

TOPCIT
ESSENCE

TOPCIT ESSENCE Technical Field 01 Software Development

TOPCIT

ESSENCE Ver.2

Technical Field
01 Software Development



TOPCIT

ESSENCE Ver.2

Technical Field

01 Software Development



TOPCIT ESSENCE is published to provide learning materials for TOPCIT examinees.

The TOPCIT Division desires the TOPCIT examinees who want to acquire the necessary practical competency in the field of ICT to exploit as self-directed learning materials.

For more information about TOPCIT ESSENCE, visit TOPCIT website or send us an e-mail.

As part of the **TOPCIT ESSENCE** contents feed into authors' personal opinions, it is not the TOPCIT Division's official stance.

Ministry of Science, ICT and Future Planning
Institute for Information and Communications Technology Promotion
Korea Productivity Center

Publisher TOPCIT Division
+82-2-398-7649 www.topcit.or.kr/en helpdesk@topcit.or.kr

Date of Publication **1st Edition** 2014. 12. 10
2nd Edition 2016. 2. 26

Copyright © Ministry of Science, ICT and Future Planning
All rights reserved.

No part of this book may be used or reproduced in any manner whatever without written permission.

TOPCIT

ESSENCE Ver.2

Technical Field

01 Software Development

TOPCIT

ESSENCