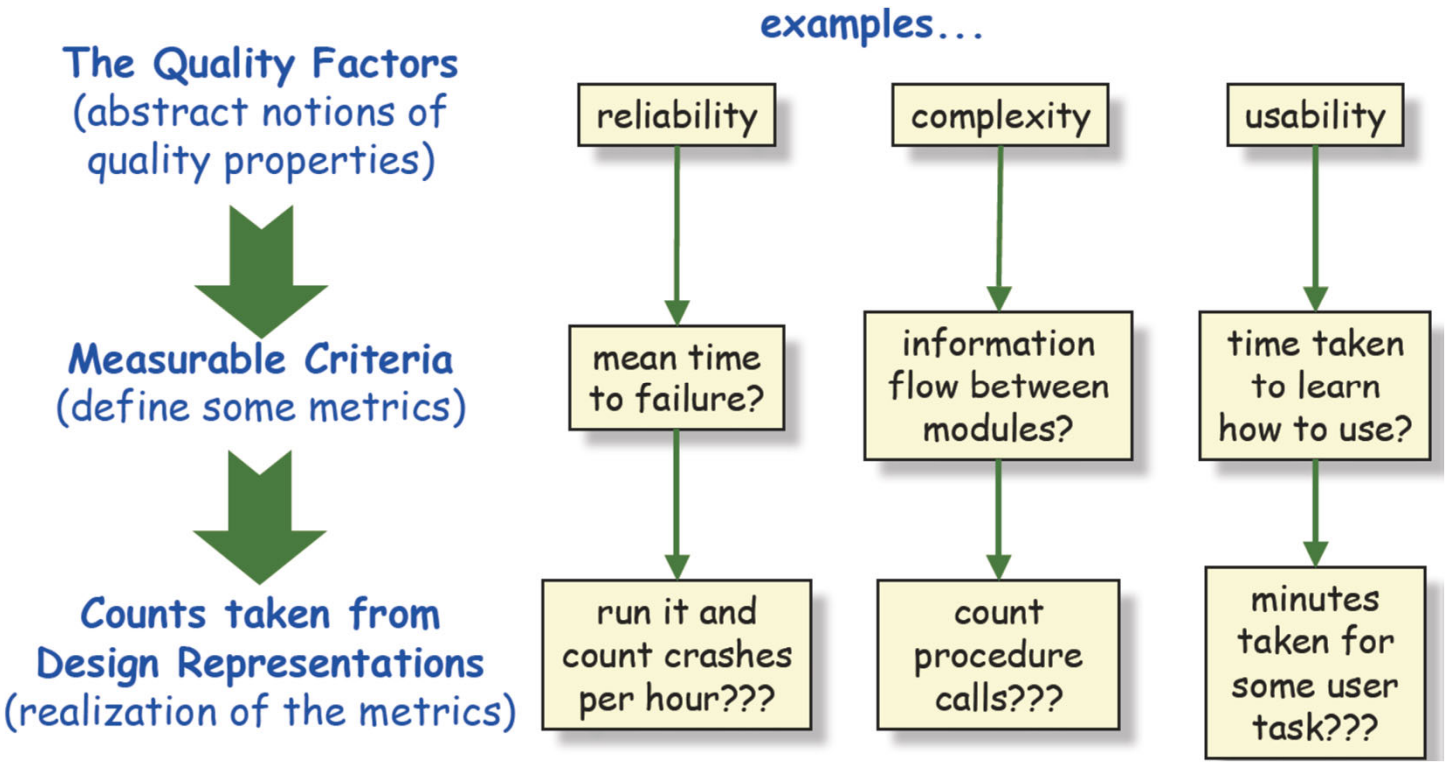
Notable quality attributes include:

- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composability [Erl]
- confidentiality
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

Many of these quality attributes can also be applied to data quality.

Making All Requirements Measurable

- How to turn vague ideas about quality into **measurables** or **verifiable**
  - **Quality M&M** (Metric & Measure)



Quality Metric & Measure

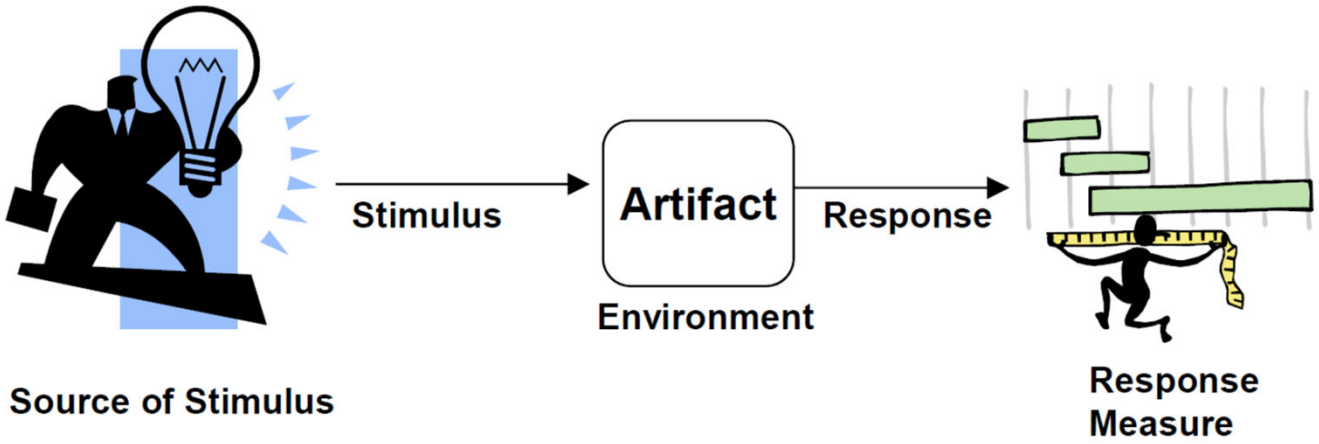
Quality Factor	Metric & Measures
Speed	- Processed transactions/second - User/event response time - Screen refresh time
Size	- Mbytes - Number of ROM chips
Ease of Use	- Training time - Number of help frames
Reliability	- Mean time to failure - Probability of unavailability - Rate of failure occurrence - Availability
Robustness	- Time to restart after failure - Percentage of events causing failure - Probability of data corruption on failure
Portability	- Percentage of target dependent statements - Number of target systems
Maintainability	- Volume of data recorded in operation - Number of failures estimated - Correction time / software size

Quality Attribute Tree : Examples

No.	Category	Response Measure	품질속성 요구사항
1	Performance	시스템 동작 Hz 수	1 Core System에서 4 Core System을 지원하는 system으로 변경될 경우, 정상적인 Operation을 수행하는 데 있어서 소모되는 전력이 1 Core System의 경우와 비교하여 35%의 Hz수를 출력하여야 한다.
2		Communication 횟수, Communication 데이터	정상적인 Operation을 수행하는 데 있어서 Core 간 Communication 으로 인해 발생되는 Overhead 증가분은 10% 이내이어야 한다.
3		사용된 Memory 용량	정상적인 Operation을 수행하는 데 있어서 memory 사용량 증가 정도는 70 % 이내이어야 한다.
4		수행 중 각 Core들의 Idle Time	정상적인 Operation을 수행하는 데 있어서 전체 system의 운영 시간 중 각 Core들이 Idle 상태에 머무르는 시간은 15% 이내 이어야 한다.
5		Data Frames/Second	정상적인 Operation하에서 Video Data Decoding 성능은 기존 1 Core System의 3배 정도인 2.5 Data Frames/Second를 만족시켜야 한다.
6		화면의 가로 세로 픽셀 수	정상적인 Operation하에서 Video Decoder가 지원 가능한 출력 화면의 크기는 기존 1 Core System의 경우와 마찬가지로 1280X720 픽셀까지이다 .
7		Bit rate	정상적인 Operation하에서 Video Decoder가 출력하는 화면의 화질을 측정하는 Bit rate는 기존 1 Core System의 경우와 마찬가지로 20Mbps를 지원 가능하여야 한다.
8	Modifiability	수정 컴포넌트 비율	4 Core System에서 향후 그 이상의 개수로 Core 숫자가 늘어날 경우, 4 Core System 기반의 Architecture 상에서 변경되는 Component와 Connector의 비율은 Parallel Node의 숫자에 해당되는 Instance 개수 증가를 제외하고 50% 미만이어야 한다.
9	Functionality	시스템 기능 가용률 :전체 기능 개수 대비 가용 기능 비율	1 Core System에서 4 Core System을 지원하는 system으로 변경될 경우, 정상적인 Operation하에서 Video Decoder가 제공하던 기능 중 가용한 기능은 100%를 만족시켜야 한다.
10		해당사항 없음	Video Decoder의 input data stream 중에 error가 있는 input frame이 입력될 경우, Video Decoder는 해당 frame의 decoding을 수행하지 않고 pass한 후 다음 frame을 읽어 들여야 한다.
11	Portability	시스템 기능 가용률 :전체 기능 개수 대비 가용 기능 비율	Cache memory size 가 32K인 Device에서 16K인 Device로 변경될 경우, 정상적인 Operation하에서 Video Decoder가 제공하던 기능 중 가용한 기능은 100%를 만족시켜야 한다.

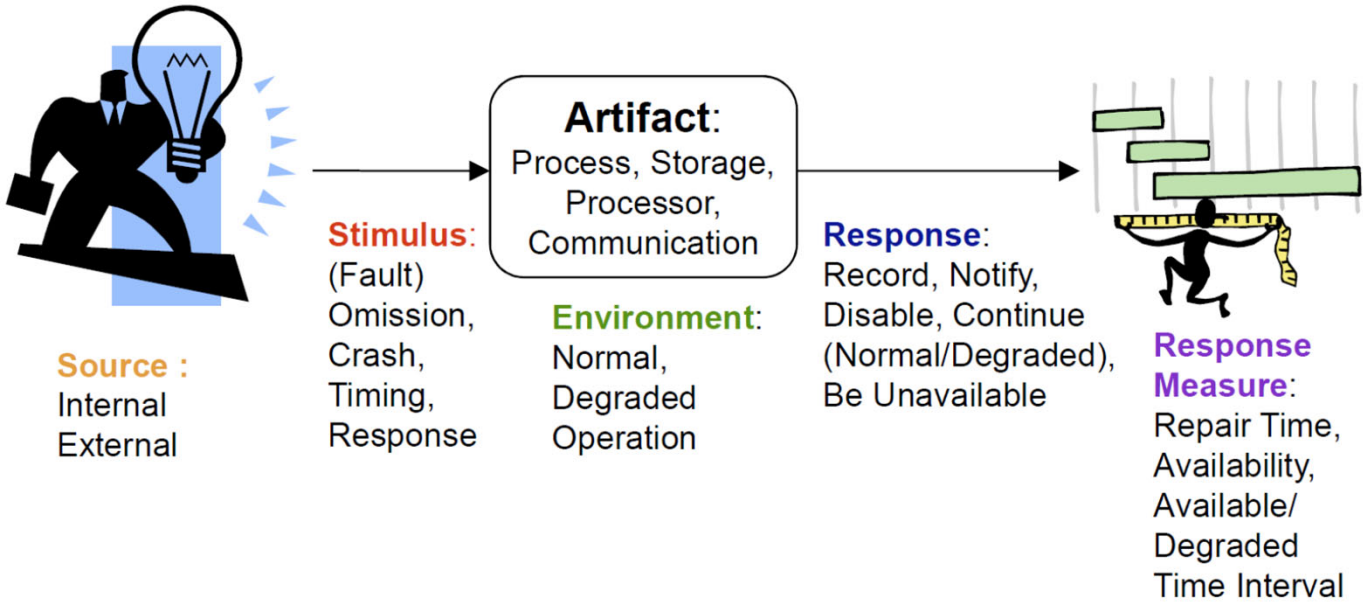
Quality Attribute Scenarios

- **QAS (Quality Attribute Scenario)** is an effective way of identifying and specifying quality-attribute-specific requirements.
  - Specific to the particular system under considerations
  - Instantiated from the attribute characterizations of general scenarios



A QAS Example for Availability

- "An **unanticipated external message** is received by a process during **normal operation**. **The process informs the operator of the receipt of the message and continues to operated with no downtime.**"

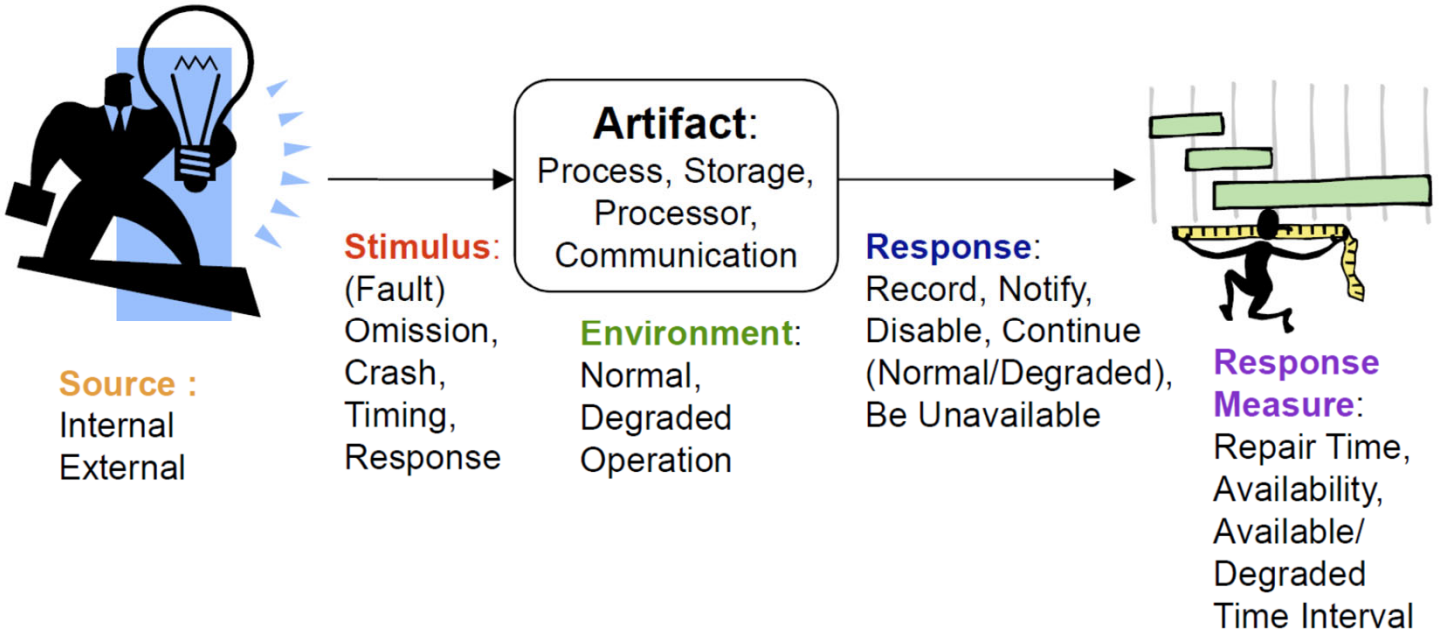


The QAS Template

Requirement-ID	*QA_###*
Category	관련된 ***Quality Attribute***가 무엇인지 기술함 ***(*** 예***: Performance, Reliability, Security*** 등***)***
Source	***Stimulus***를 발생시키는 주체가 무엇인지 기술함
Stimulus	시스템에 입력되는 내외부 자극이 무엇인지 기술함
Artifacts	***Stimulus***의 영향을 받는 시스템의 내부 모듈, 컴포넌트, 혹은 시스템 전체
Environment	해당 ***Stimulus*** 발생시 시스템의 환경***(***운영모드일 수도 있고, 그 외 다양한 상황 가능***)***
Response	기술된 ***Environment***에서 ***Artifact***이 ***Stimulus***를 받아들인 후 취하는 ***Action***이 무엇인지 설명함
Response Measure	위의 ***Response***의 정도를 측정하는 단위가 무엇인지 기술함 ***(***예***:*** 초당 데이터 처리량, 반응시간, 시간일 경우 ***hour, minute, second*** 여부 등***)***
Priority	***Quality Attribute Tree*** 상에서의 우선순위
Description	위에 기술된 ***Source***부터 ***Response Measure***까지의 내용을 하나의 문장으로 요약해서 기술함

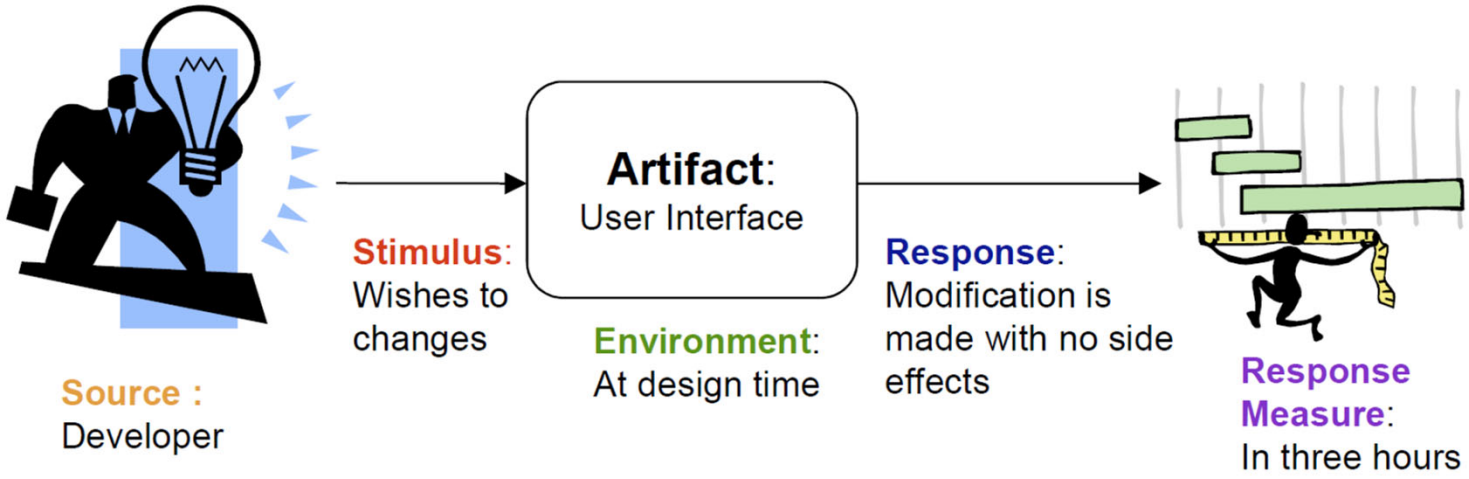
QAS Example – Availability (Reliability)

- "An **unanticipated external message** is received by a process during **normal operation**. **The process informs the operator of the receipt of the message and continues to operated with no downtime.**"



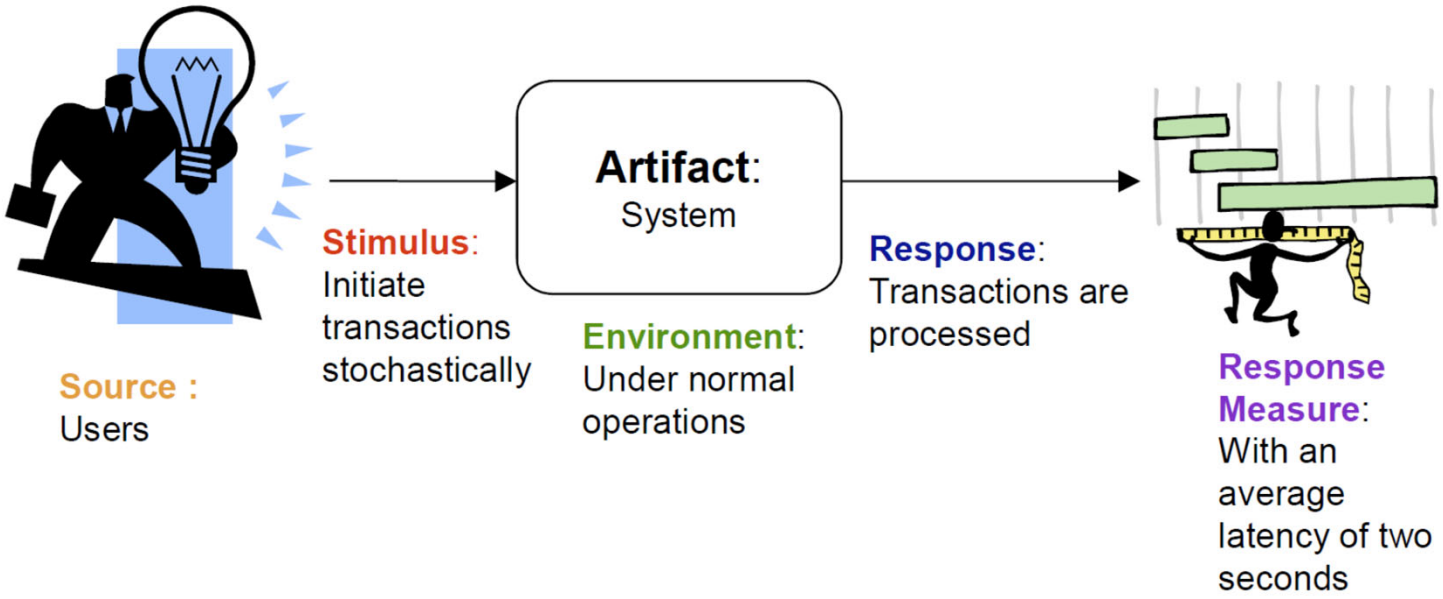
QAS Example – Modifiability (Adaptability)

- "A developer wishes to change the user interface to make a screen's background color blue. This change will be made to the code at design time. It will take less than three hours to make and test the change and no side effect changes will occur in the behavior."



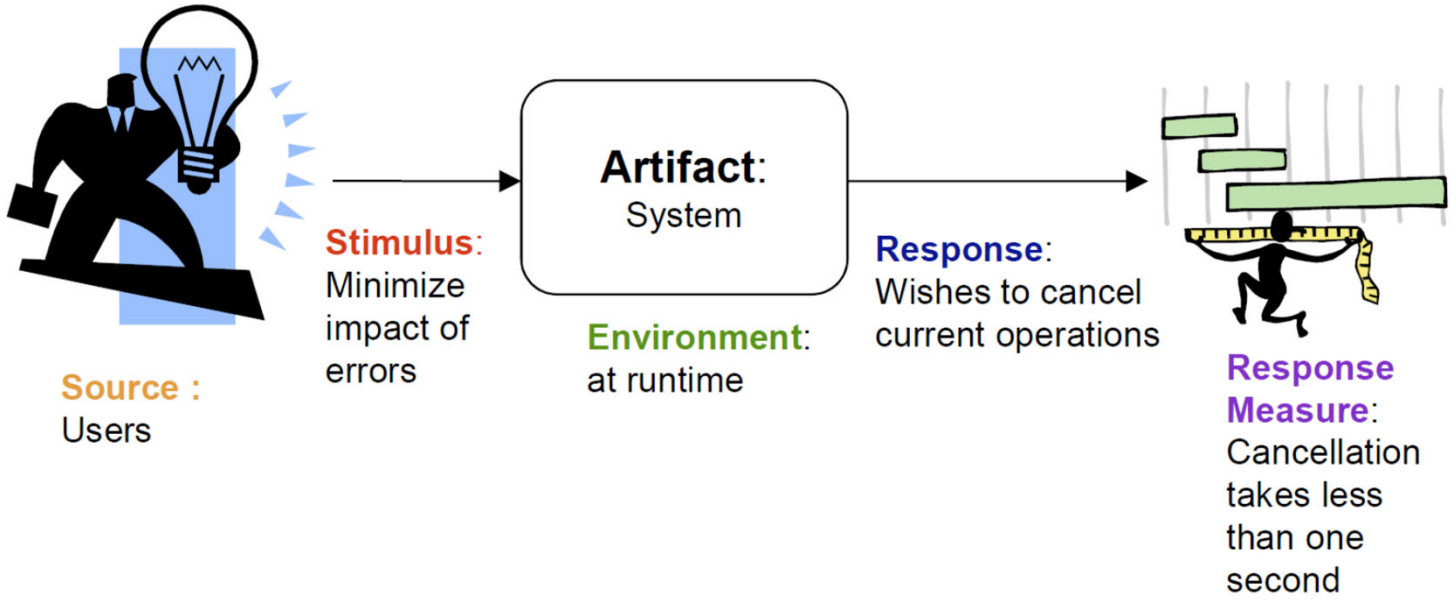
QAS Example – Performance

- "**Users initiate 1,000 transactions per minute stochastically under normal operations**, and **these transactions are processed with an average latency of two seconds**."



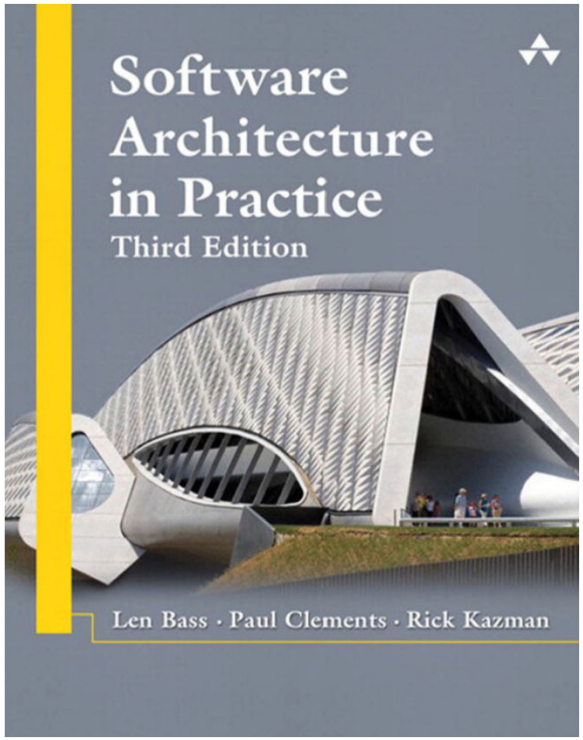
QAS Example – Usability

- "A user wanting to minimize the impact of an error, wishes to cancel a system operation at runtime; cancellation takes place in less than one second."



Tactics

- **Techniques** that architects have been using for years to manage quality attribute response goals.
  - Design decisions that influence the control of a quality attribute response



Quality Attribute - Availability

- Ability of a system to mask or repair faults such that the cumulative service outage period does not exceed a required value over a specified time interval

- The availability of a system can be calculated as the probability that it will provide the specified services within required bounds over a specified time interval.

- When referring to hardware, there is a well-known expression used to derive steady-state availability:
  - MTBF : the mean time between failures
  - MTTR : the mean time to repair

Quality Attribute Scenario – Availability

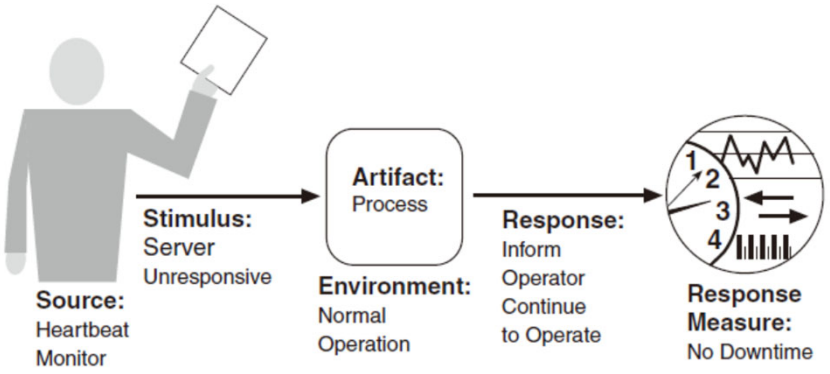


FIGURE 5.3 Sample concrete availability scenario

Portion of Scenario	Possible Values
Source	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	Fault: omission, crash, incorrect timing, incorrect response
Artifact	Processors, communication channels, persistent storage, processes
Environment	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
Response	Prevent the fault from becoming a failure Detect the fault: ▪ Log the fault ▪ Notify appropriate entities (people or systems) Recover from the fault: ▪ Disable source of events causing the fault ▪ Be temporarily unavailable while repair is being effected ▪ Fix or mask the fault/failure or contain the damage it causes ▪ Operate in a degraded mode while repair is being effected
Response Measure	Time or time interval when the system must be available Availability percentage (e.g., 99.999%) Time to detect the fault Time to repair the fault Time or time interval in which system can be in degraded mode Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing

Tactics for Availability

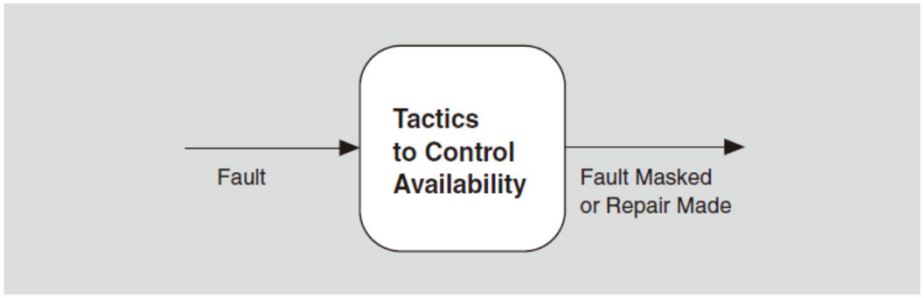
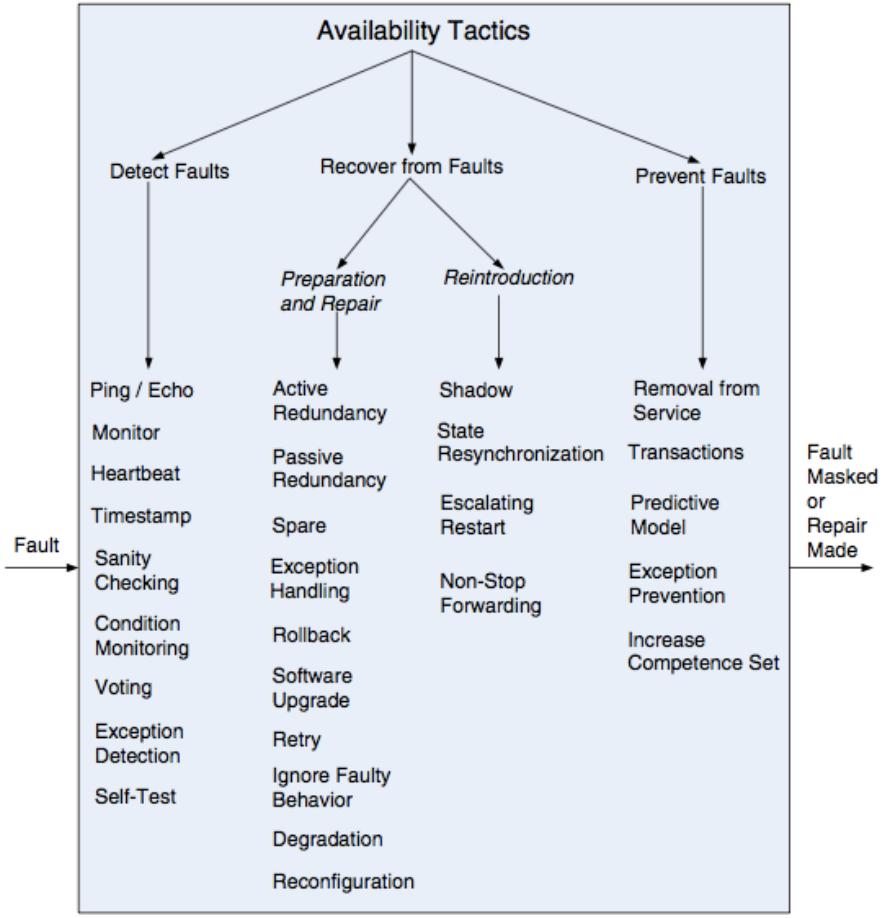


FIGURE 5.4 Goal of availability tactics



Quality Attribute - Interoperability

- The degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context

- The definition includes
  - Syntactic interoperability: The ability to exchange data
  - Semantic interoperability: The ability to correctly interpret the data being exchanged

Quality Attribute Scenario – Interoperability

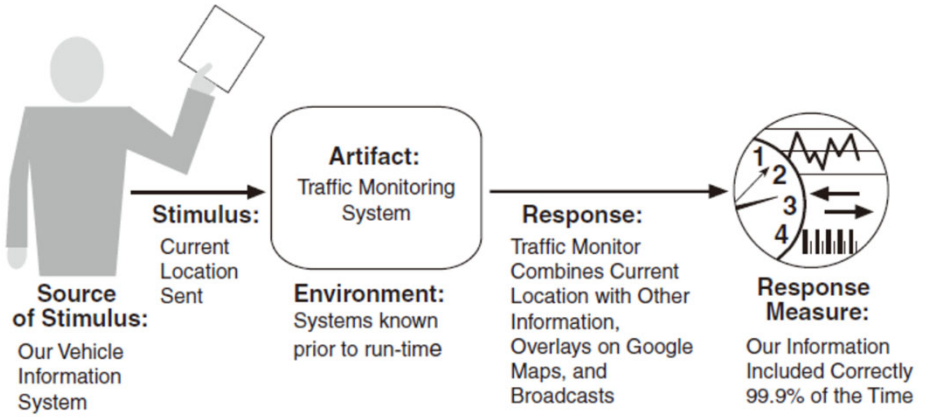


FIGURE 6.1 Sample concrete interoperability scenario

Portion of Scenario	Possible Values
Source	A system initiates a request to interoperate with another system.
Stimulus	A request to exchange information among system(s).
Artifact	The systems that wish to interoperate.
Environment	System(s) wishing to interoperate are discovered at runtime or known prior to runtime.
Response	One or more of the following: ▪ The request is (appropriately) rejected and appropriate entities (people or systems) are notified. ▪ The request is (appropriately) accepted and information is exchanged successfully. ▪ The request is logged by one or more of the involved systems.
Response Measure	One or more of the following: ▪ Percentage of information exchanges correctly processed ▪ Percentage of information exchanges correctly rejected

Tactics for Interoperability

- **Locate**
  - Discover service: Locate a service through searching a known directory service.
- **Manage Interfaces**
  - Orchestrate: Uses a control mechanism to coordinate and manage and sequence the invocation of particular se rvices (which could be ignorant of each other).
  - Tailor interface: Adds or removes capabilities to an interface.

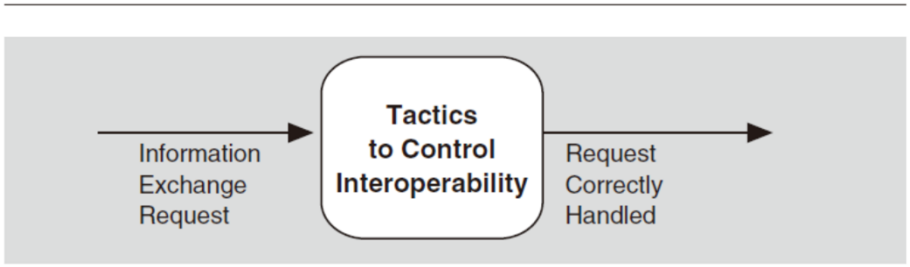
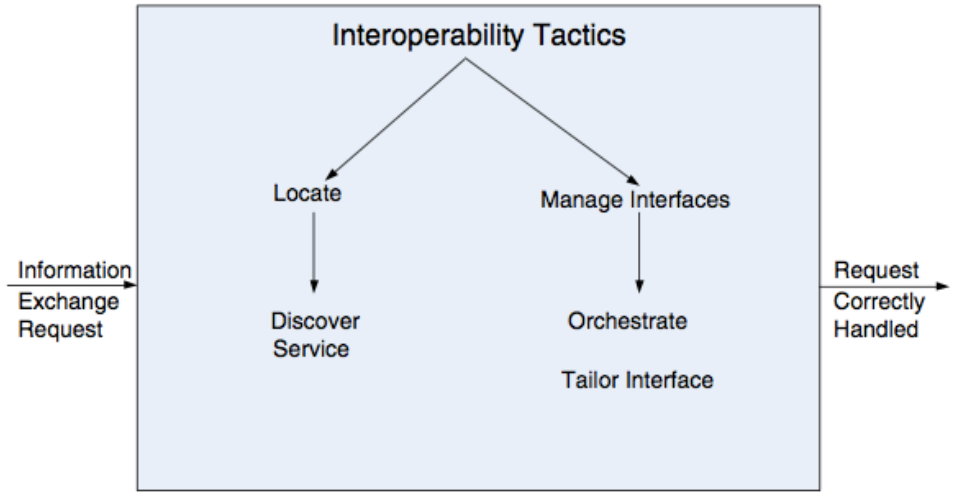


FIGURE 6.2 Goal of interoperability tactics



Quality Attribute - Modifiability

- The ability to quickly make changes to a system at a higher performance-to-price ratio
  - Often based on some specific changes and is measured by examining the costs of these changes.
    - What can change?
    - What is the likelihood of the change?
    - When is the change made and who makes it?
    - What is the cost of the change?

Quality Attribute Scenario – Modifiability

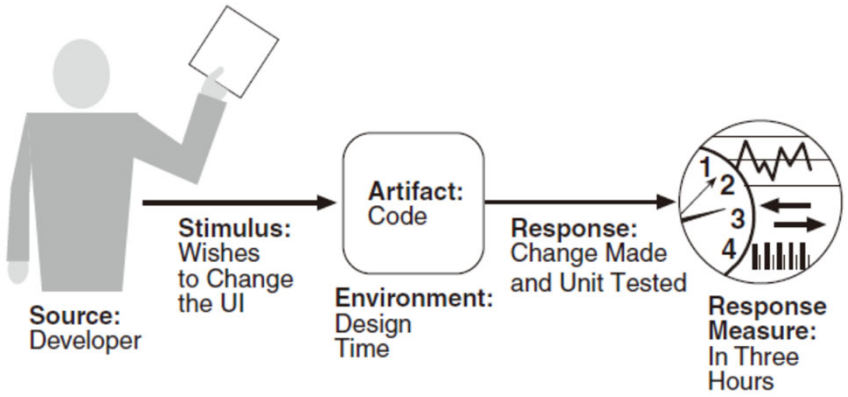


FIGURE 7.1 Sample concrete modifiability scenario

Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ▪ Make modification ▪ Test modification ▪ Deploy modification
Response Measure	Cost in terms of the following: ▪ Number, size, complexity of affected artifacts ▪ Effort ▪ Calendar time ▪ Money (direct outlay or opportunity cost) ▪ Extent to which this modification affects other functions or quality attributes ▪ New defects introduced

Tactics for Modifiability

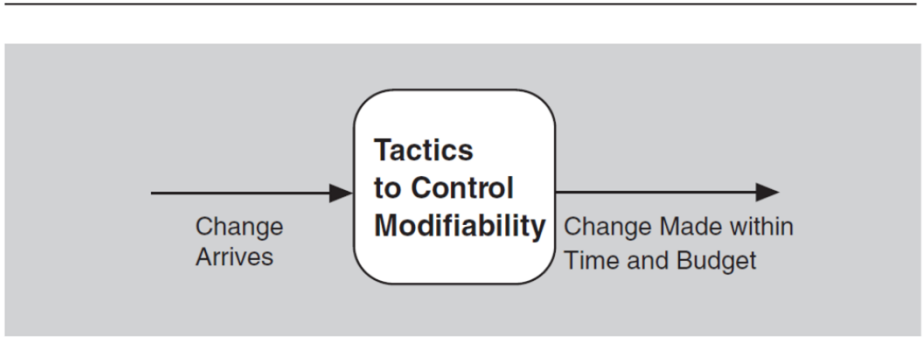
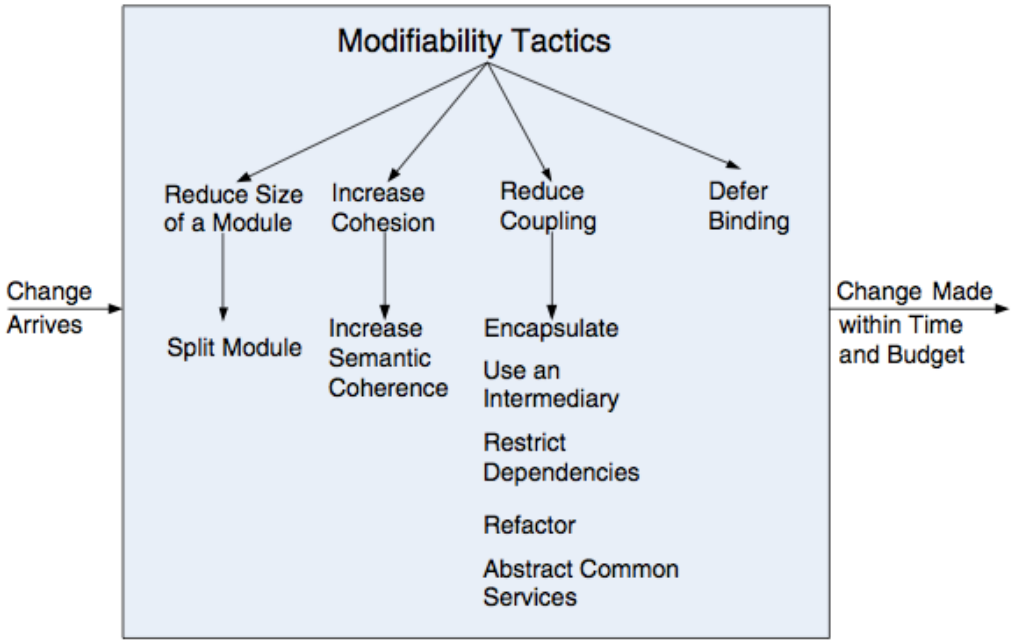


FIGURE 7.2 The goal of modifiability tactics



Quality Attribute - Performance

- About time and the software system's ability to meet timing requirements
  - When events occur, the system or some element of the system must respond to them in time.
    - interrupts, messages, requests from users or other systems, or clock events marking the passage of time
  - Characterizing the events that can occur (and when they can occur) and the system or element's time-based re sponse to those events is the essence is discussing performance.

Quality Attribute Scenario – Performance

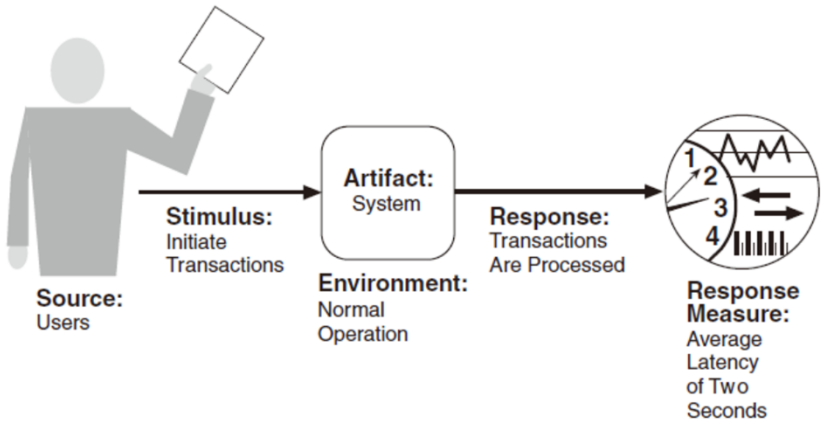


FIGURE 8.1 Sample concrete performance scenario

Portion of Scenario	Possible Values
Source	Internal or external to the system
Stimulus	Arrival of a periodic, sporadic, or stochastic event
Artifact	System or one or more components in the system
Environment	Operational mode: normal, emergency, peak load, overload
Response	Process events, change level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate

Tactics for Performance

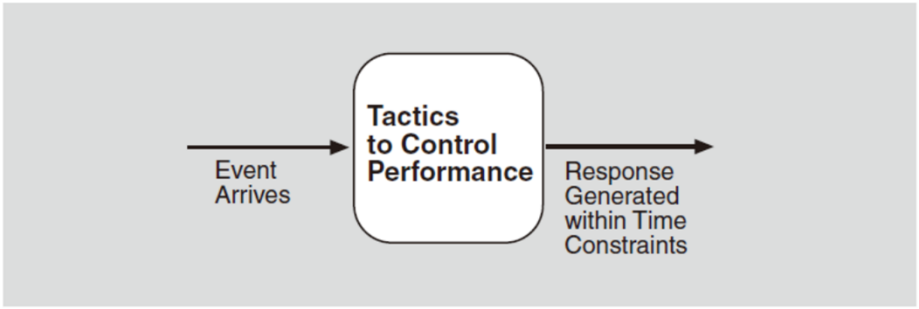
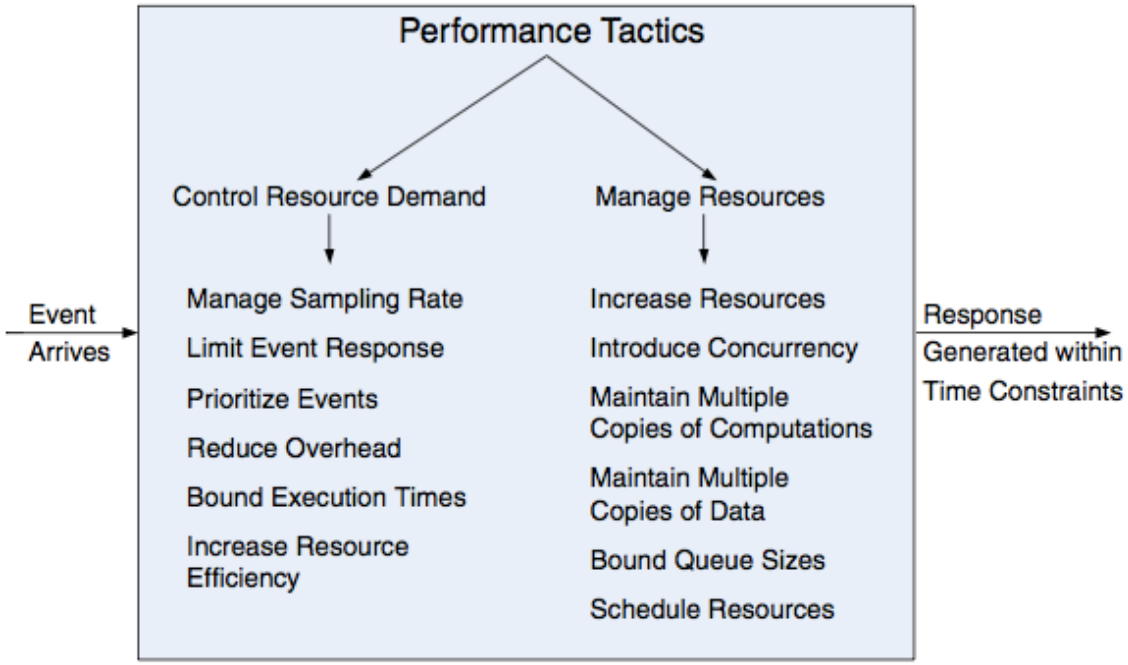


FIGURE 8.2 The goal of performance tactics



Quality Attribute - Security

- A measure of the system's ability to protect data and information from unauthorized access while still pr oviding access to people and systems that are authorized

- The simplest approach to characterizing security has 3 characteristics **(CIA)**:
  - **Confidentiality**
  - **Integrity**
  - **Availability**

- Other characteristics that are used to support CIA are these:
  - Authentication
  - Nonrepudiation
  - Authorization

Quality Attribute Scenario – Security

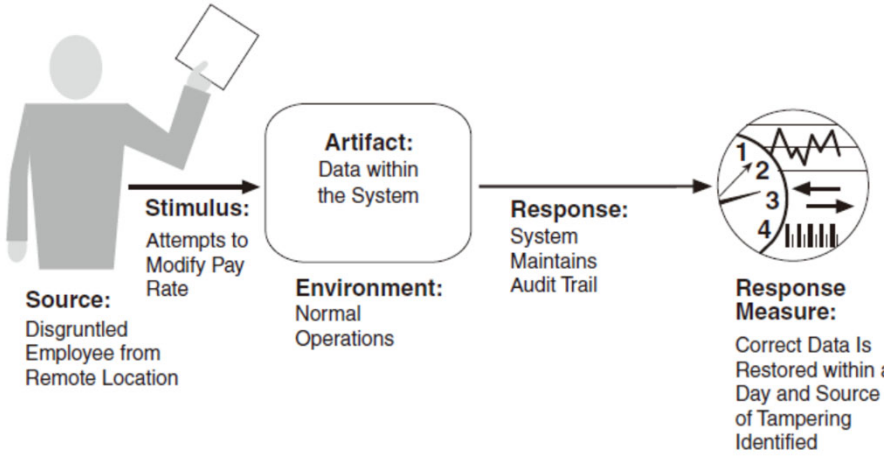


FIGURE 9.1 Sample concrete security scenario

Portion of Scenario	Possible Values
Source	Human or another system which may have been previously identified (either correctly or incorrectly) or may be currently unknown. A human attacker may be from outside the organization or from inside the organization.
Stimulus	Unauthorized attempt is made to display data, change or delete data, access system services, change the system's behavior, or reduce availability.
Artifact	System services, data within the system, a component or resources of the system, data produced or consumed by the system
Environment	The system is either online or offline; either connected to or disconnected from a network; either behind a firewall or open to a network; fully operational, partially operational, or not operational.
Response	Transactions are carried out in a fashion such that ▪ Data or services are protected from unauthorized access. ▪ Data or services are not being manipulated without authorization. ▪ Parties to a transaction are identified with assurance. ▪ The parties to the transaction cannot repudiate their involvements. ▪ The data, resources, and system services will be available for legitimate use. The system tracks activities within it by ▪ Recording access or modification ▪ Recording attempts to access data, resources, or services ▪ Notifying appropriate entities (people or systems) when an apparent attack is occurring
Response Measure	One or more of the following: ▪ How much of a system is compromised when a particular component or data value is compromised ▪ How much time passed before an attack was detected ▪ How many attacks were resisted ▪ How long does it take to recover from a successful attack ▪ How much data is vulnerable to a particular attack

Tactics for Security

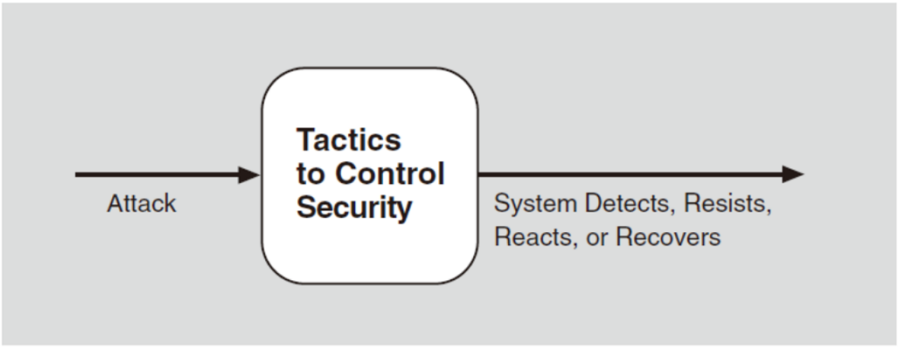
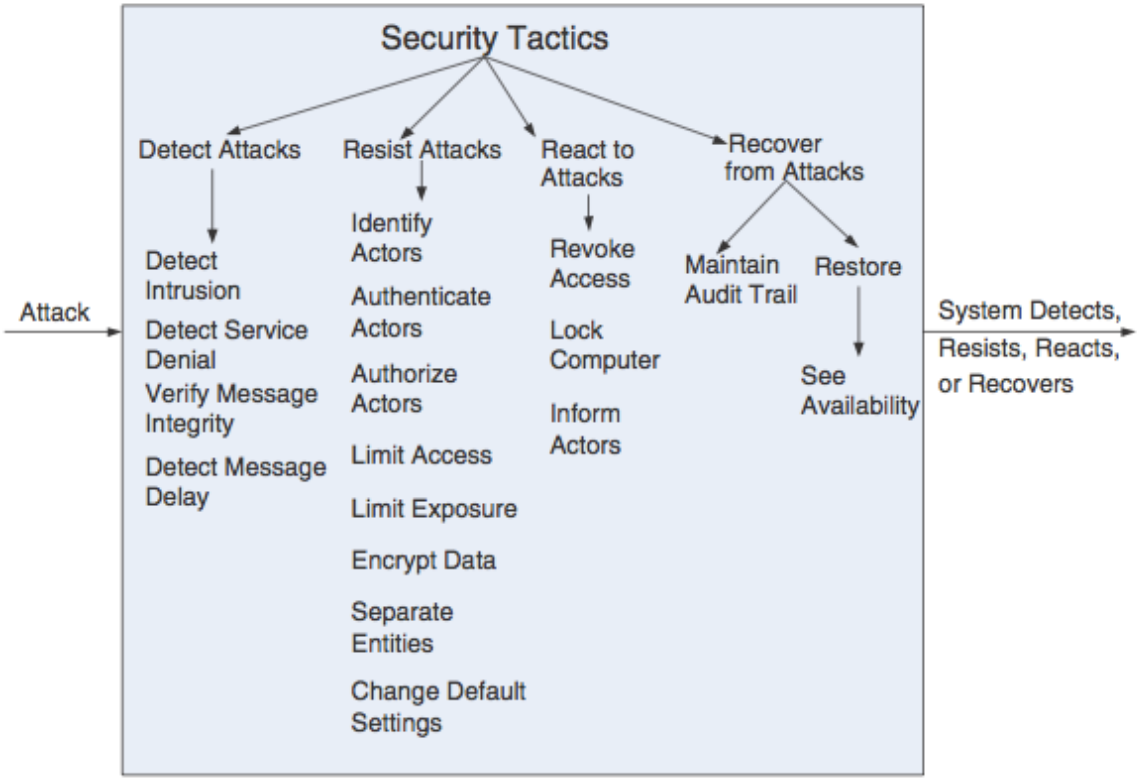


FIGURE 9.2 The goal of security tactics



Quality Attribute - Testability

- The ease with which software can be made to demonstrate its faults through (typically execution-based) testing
  - Specifically, testability refers to the probability, assuming that the software has at least one fault, that it will fail o n its next test execution.
  - Intuitively, a system is testable if it “gives up” its faults easily.

Quality Attribute Scenario – Testability

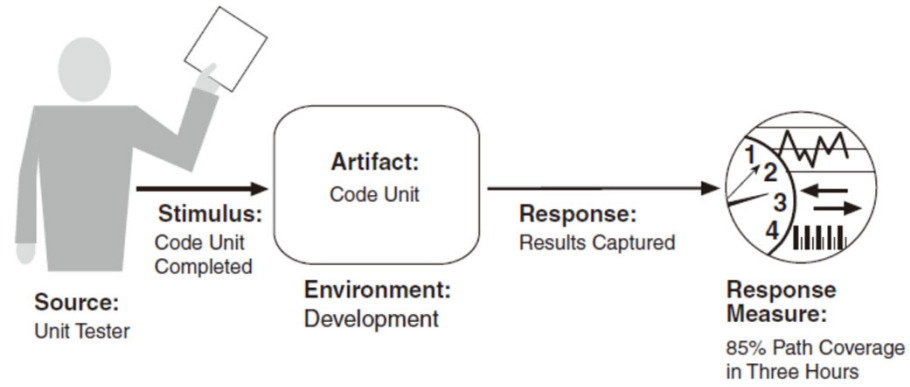


FIGURE 10. 2 Sample concrete testability scenario

Portion of Scenario	Possible Values
Source	Unit testers, integration testers, system testers, acceptance testers, end users, either running tests manually or using automated testing tools
Stimulus	A set of tests is executed due to the completion of a coding increment such as a class layer or service, the completed integration of a subsystem, the complete implementation of the whole system, or the delivery of the system to the customer.
Environment	Design time, development time, compile time, integration time, deployment time, run time
Artifacts	The portion of the system being tested
Response	One or more of the following: execute test suite and capture results, capture activity that resulted in the fault, control and monitor the state of the system
Response Measure	One or more of the following: effort to find a fault or class of faults, effort to achieve a given percentage of state space coverage, probability of fault being revealed by the next test, time to perform tests, effort to detect faults, length of longest dependency chain in test, length of time to prepare test environment, reduction in risk exposure (size(loss) × prob(loss))

Tactics for Testability

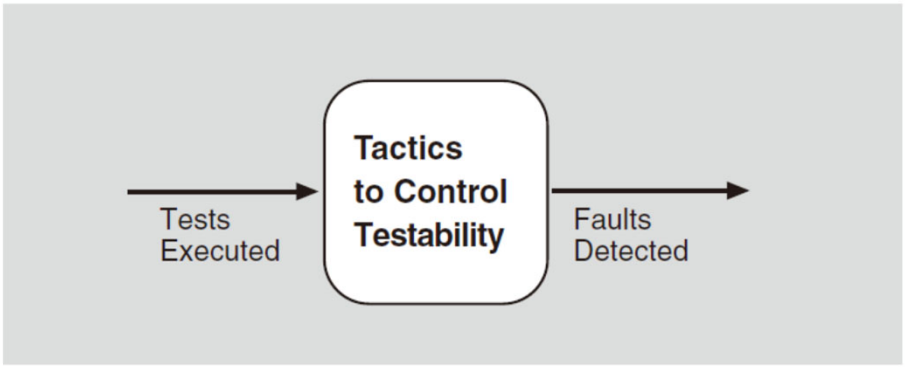
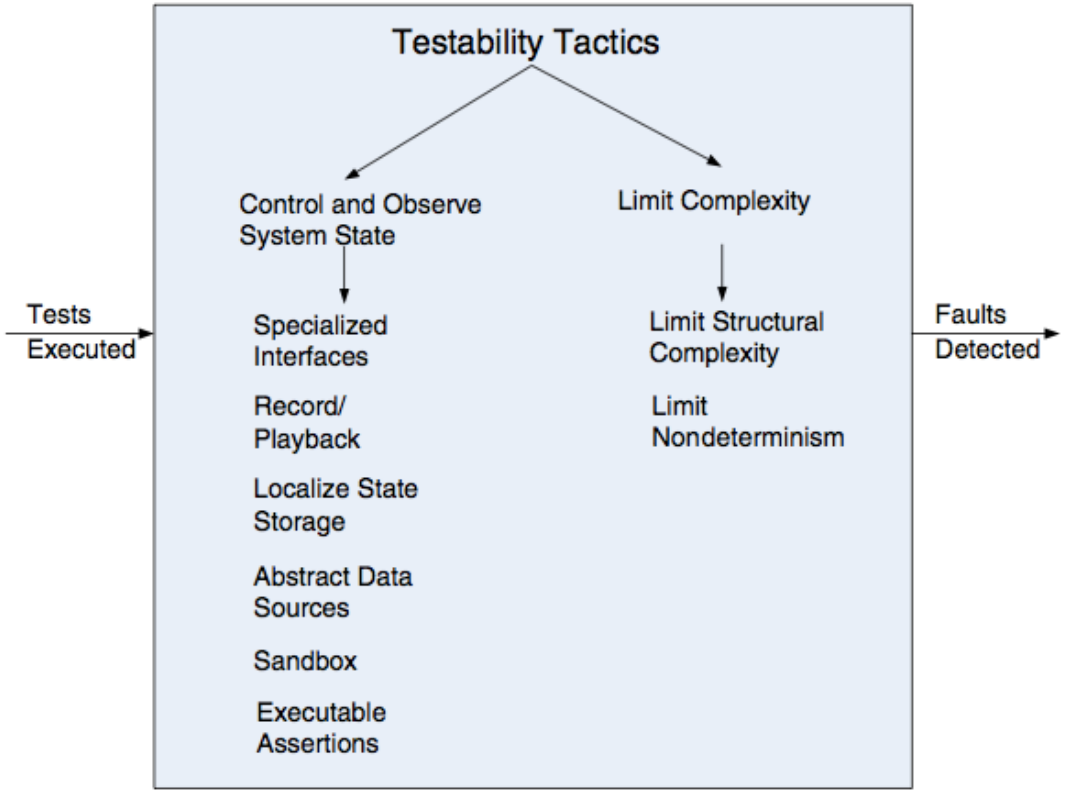


FIGURE 10.3 The goal of testability tactics



Quality Attribute - Usability

- Concerned with how easy it is for the user to accomplish a desired task and the kind of user support th e system provides

- Usability comprises the following areas:
  - Learning system features.
  - Using a system efficiently.
  - Minimizing the impact of errors.
  - Adapting the system to user needs.
  - Increasing confidence and satisfaction.

Quality Attribute Scenario – Usability

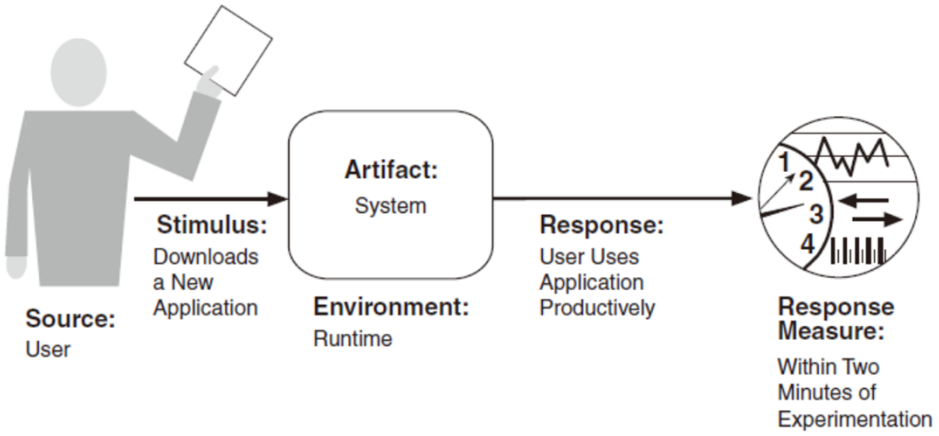


FIGURE 11.1 Sample concrete usability scenario

Portion of Scenario	Possible Values
Source	End user, possibly in a specialized role
Stimulus	End user tries to use a system efficiently, learn to use the system, minimize the impact of errors, adapt the system, or configure the system.
Environment	Runtime or configuration time
Artifacts	System or the specific portion of the system with which the user is interacting
Response	The system should either provide the user with the features needed or anticipate the user's needs.
Response Measure	One or more of the following: task time, number of errors, number of tasks accomplished, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, or amount of time or data lost when an error occurs

Tactics for Usability

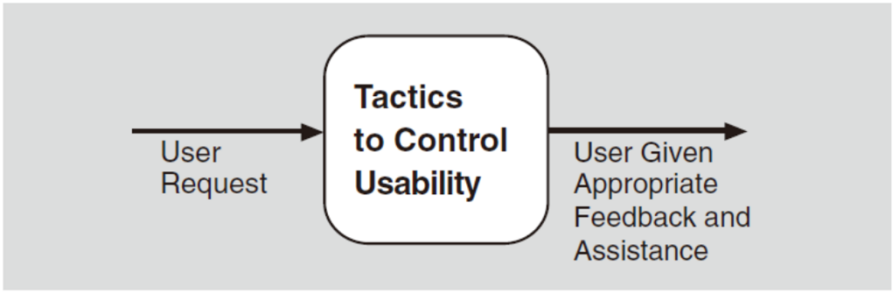
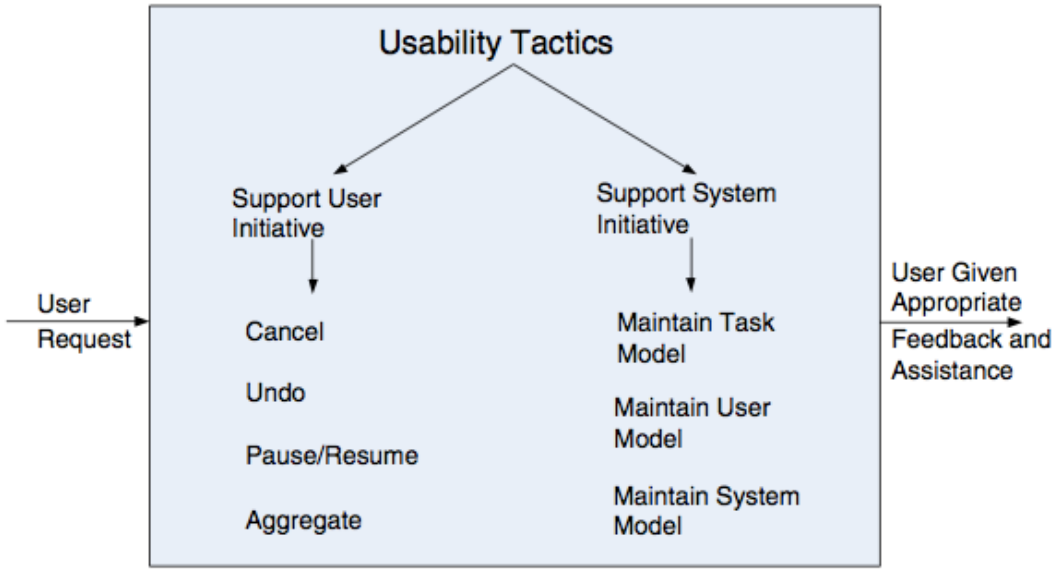


FIGURE 11.2 The goal of runtime usability tactics



Quality Requirements and Architecture Evaluation

- Quality requirements gives important information such as
  - "*Is the architecture suitable for the system for which it was devised?*"
  - "*Which of two competing architectures is most suitable for the system at hand?*"

- **An architecture is suitable if**,
  - The system that results from it **will meet its quality goals**.
    - A system is modifiable or not wrt. a specific kind of change.
    - A system is secure or not wrt. a specific kind of threat.
    - A system is reliable or not wrt. a specific kind of fault occurrence.
    - A system performs well or not wrt. specific performance criteria.
    - An architecture is buildable or not wrt. specific time and budget constraints.

- Questioning techniques for **architecture evaluation**
  - Rely on thought experiments to check architecture suitability
  - Scenario-based style: **ATAM** **(Architecture Tradeoff Analysis Method)**
  - Checklist-based style

Quality Attribute Workshop (QAW)

- **Quality Attribute Workshop (QAW)**
  - Facilitated method
    - System-centric
    - Used before the software architecture has been created
  - Engages system stakeholders early in the life-cycle
  - Reveals the driving quality attribute requirements of a software-intensive system
    - Scenario-based

- Outputs of a QAW
  - **Quality attribute requirements** for the system, documented as refined and prioritized **QAS**.
  - The quality attribute scenarios can then be used as the basis for designing the software architecture for the system.

The QAW Steps

1. **QAW Introduction**
2. **Business/Mission Presentation**
3. **Architectural Plan Presentation**
4. **Identification of Architectural Drivers**
5. **Scenario Brainstorming**
6. **Scenario Consolidation**
7. **Scenario Prioritization**
8. **Scenario Refinement**

The QAW Steps in Detail

1. **QAW Presentation and Introduction**
   - QAW facilitators describe the motivation for the QAW and explain each step of the method.
2. **Business/Mission Presentation**
   - A stakeholder presents the business and/or programmatic drivers for the system.
3. **Architectural Plan Presentation**
   - A technical stakeholder presents the system architectural plans as they stand with respect to early documents, such as high-level system descriptions, context drawings, or other artifacts that describe the system's technical details.
4. **Identification of Architectural Drivers**
   - Architectural drivers often include high-level requirements, business/mission concerns, and various quality attributes.
   - During this step, the facilitators and stakeholders reach a consensus about which drivers are key to the system.
5. **Scenario Brainstorming**
   - Stakeholders generate real-world scenarios for the system. Scenarios comprise a related stimulus, an environmental condition, and a response.
   - Facilitators ensure that at least one scenario addresses each of the architectural drivers identified in Step 4.
6. **Scenario Consolidation**
   - Scenarios that are similar in content are consolidated.
7. **Scenario Prioritization**
   - Stakeholders prioritize the scenarios through a voting process.
8. **Scenario Refinement**
   - For the top four or five scenarios, the following are described: the business/mission goals that are affected by those scenarios, the relevant quality attributes associated with those scenarios

Mini-QAW

1. **Mini-QAW Introduction**
2. **Introduction to Quality Attributes, Quality Attributes Taxonomy**
3. **Scenario Brainstorming**
   - **“Walk the System Properties Web” activity**
4. **Raw Scenario Prioritization**
   - **Dot voting**
5. **Scenario Refinement**
   - **While time remains**
6. **Review Results with Stakeholders**

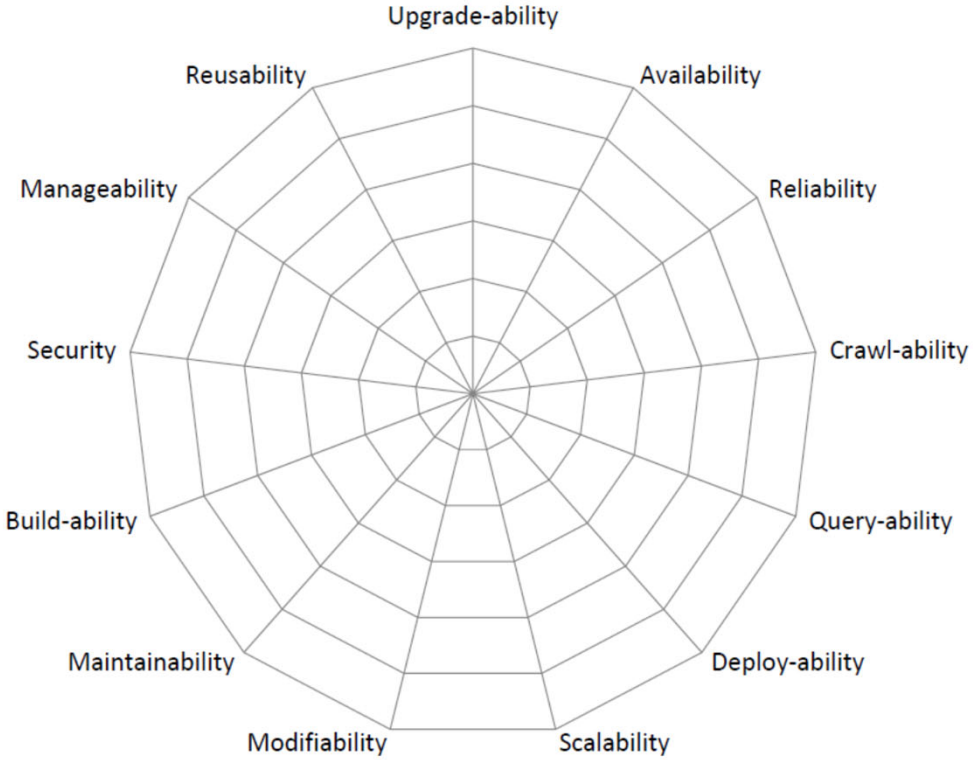
1. Mini-QAW Introduction

- First, sketch **a rough architecture** and **major objectives/functions**
- Take into account specific roles of all stakeholders.
  - For example, “Accessary Service Framework” may have 6~8 different stakeholders and goals.

Stakeholders	주요 역할	희망사항 - Goal
안드로이드 OS 관리자	안드로이드 OS가 안정되게 동작하도록 관리한다.	• 안드로이드 Interface를 확장하지 않았으면 좋겠다. • Interface에 adapter나 wrapper를 붙이지 않았으면 좋겠다. • (가능하면 자세하게)

2. Introduction to Quality Attributes, Quality Attributes Taxonomy

- Define your own **system properties web**
  - Select appropriate quality factors for your system under consideration.



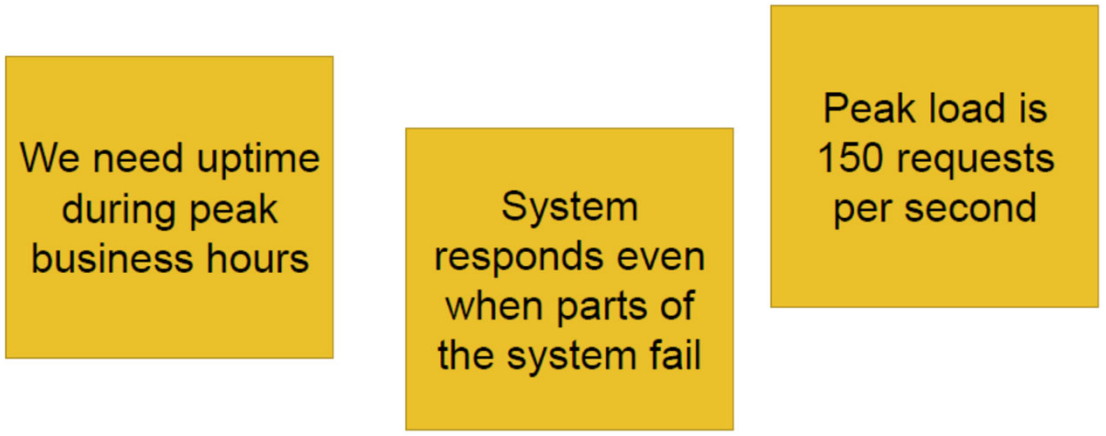
3. Scenario Brainstorming

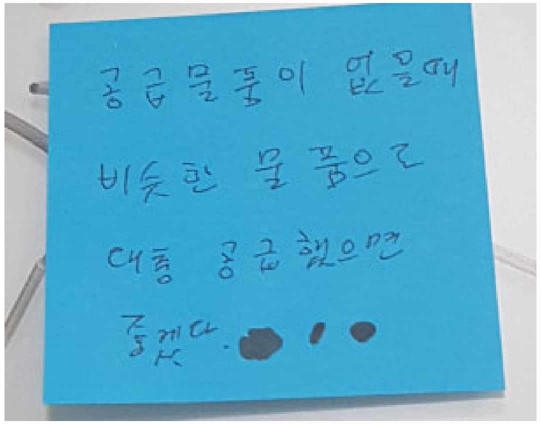
- Identify raw quality attribute scenarios
  - Timing: 30 minutes to 2-3 hours

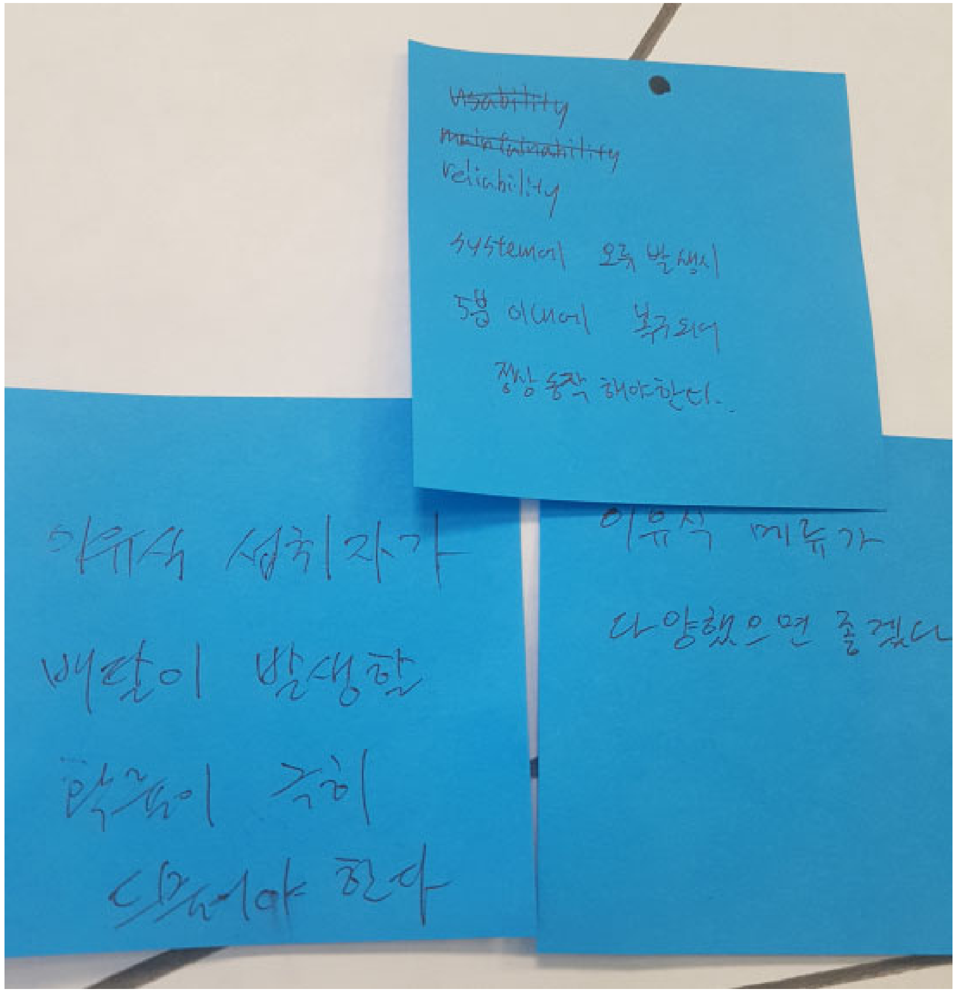
- Steps:
  1. Start with a Scenario on the web, ask “Is this Quality attribute relevant to your system?”
  2. If Yes, spend 5 minutes brainstorming scenarios / concerns on that scenario.
  3. Write raw scenarios on stickies and put on web
  4. After 5 minutes, move to next scenario

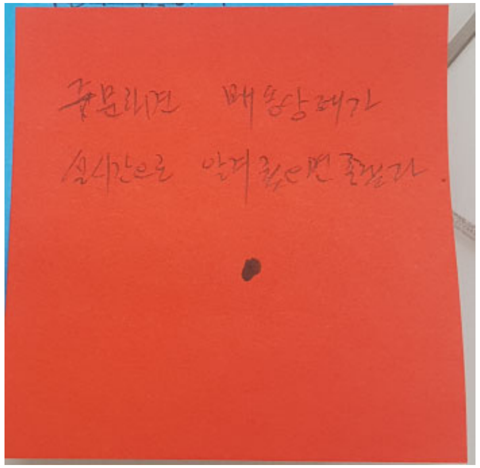
Raw Quality Attribute Scenario

- Informally describes a stakeholder's concern and concrete instances of quality attributes

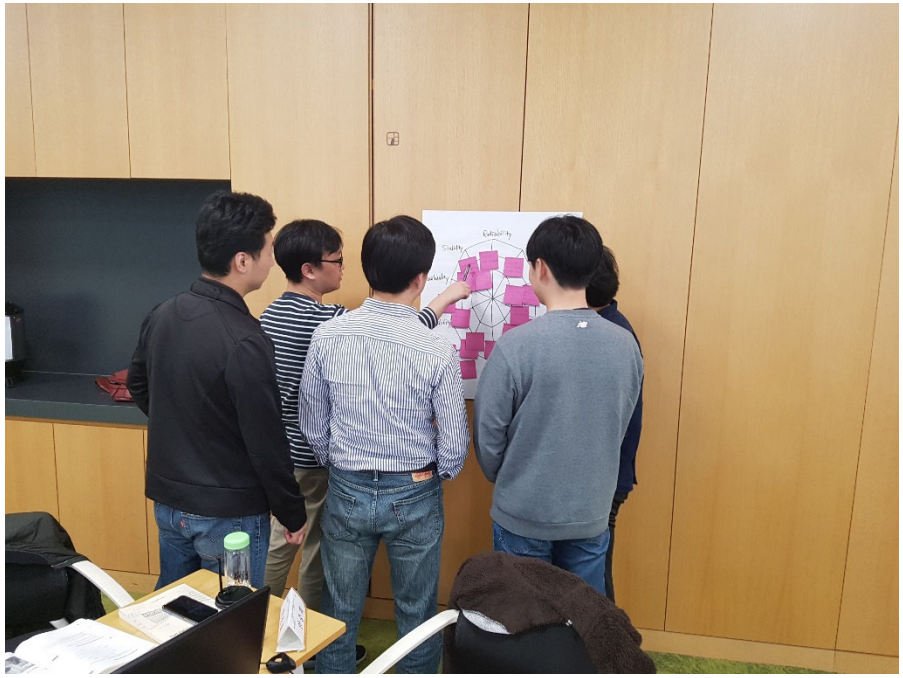


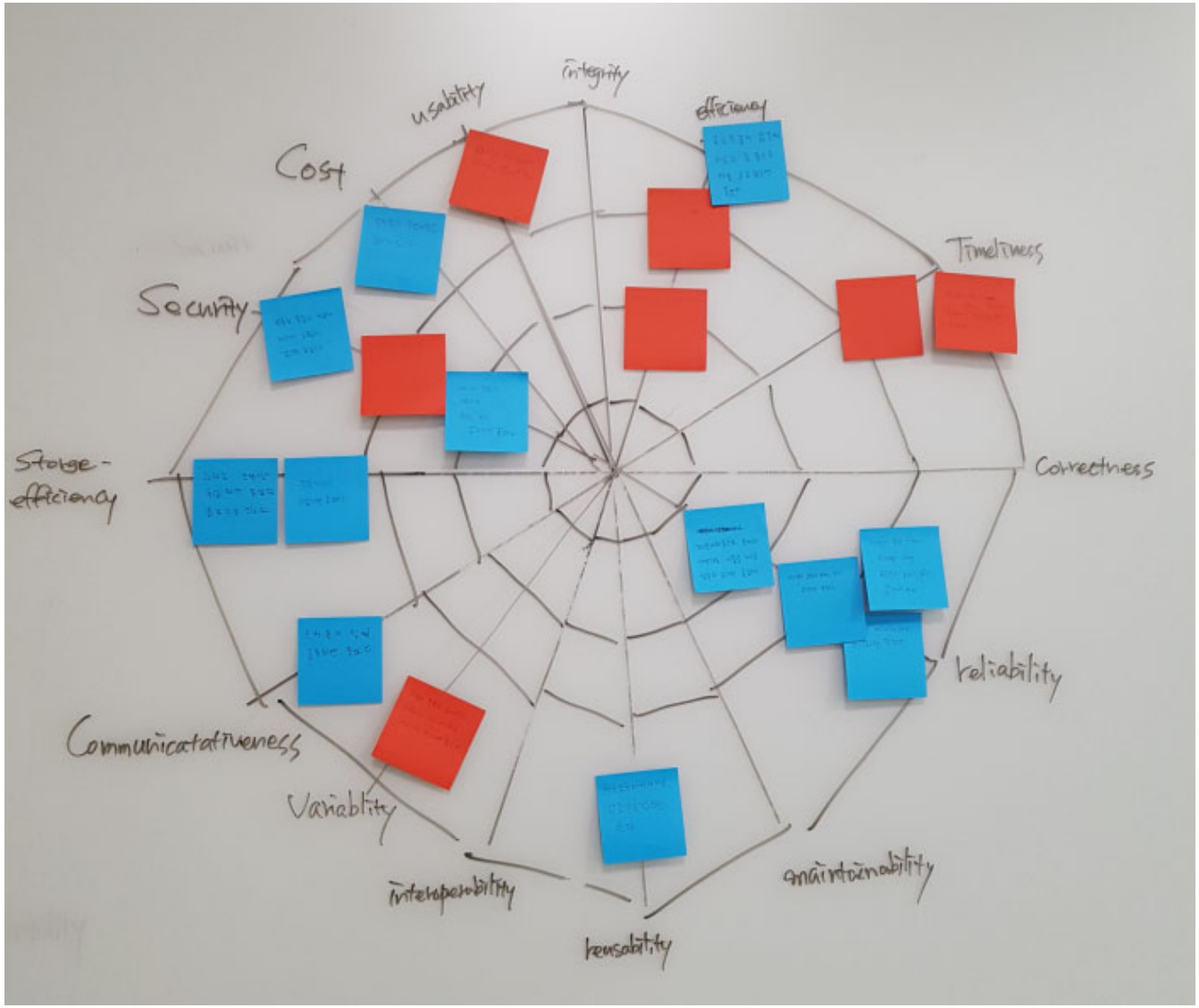

공급물품이 없을때
비슷한 물품으로
대충 공급했으면
좋겠다.


~~usability~~
~~maintainability~~
reliability
system에 오류 발생시
5분 이내에 복구되어
정상 동작 해야한다.
이유식 섭취자가
배탈이 발생할
확률이 극히
드물어야 한다
이유식 메뉴가
다양했으면 좋겠다


주문하면 배송상태가
실시간으로 알려줬으면 좋겠다.

“Walk the System Properties Web” Activity




Cost
usability
integrity
efficiency
Timeliness
Security
Correctness
Storage-
efficiency
Reliability
Communicativeness
Variablity
interoperability
maintainability
reusability

로봇의 이상징후가 미리 감지되어야 한다.

부품의 재고 현황을 쉽게 파악했으면 좋겠다

유사시에도 현장 모니터링이 가능하도록 CCTV 화면도 동시에 또는 전환해가며 볼 수 있어야 한다.

'이슈가 발생한 구간이 눈에 띄는 색으로 강조되어 표시되면 좋겠다

상품이 실수 없이 정확하게 목적지에 도착해야 한다

Maintainability

Reliability

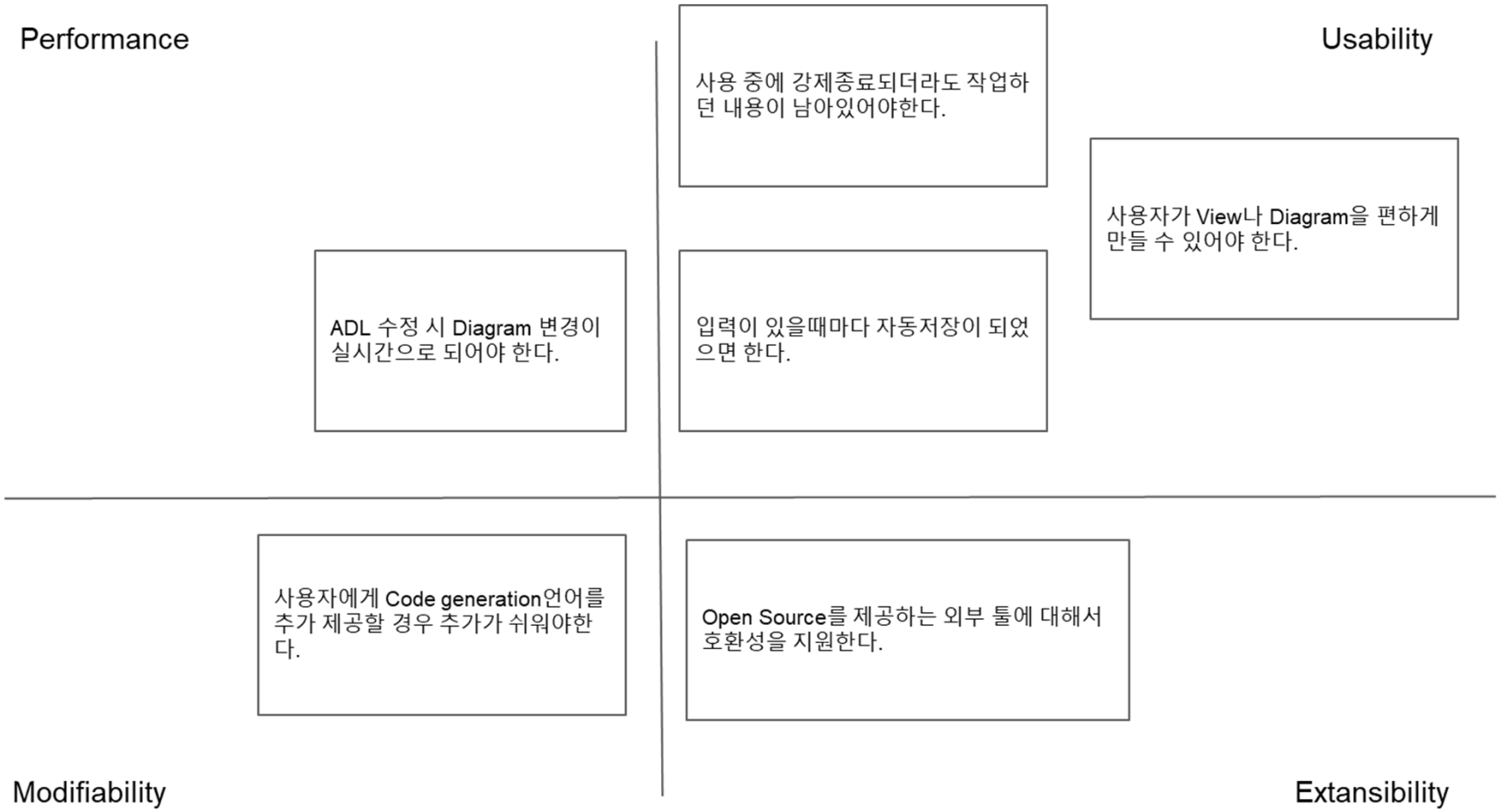
Upgrade-ability

소프트웨어 이슈가 있을 시 유연하게 대처할 수 있도록 쉬운 업그레이드가 가능해야 한다.

Safety

현장 출입 시 안전을 위한 보호 장비가 제공되어야 한다.

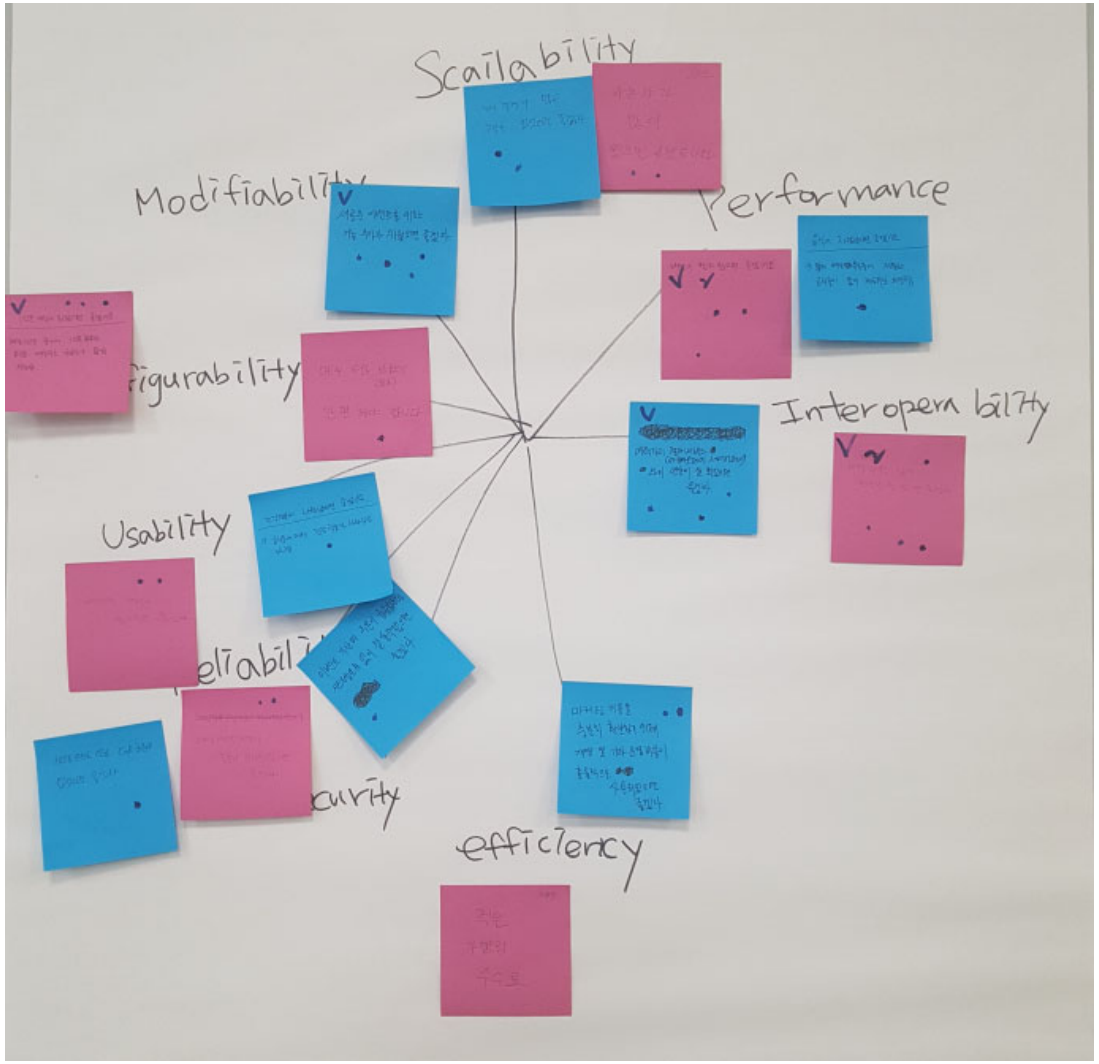
문제 발생 시 긴급하게 중단할 수 있어야 한다


Performance
Usability
사용 중에 강제종료되더라도 작업하던 내용이 남아있어야한다.
사용자가 View나 Diagram을 편하게 만들 수 있어야 한다.
ADL 수정 시 Diagram 변경이 실시간으로 되어야 한다.
입력이 있을때마다 자동저장이 되었으면 한다.
사용자에게 Code generation언어를 추가 제공할 경우 추가가 쉬워야한다.
Open Source를 제공하는 외부 툴에 대해서 호환성을 지원한다.
Modifiability
Extansibility

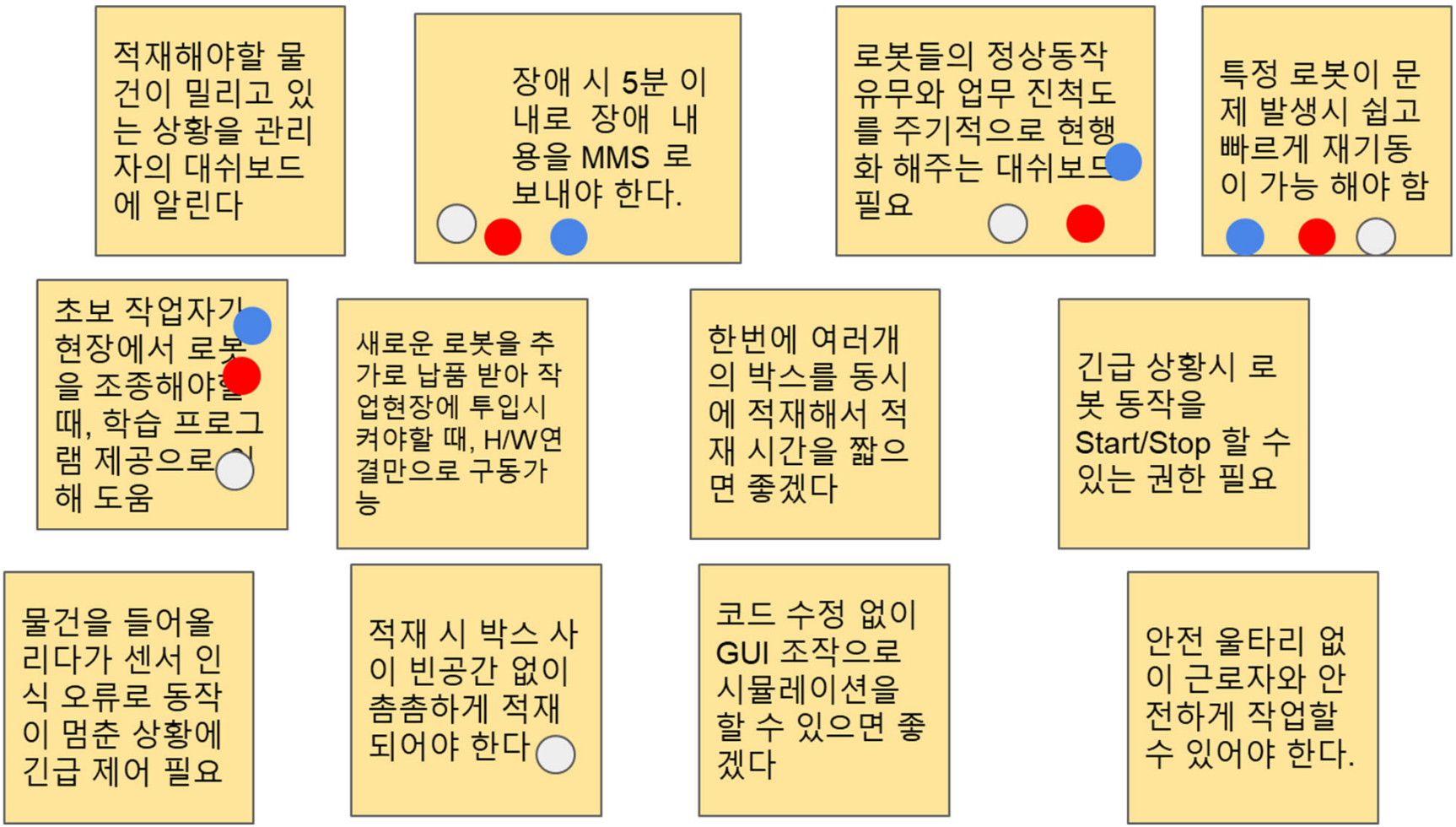
4. Raw Scenario Prioritization

- Identify Highest Priority Scenarios using dot voting
  - Timing : 5 minutes

- Steps:
  - Dot Voting:
    - Each stakeholder gets n / 3 + 1 dots for scenarios where n = # scenarios
    - 2 votes to choose “top quality attribute”


Scailability
Modifiability
Performance
Interopern bility
Usability
efficiency

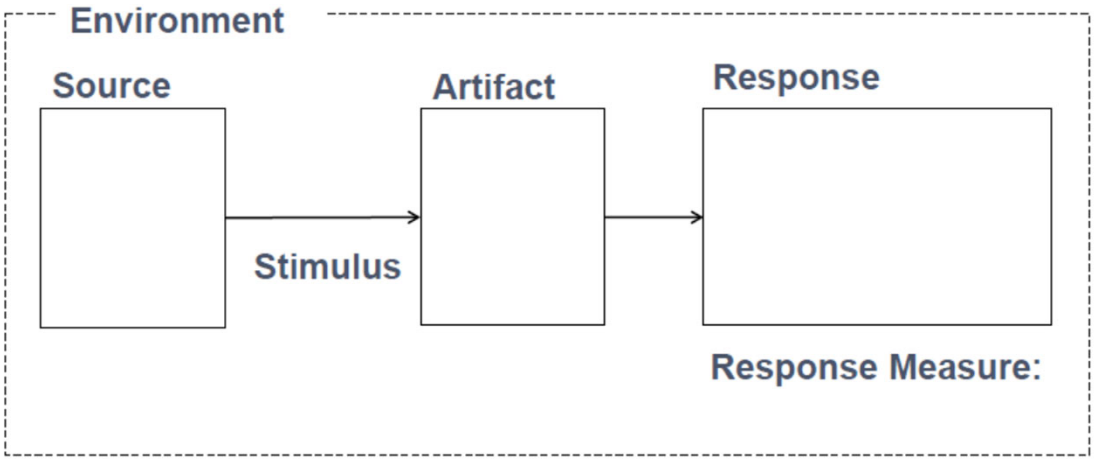
Raw Quality Attribute Scenario



5. Scenario Refinement

- Generate Quality Attribute Scenarios based on raw notes
  - Timing : 30 - 60 minutes

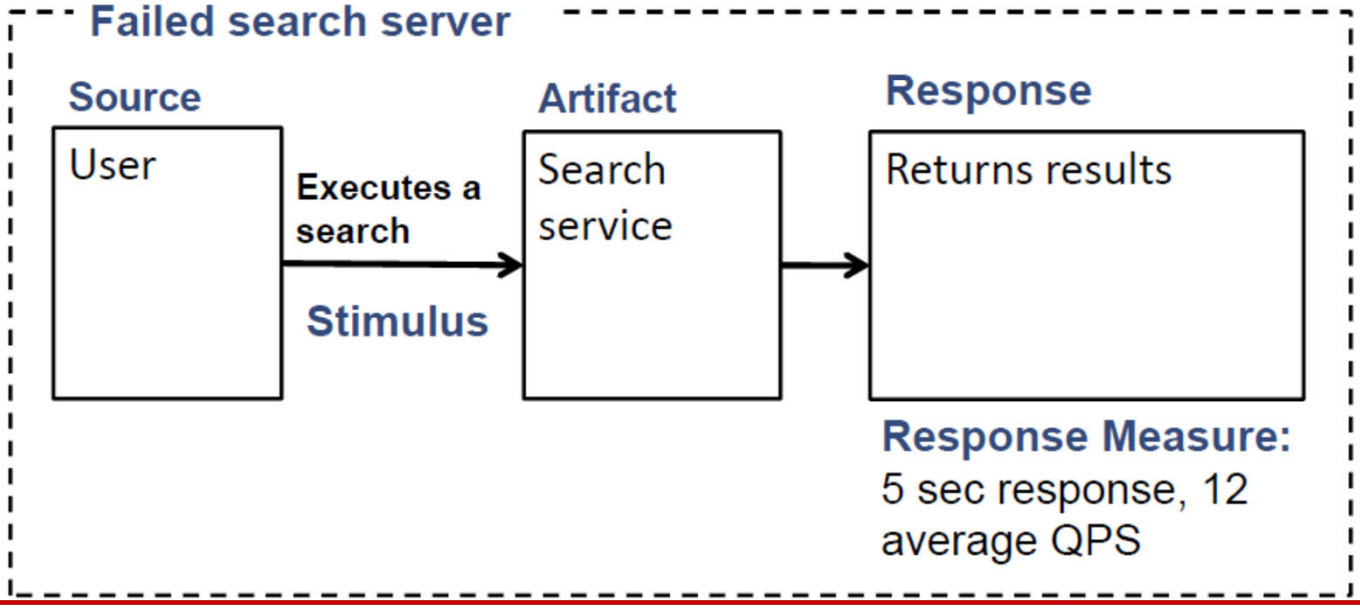
- Steps :
  1. Start with high priority scenario
  2. Fill out the worksheet, identifying the components of a quality attribute scenario
  3. Complete and present to stakeholders



Availability

Example

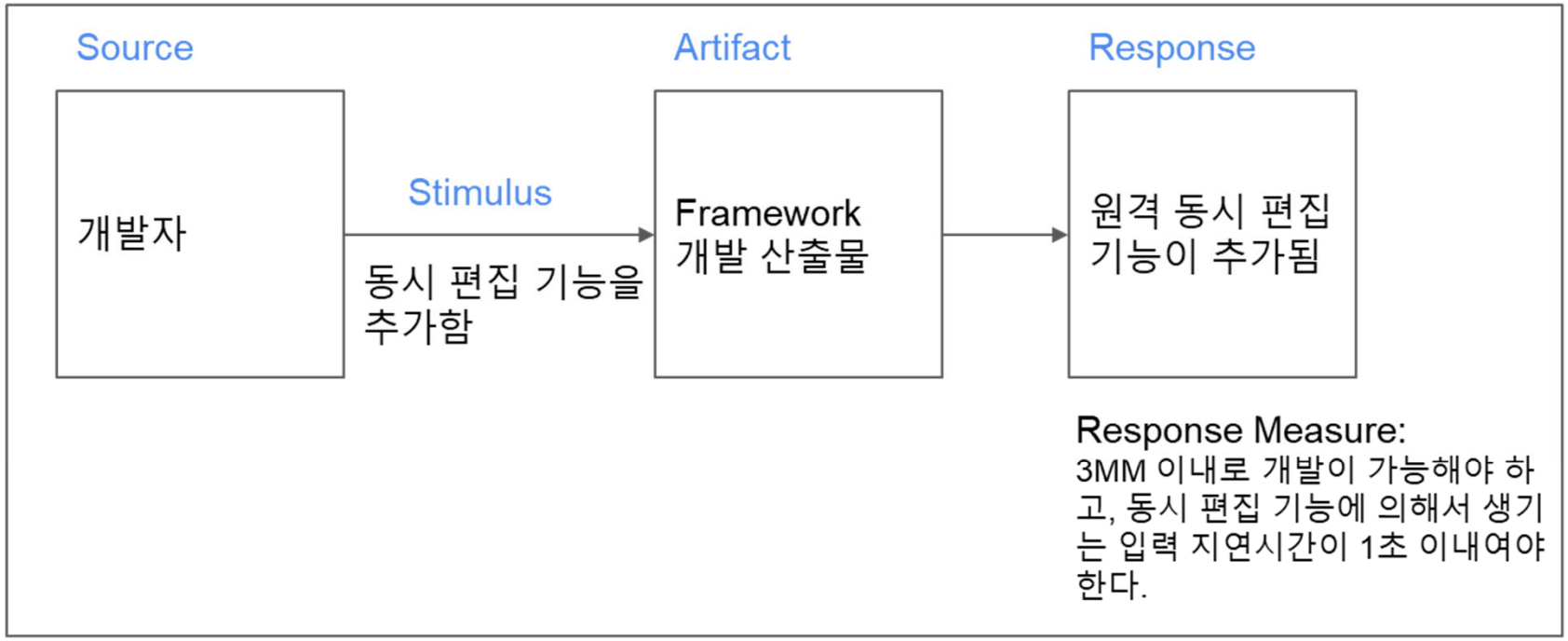
Raw Scenario: In the event of hardware failure, search service is expected to return results during normal working hours for US services representatives.



Refined Scenario: In the event of hardware failure, search service is expected to return results within 5 sec, in 12 average QPA (Queries Per Sec)

Raw scenario: Framework에 원격 동시 편집 기능을 추가하려고 할때 쉽게 추가할 수 있어야 한다 (Modifiability)

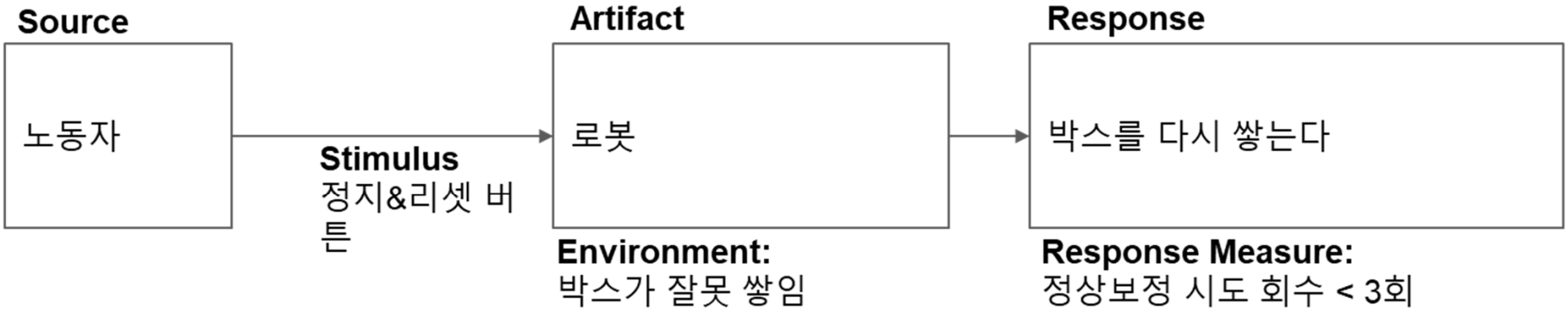
Env: Framework가 로컬 편집 기능만 지원



Refined Scenario: Framework에 동시 편집 기능을 추가하려고 할때, 3MM 이내로 개발이 가능해야 하고, 동시 편집 기능에 의해서 생기는 입력 지연시간이 1초 이내여야 한다

Scenario Refinement: Robustness

Raw Scenario : 로봇이 물건을 잘못쌓아도 다시 시작시키면 다음번은 정상적으로 쌓는다.



Refined Scenario :
로봇이 물건을 잘못쌓은 경우, 노동자가 로봇을 정지하고 reset 버튼을 누르면 2회 이내에 박스를 원래 위치에 두고 다시 정상적으로 쌓는다.

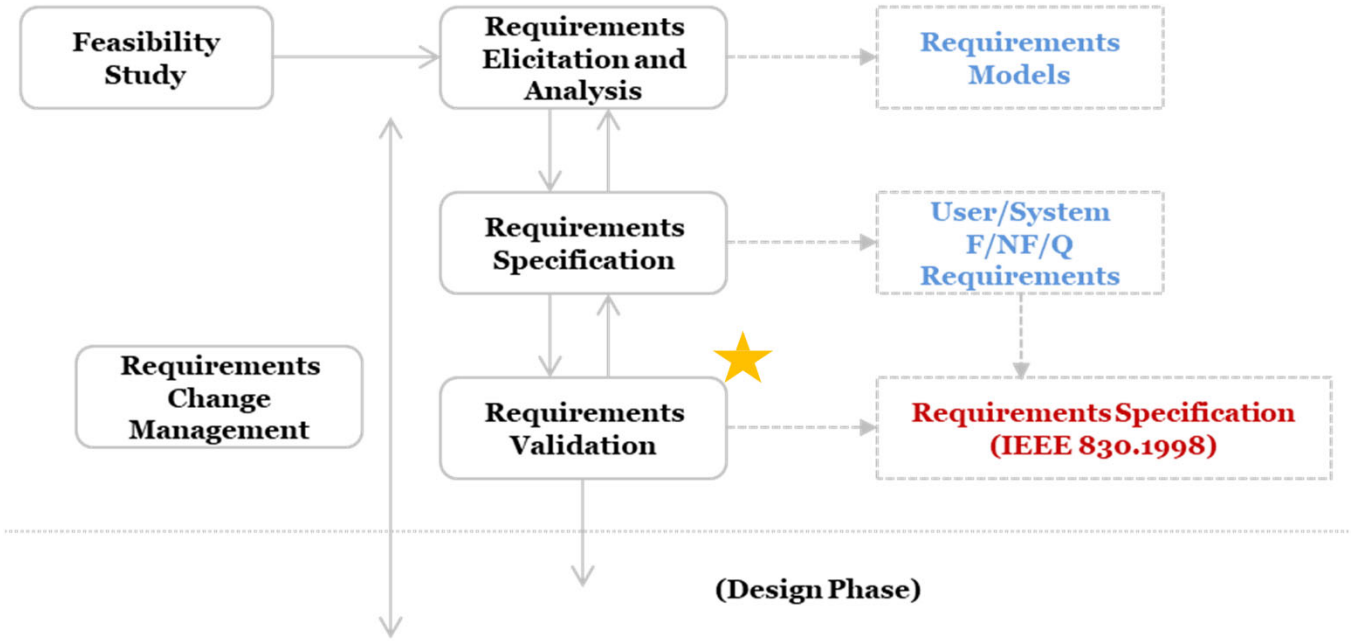
Exercise 6: Mini-QAW

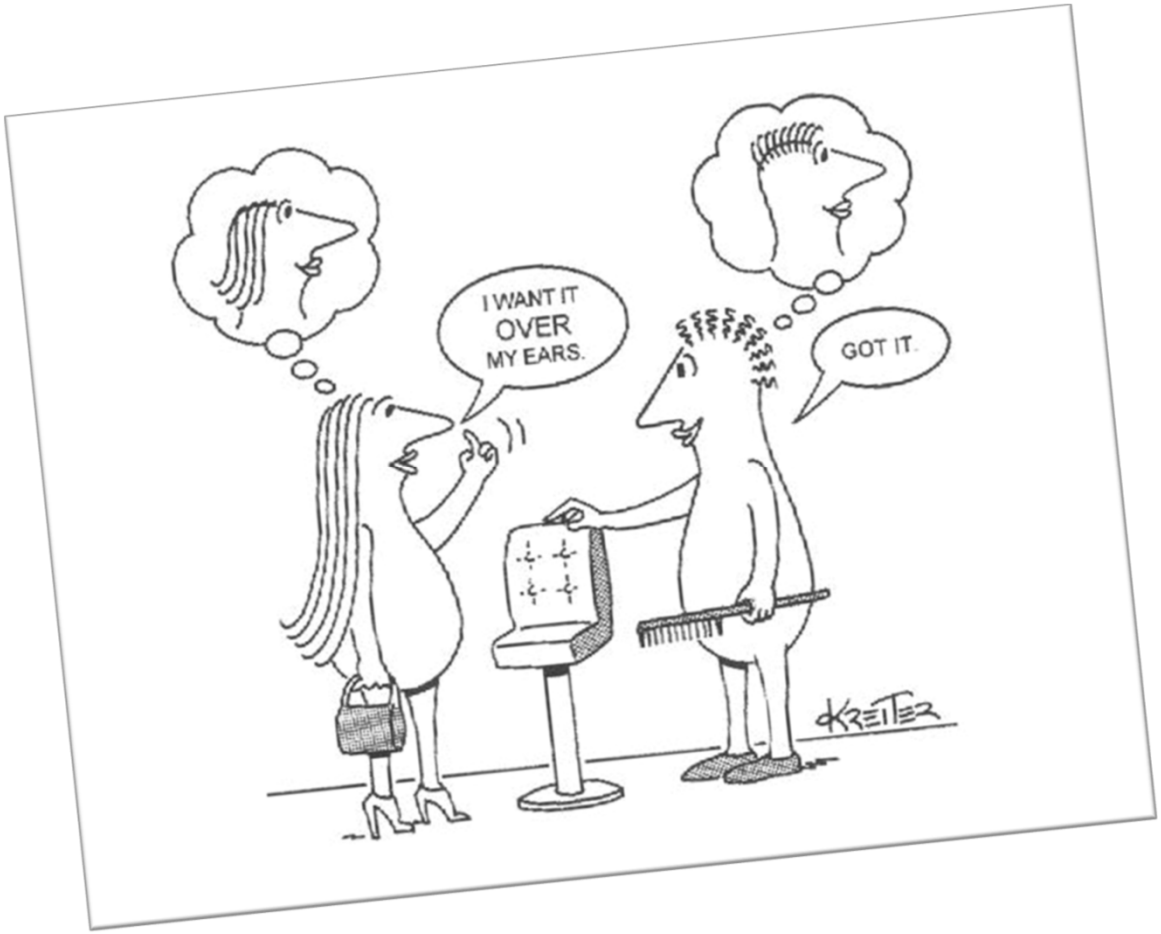
- Perform the **Mini-QAW** for your "**Advanced OOO Digital Watch System**"
  - Follow the steps of Mini-QAW
  - Refine 4 QASs

- **The Mini-QAW steps :**
  1. Mini-QAW Introduction
     - Assign different roles of stakeholders to all team members
     - Define/share the overall context/boundary of the system under consideration (SUC)
  2. Introduction to Quality Attributes, Quality Attributes Taxonomy
     - Select 4~8 quality factors relevant to the SUC
  3. Scenario Brainstorming
     - Identify 20 raw quality attribute scenarios
     - "Walk the System Properties Web" activity
  4. Raw Scenario Prioritization
     - Dot voting to select 4 scenarios
  5. Scenario Refinement
     - Generate 4 well-refined QASs
     - Find an appropriate tactics for each QAS
  6. Review Results with Stakeholders

9. Requirements Validation

Requirements Engineering Process

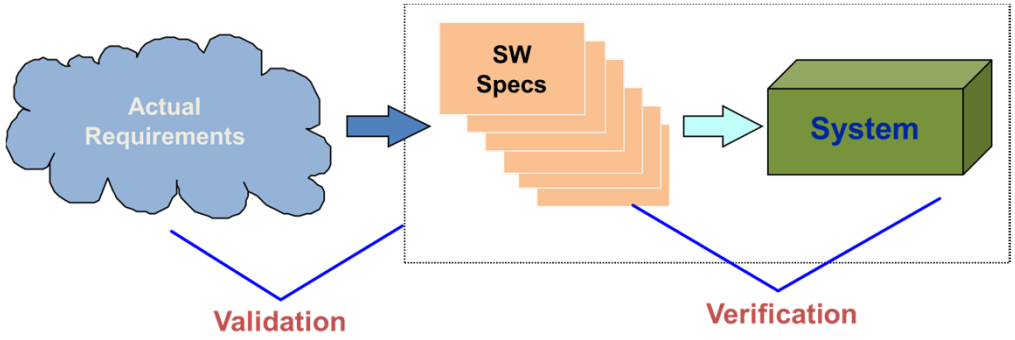



I WANT IT OVER MY EARS.
GOT IT.
KREITER

Verification and Validation in SDLC

- **Validation**: "*Does the software system meets the user's real needs?*"
  - Are we building the right software?
  - Does our problem statement accurately capture the real problem?
  - Did we account for the needs of all the stakeholders?

- **Verification**: "*Does the software system meets the requirements specifications?*"
  - Are we building the software right?
  - Does our design meet the spec?
  - Does our implementation meet the spec?
  - Does the delivered system do what we said it would do?
  - Are our requirements models consistent with one another?

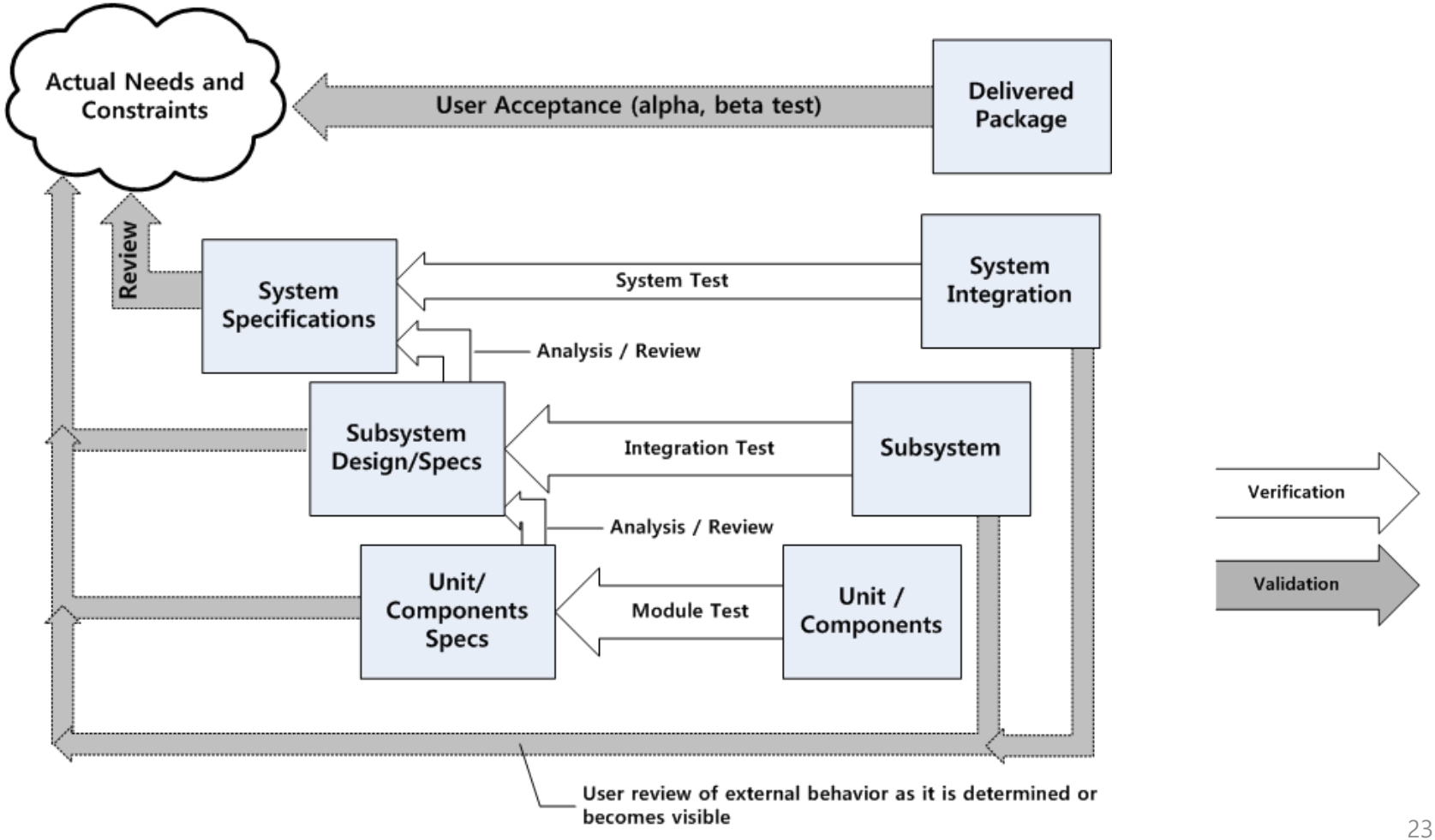


V&V Depends on the Specification

- Unverifiable (but validatable) specification: "If a user presses a request button at floor i, an available elevator must arrive at floor i soon."

- Verifiable specification: "If a user presses a request button at floor i, an available elevator must arrive at floor i within 30 seconds"

V-Model of V&V Activities in SDLC



V&V for Requirements Models

- **Verification**
  - *"Is the model well-formed?"*
  - *"Are the parts of the model consistent with one another?"*

- **Validation:**
  - **Animation** of the model on small examples is possible.
  - 'What if' questions:
    - Reasoning about the consequences of particular requirements;
    - Reasoning about the effect of possible changes
    - "Will the system ever do the following,"
  - State exploration
    - E.g., use **model checking** to find traces that satisfy some property

- Generation techniques for requirements validation
  - **Prototyping (Simulation)**
  - **Test-case generation**
  - **Review**

Reviews, Walkthroughs, Inspections

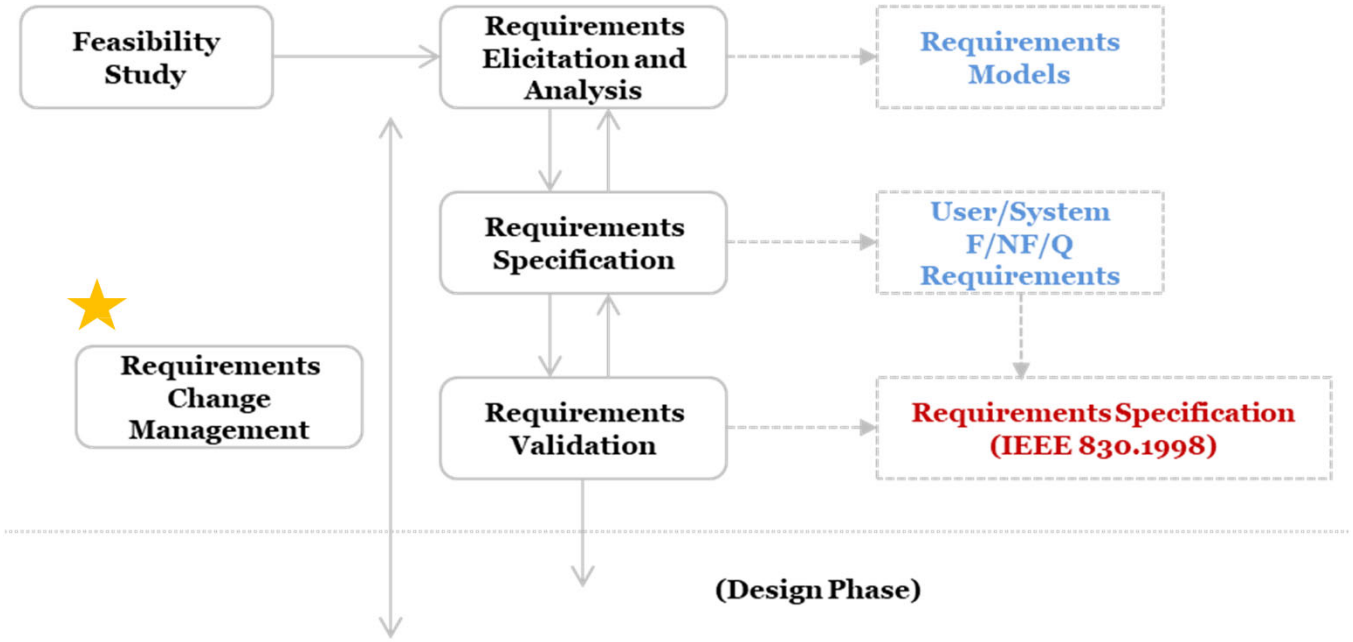
- **Management Reviews**
  - Preliminary design review (PDR), critical design review (CDR), formal technical review (FTR), formal business review (FBR), etc.
  - Used to provide confidence that the design is sound
  - Attended by management and sponsors (customers)

- **Walkthroughs**
  - Developer technique (usually informal)
  - Used by development teams to improve quality of product
  - Focusing on finding defects

- **(Fagan) Inspections**
  - A process management tool
  - Used to improve quality of the development process
  - Collect defect data to analyze the quality of the process
  - Written output is important

10. Requirements Change Management

Requirements Engineering Process

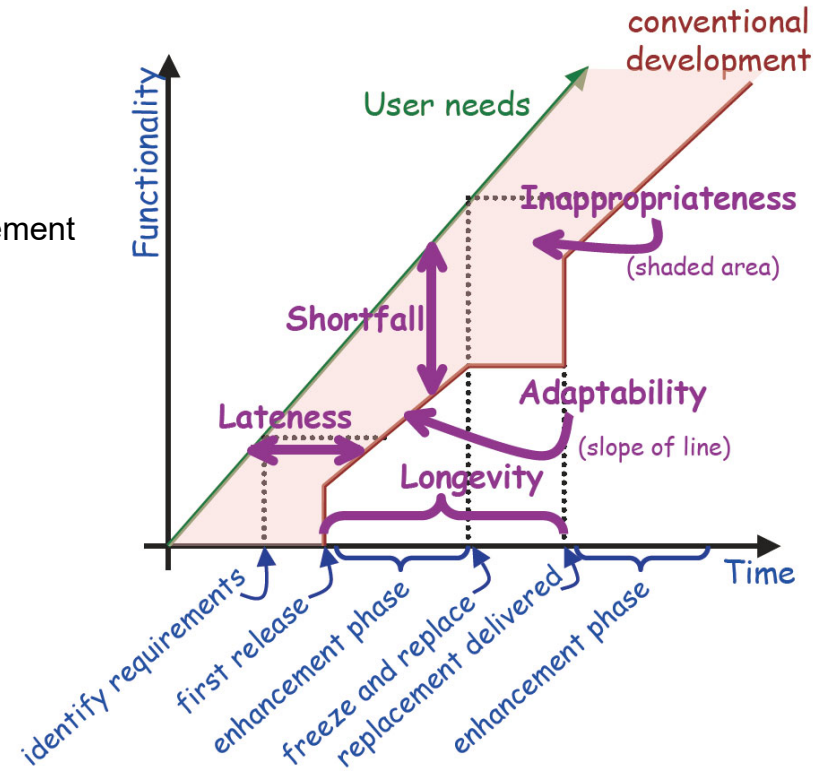


Laws of Program Evolution

- **Continuing Change**
  - Any software that reflects some external reality undergoes continual change or becomes progressively less useful
    - Change continues until it is judged more cost effective to replace the system
- **Increasing Complexity**
  - As software evolves, its complexity increases
- **Fundamental Law of Program Evolution**
  - Software evolution is self-regulating
    - With statistically determinable trends and invariants
- **Conservation of Organizational Stability**
  - During the active life of a software system, the work output of a development project is roughly constant, regardless of resources
- **Conservation of Familiarity**
  - The amount of change in successive releases is roughly constant

Requirements Growth Model

- **Davis's model (1988):**
  - User needs evolve continuously
    - May not be linear or continuous (hence no scale shown)
  - Traditional development always lags behind needs growth
    - First release implements only part of the original requirements
    - Functional enhancement adds new functionality
    - Eventually, further enhancement becomes too costly, and a replacement is planned
    - The replacement also only implements part of its requirements,
    - and so on...



Software Aging

- **Causes of Software Aging**
  - Failure to update the software to meet changing needs
    - Customers switch to a new product, if benefits outweigh switching costs
  - Changes to software tend to reduce its coherence

- **Costs of Software Aging**
  - Owners of aging software find it hard to keep up with the marketplace
  - Deterioration in space/time performance due to deteriorating structure
  - Aging software gets more buggy
    - Each “bug fix” introduces more errors than it fixes

- **Ways of Increasing longevity**
  - Design for change
    - Design patterns
    - Architecture styles
  - Document the software carefully
  - Requirements and designs should be reviewed by those responsible for its maintenance
  - Software Rejuvenation

Software Maintenance

- Maintenance philosophies
  - "Throw-it-over-the-wall" : someone else is responsible for maintenance
    - Investment in knowledge and experience is lost
    - Maintenance becomes a reverse engineering challenge
  - "Mission orientation" : development team make a long-term commitment to maintaining/enhancing the software

- **Basili's maintenance process models**:
  - **Quick-fix model**
    - Changes made at the code level, as easily as possible
    - Rapidly degrades the structure of the software
  - **Iterative enhancement model**
    - Changes made based on an analysis of the existing system
    - Attempts to control complexity and maintain good design
  - **Full-reuse model**
    - Starts with requirements for the new system, reusing as much as possible
    - Needs a mature reuse culture to be successful

Managing Requirements Change

- Managers need to respond to requirements change
  - Adding new requirements during development
  - Modifying requirements during development
  - Removing requirements during development

- **Key techniques**
  - Change Management (Process)
  - Release Planning
  - Requirements Prioritization
  - Requirements Traceability
  - Architectural Stability

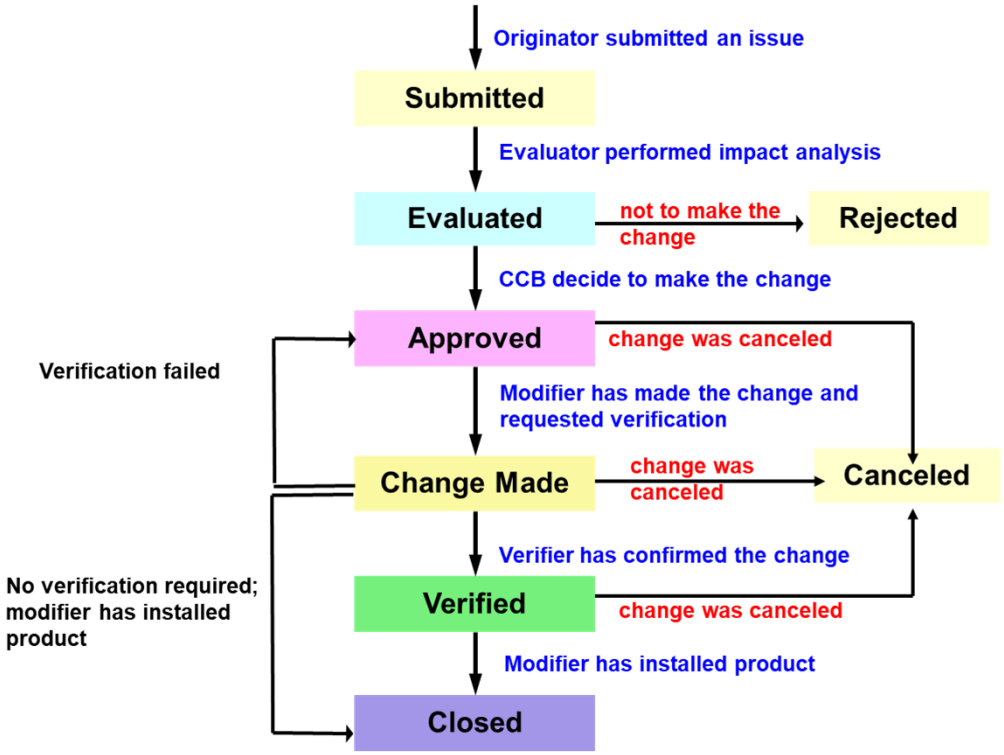
Change Management

- **Configuration Management**
  - Each distinct product is a **Configuration Item (CI)**
  - Each configuration item is placed under version control
  - Control which version of each CI belongs to which build of the system

- **Baseline**
  - A stable version of a document or system
    - Safe to share among the team
  - Formal approval process for changes should be incorporated into the next baseline

Change Management Process

- **Change Management Process**
  - All proposed changes are submitted formally as change requests
  - A review board reviews these periodically and decides which to accept



Requirements Traceability

- **From IEEE-STD-830.1998:**
  - **Backward traceability**
    - To previous stages of development
    - The origin of each requirement should be clear
  - **Forward traceability**
    - To all documents spawned by the SRS
    - Facilitation of referencing of each requirement in future documentation

- **From DOD-STD-2167A:**
  - A requirements specification is traceable if:
    1) It contains or implements all applicable stipulations in predecessor document
    2) A given term, acronym, or abbreviation means the same thing in all documents
    3) A given item or concept is referred to by the same name in the documents
    4) All material in the successor document has its basis in the predecessor document, that is, no untraceable material has been introduced
    5) The two documents do not contradict one another

Traceability Difficulties

- **Cost**
  - Very little automated support
  - Full traceability is very expensive and time-consuming
- **Delayed gratification**
  - The people defining traceability links are not the people who benefit from it
    - Development vs. V&V
  - Much of the benefit comes late in the lifecycle
    - Testing, integration, maintenance
- **Size and diversity**
  - Huge range of different document types, tools, decisions and responsibilities
  - No common schema exists for classifying and cataloging these
  - In practice, traceability concentrates only on baselined requirements

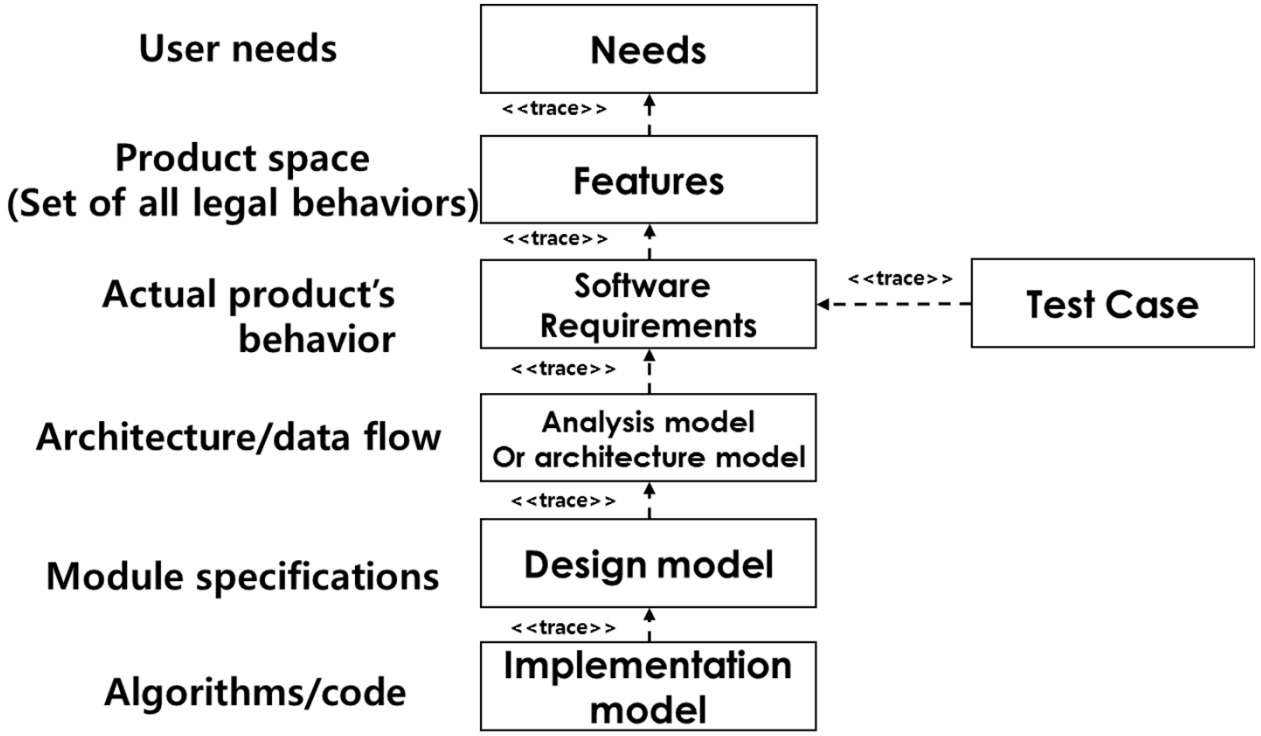
Traceability in Practice

- **Coverage**
  - Forward: Links from requirements forward to designs, code, test cases,
  - Backward: Links back from designs, code, test cases to requirements
  - links between requirements at different levels

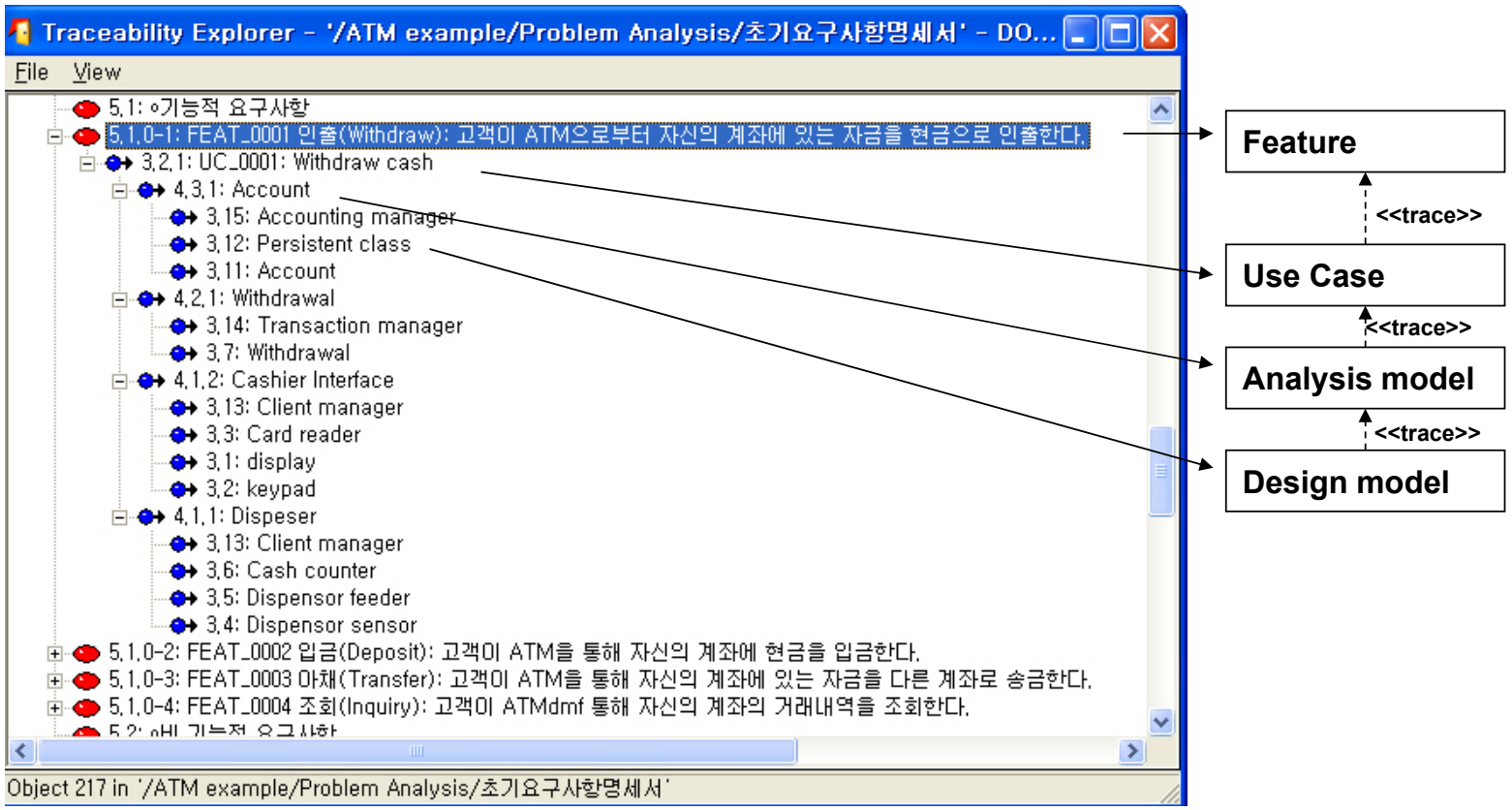
- **Traceability process**
  - Assign each sentence or paragraph a unique id number
  - Manually identify linkages
  - Use manual tables to record linkages in a document
  - Use a traceability tool (database) for project wide traceability
  - Tool then offers ability to
    - Follow links
    - Find missing links
    - Measure overall traceability

Example : Requirements Traceability

- When a high-level artifact derives a refined artifact, **Traceability link** should be generated between two artifacts.

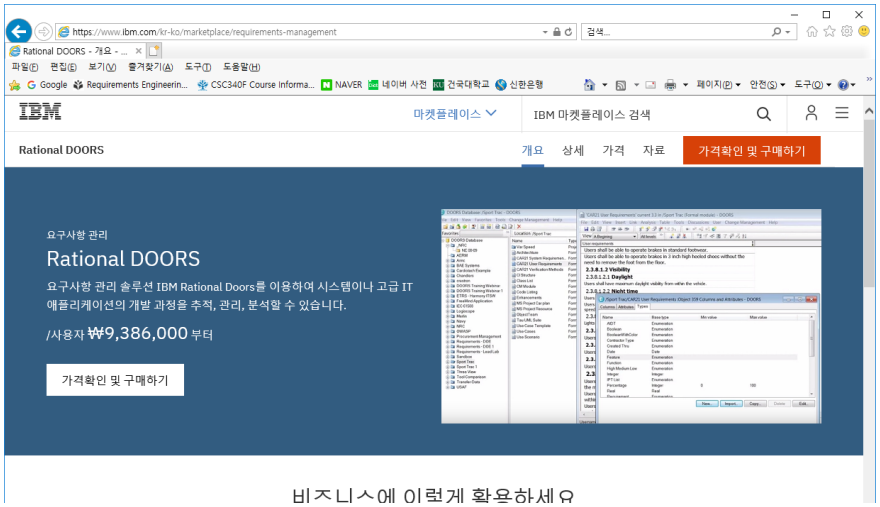


- Traceability link in DOORS

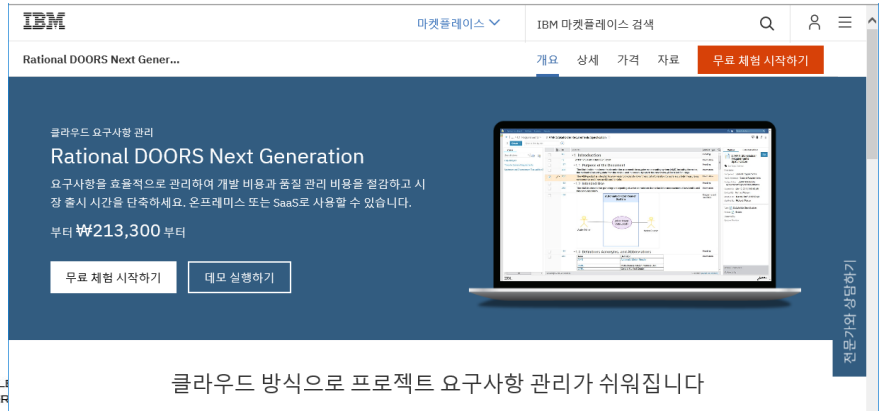


Requirements Management Tools

- **IBM Rational DOORS**

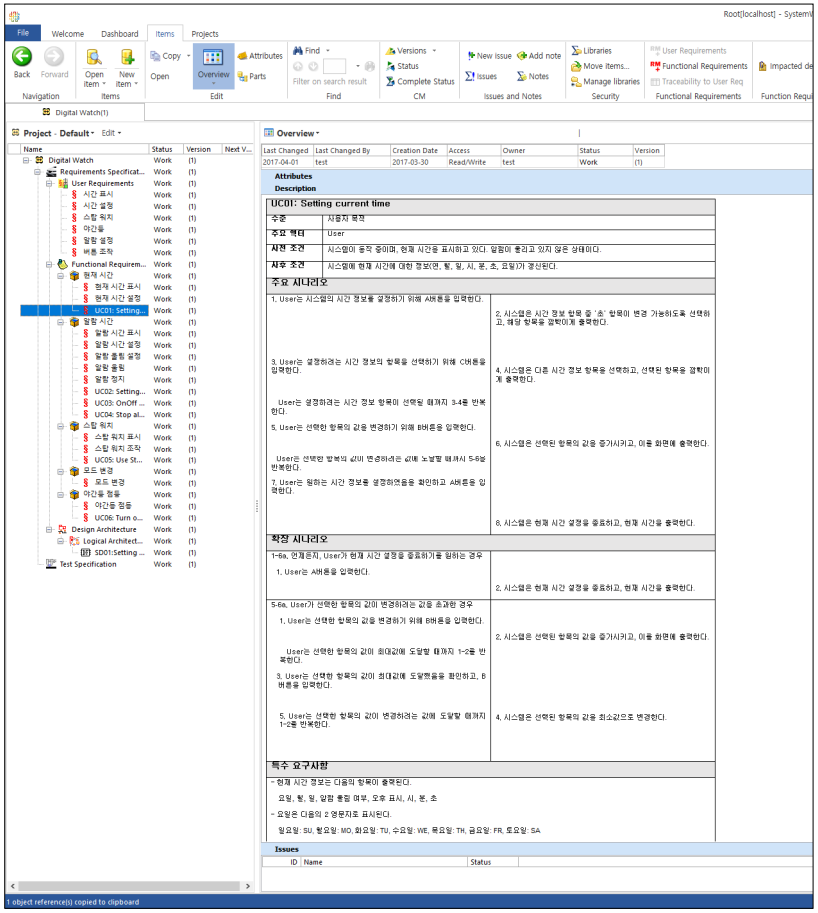


비즈니스에 이렇게 활용하세요



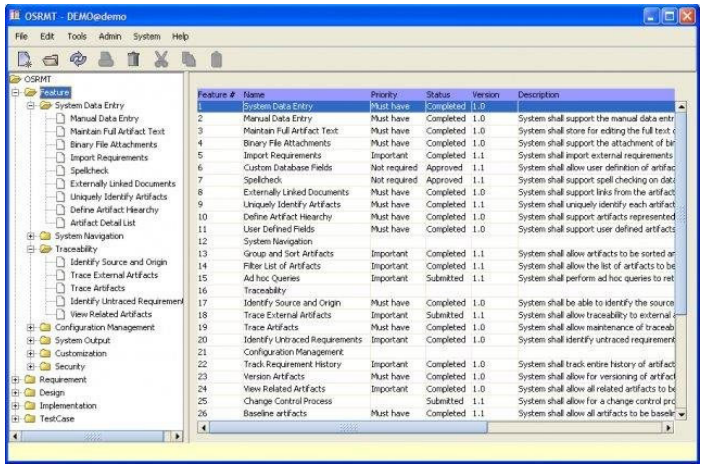
클라우드 방식으로 프로젝트 요구사항 관리가 쉬워집니다

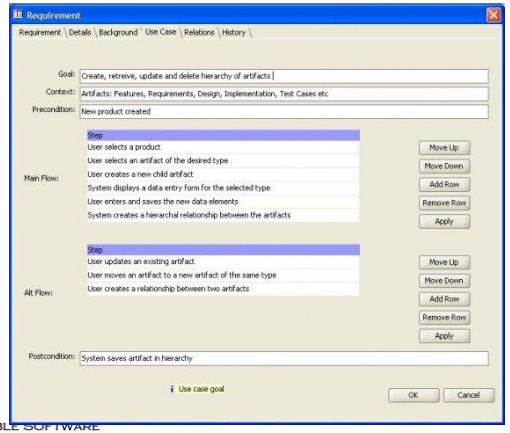
- **ESG PRACTICA RM+**



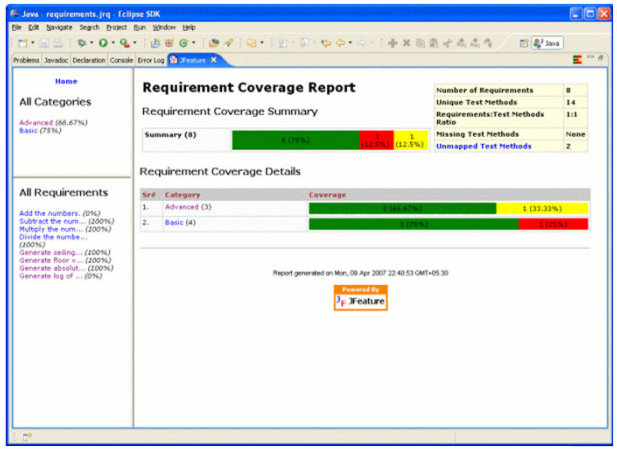
Requirements Management Tools

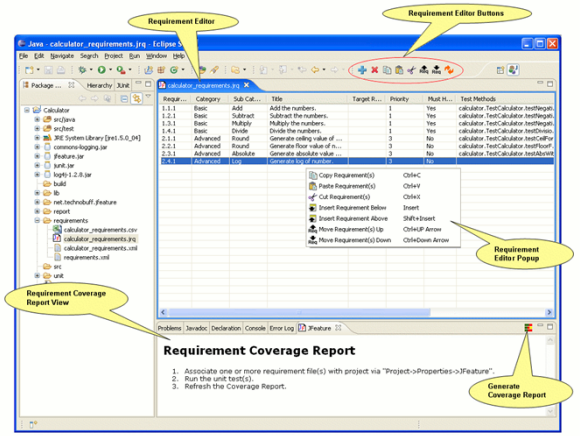
- **OSRMT**





- **JFeatures**



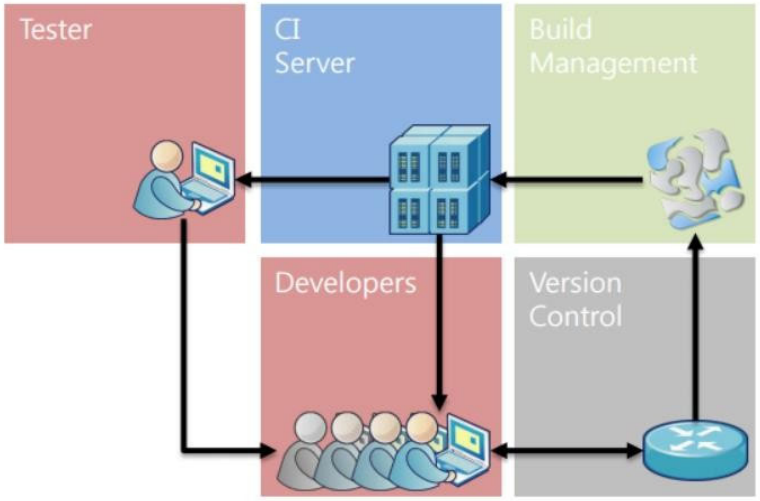


CTIP

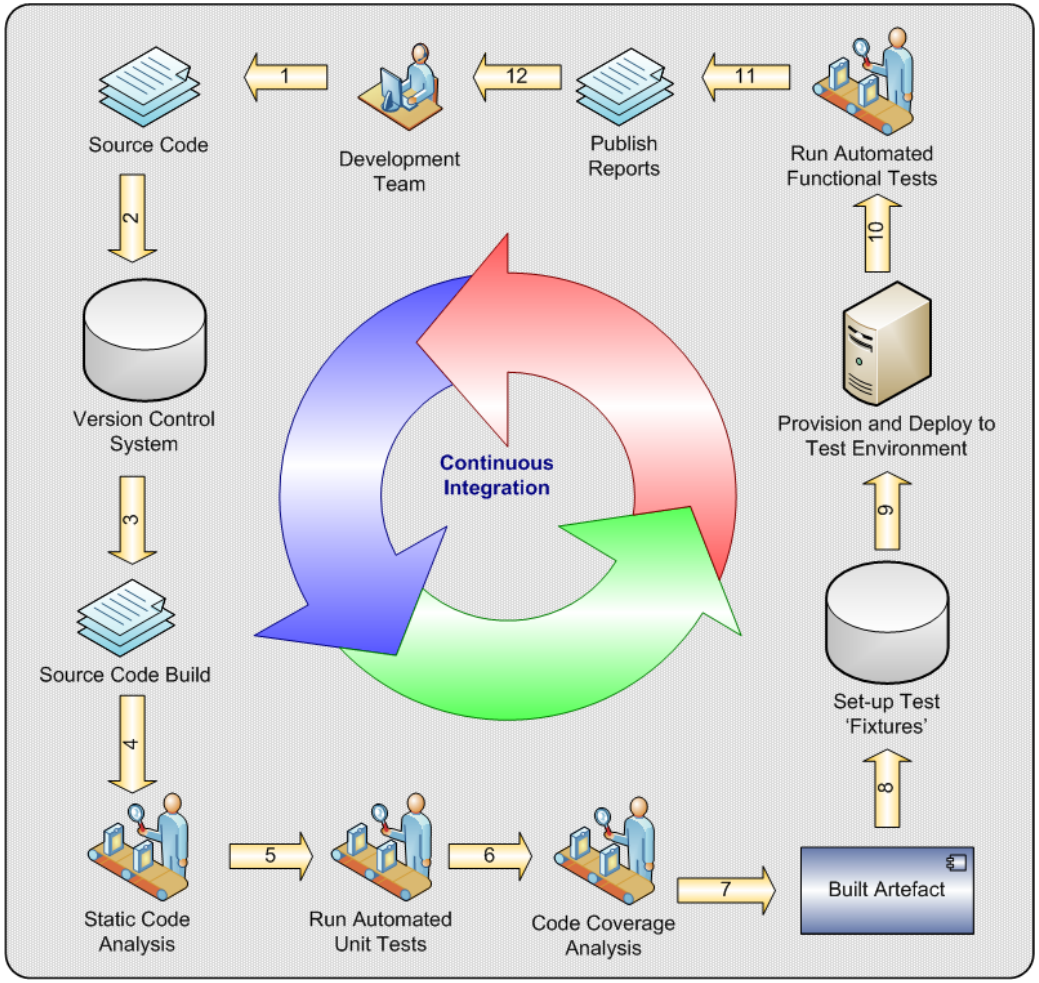
- **Continuous Integration (CI)**
  - A software development practice where members of a team integrate their work frequently, usually each person integrates at least daily, leading to multiple integrations per day.
  - Each integration is verified by an automated build (**including test**) to detect integration errors as quickly as possible.

- **Continuous test & integration platform (CTIP)**
  - Continuous integration + continuous test

- CTIP consists of several (semi-)automatic tools
  - Continuous integration management
  - Version control
  - Build automation
  - Issue tracking (communication)
  - Static analysis
  - Testing tool (automation, management)
  - Etc.



CTIP Process

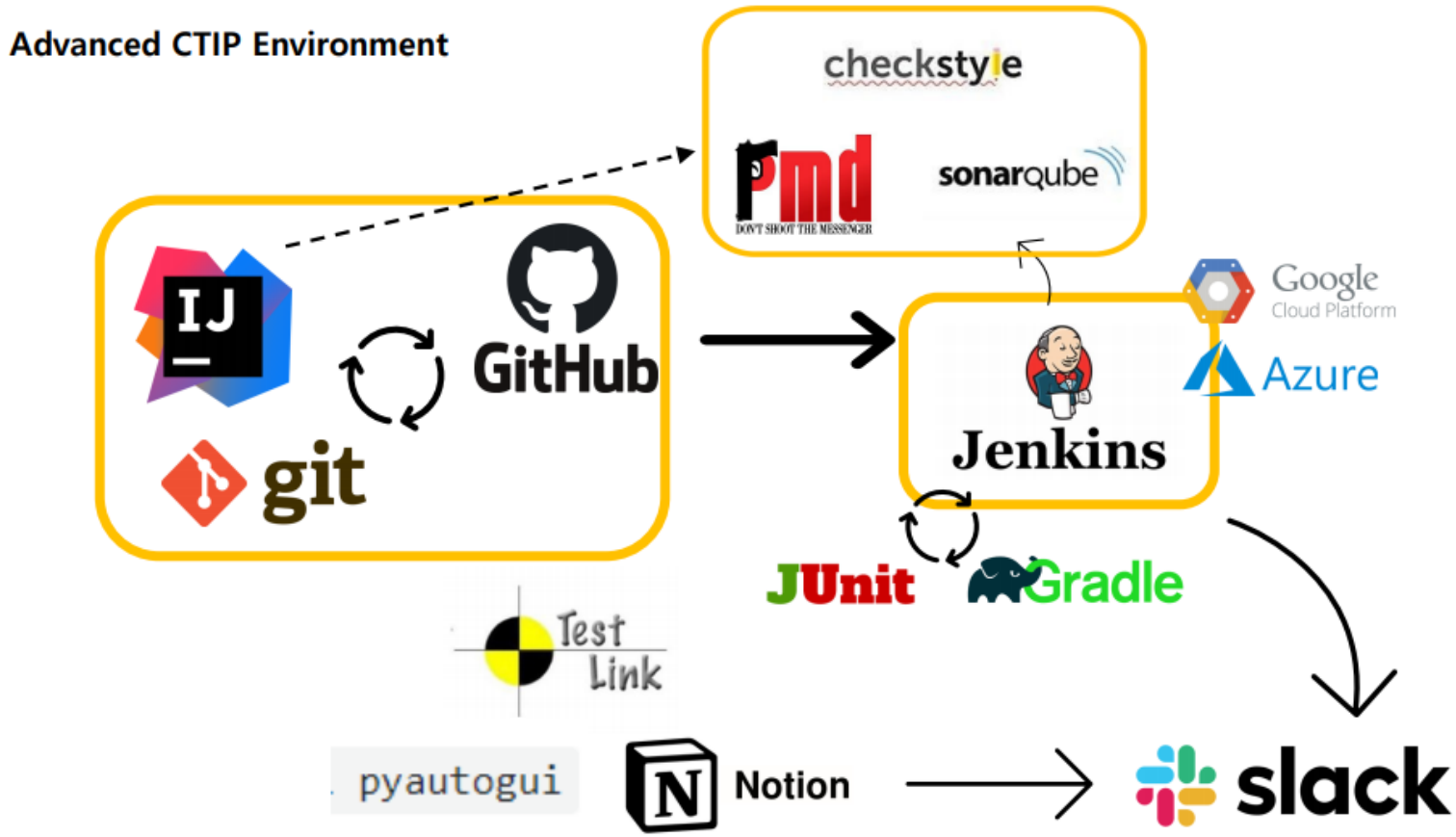


Recent CI Tools (2022)

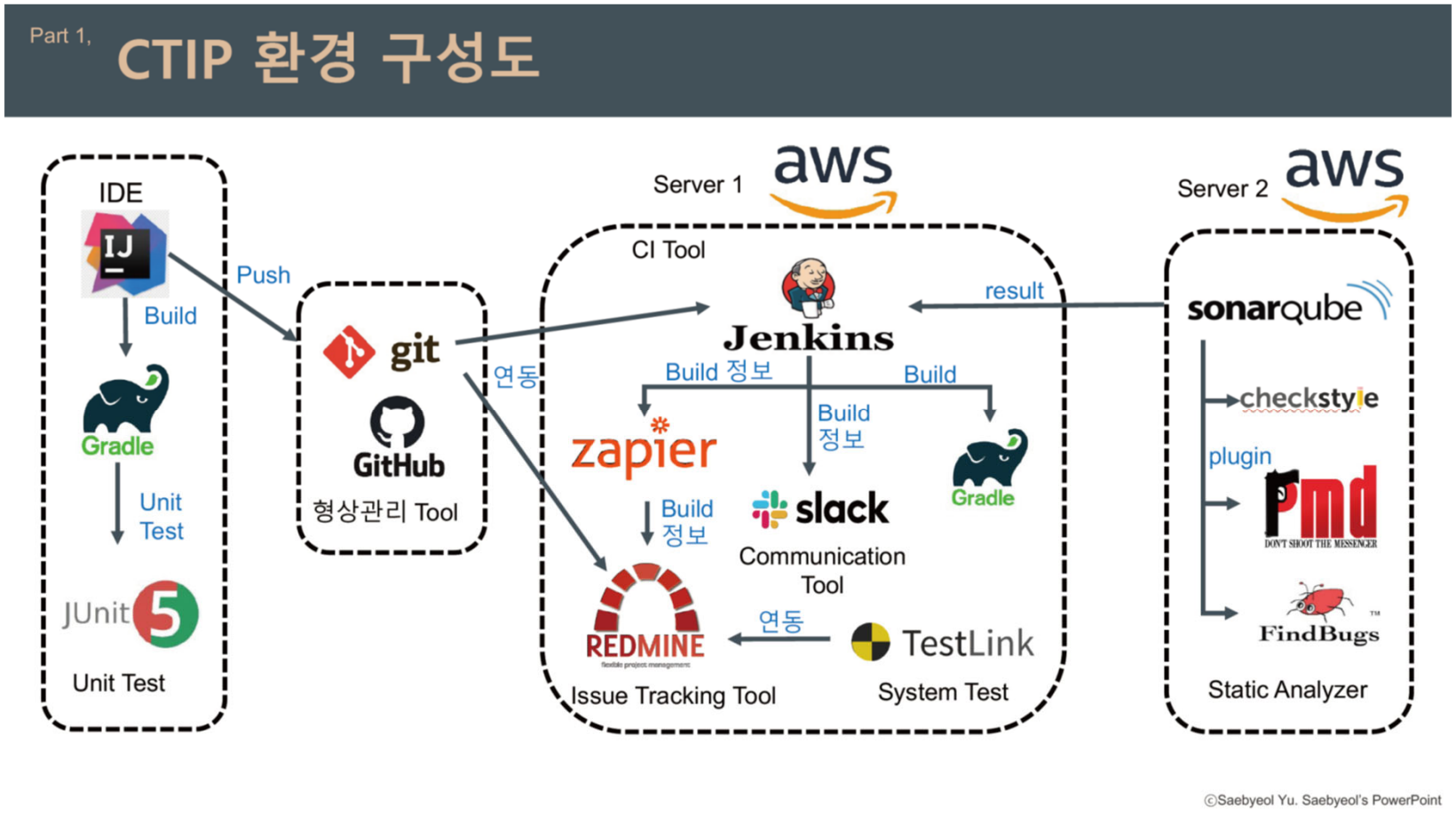
	Jenkins	circleci	TeamCity	Bamboo	GitLab
Open source	Yes	No	No	No	No
Ease of use & setup	Medium	Medium	Medium	Medium	Medium
Built-in features	3/5	4/5	4/5	4/5	4/5
Integration	★ ★ ★ ★ ★	★ ★ ★	★ ★ ★ ★	★ ★ ★	★ ★ ★ ★
Hosting	On premise & Cloud	On premise & Cloud	On premise	On premise & Bitbucket as Cloud	On premise & Cloud
Free version	Yes	Yes	Yes	Yes	Yes
Build Agent License Pricing	Free	From $39 per month	From $299 one-off payment	From $10 one-off payment	From $4 per month per user
Supported OSs	Windows, Linux, macOS, Unix-like OS	Linux or MacOS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux distributions: Ubuntu, Debian, CentOS, Oracle Linux

https://www.katalon.com/resources-center/blog/ci-cd-tools/

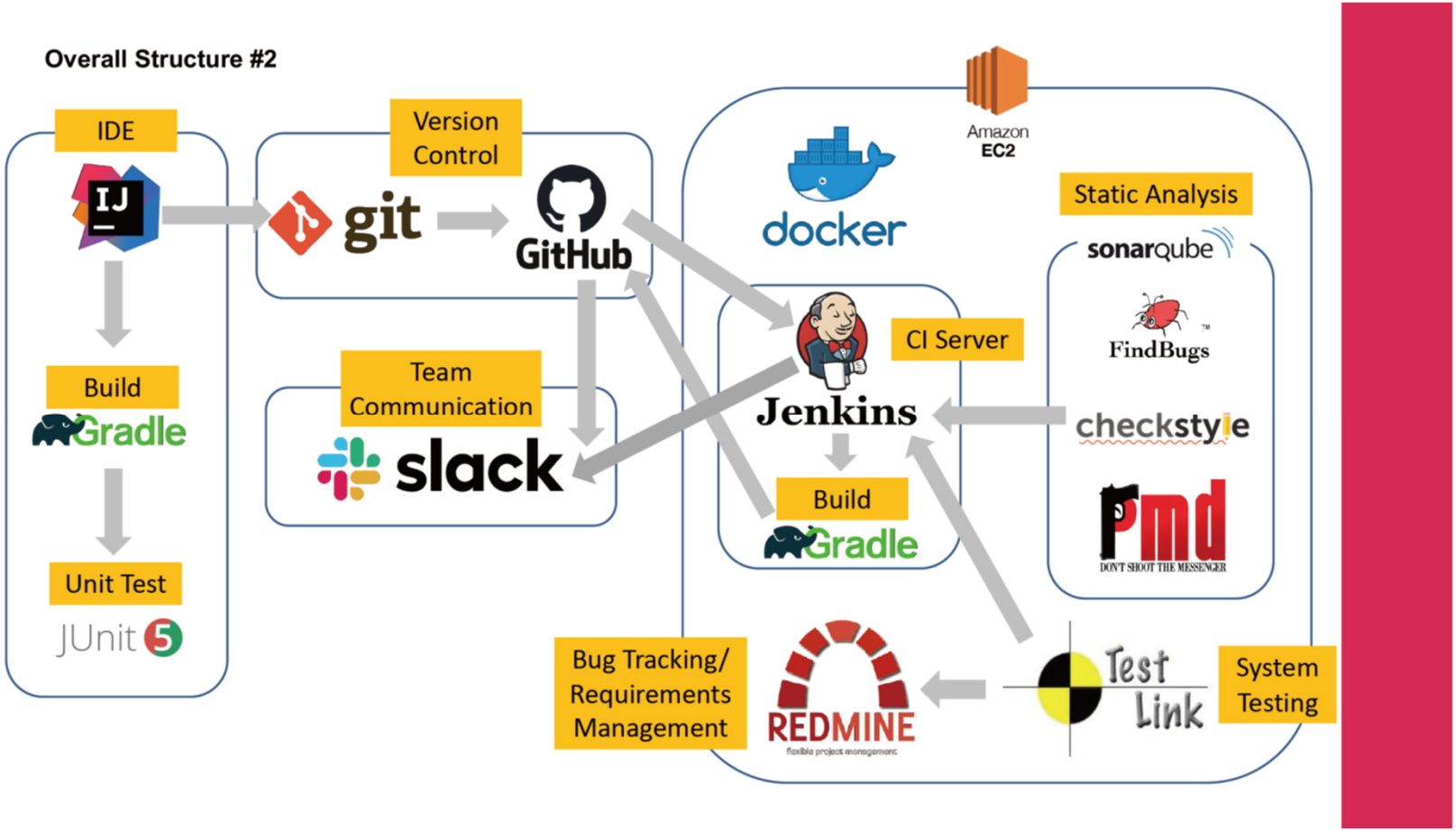
CTIP Examples (2021)



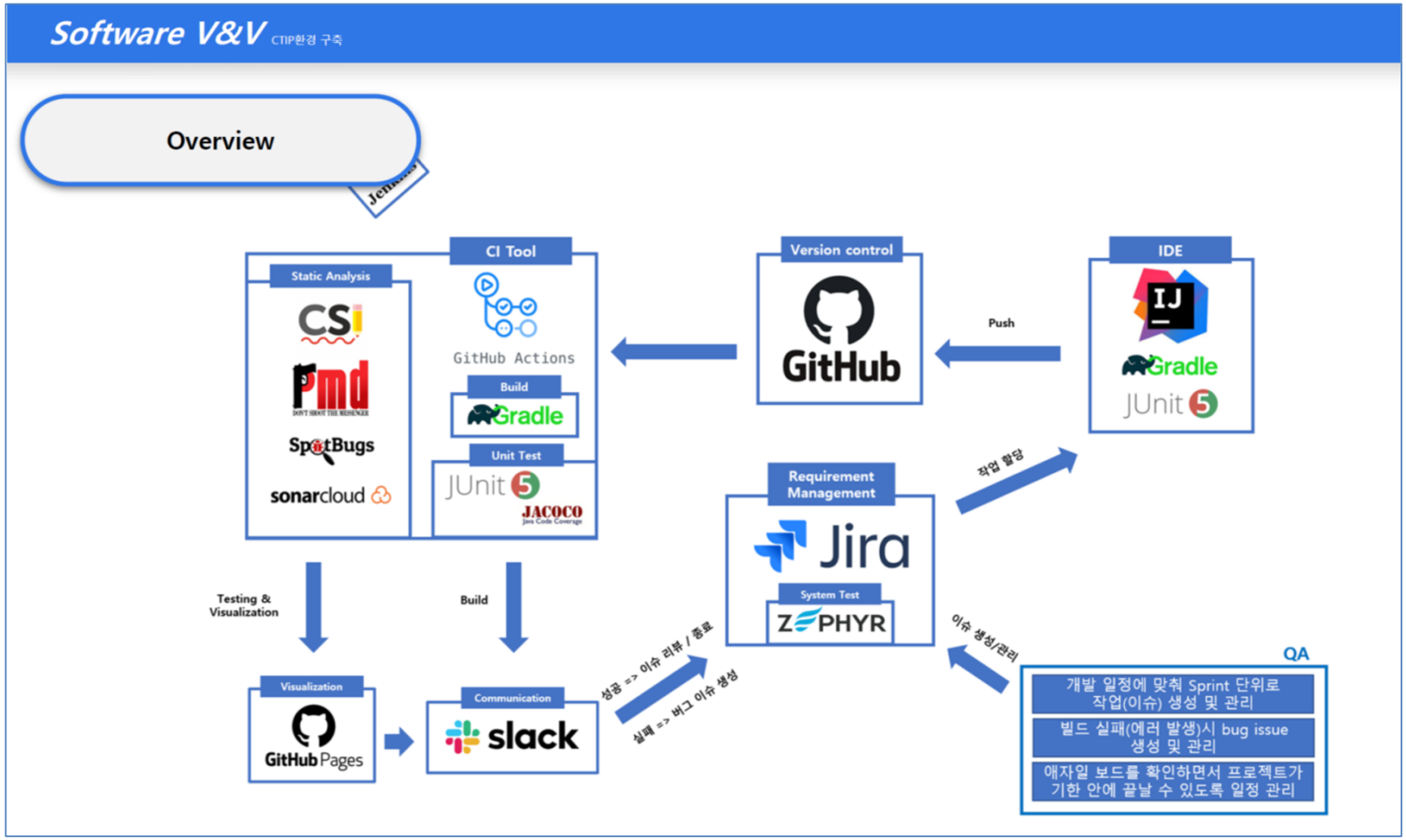
CTIP Examples (2021)



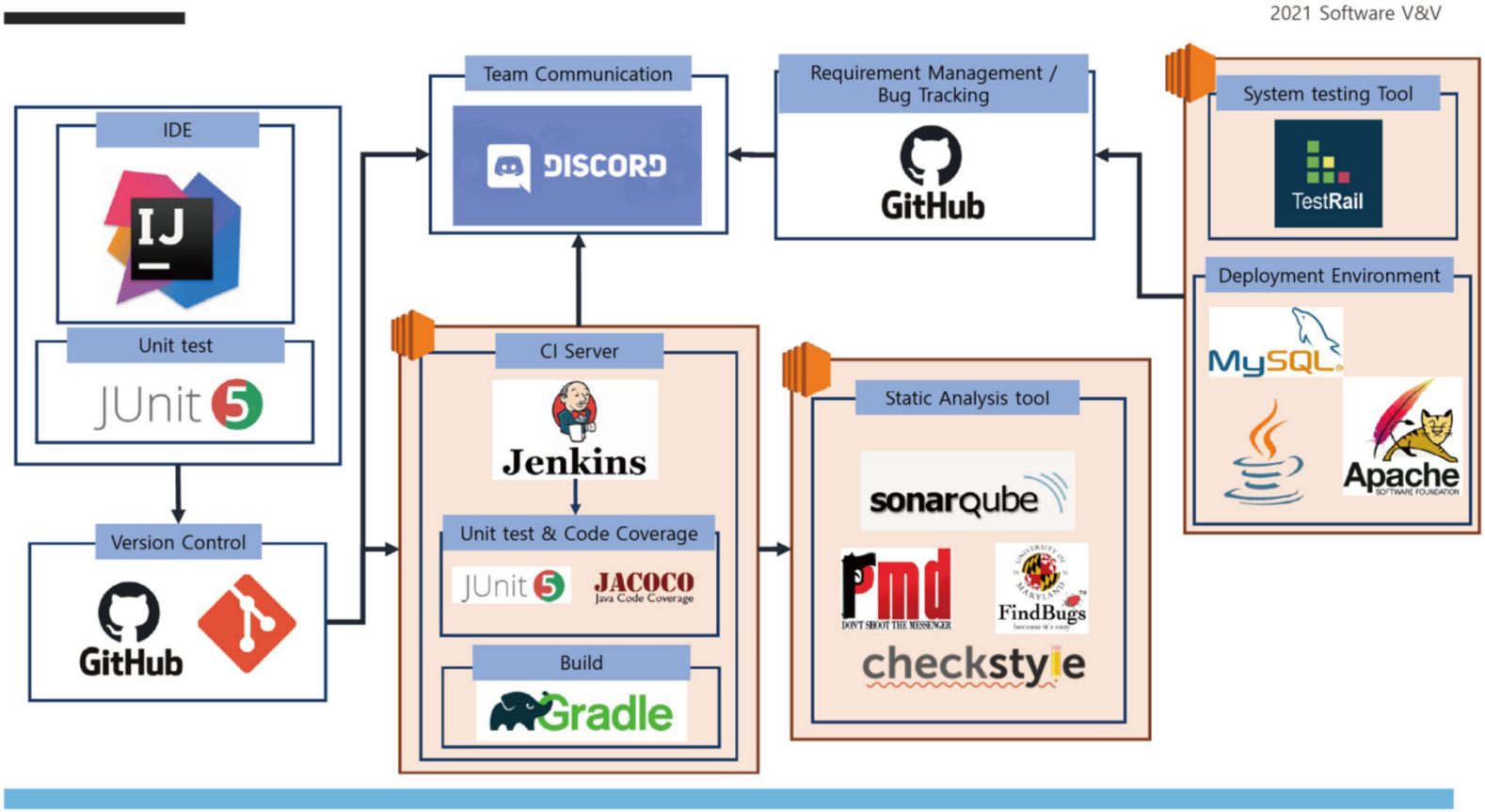
CTIP Examples (2021)



CTIP Examples (2021)



CTIP Examples (2021)



Summary

Requirements Engineering Process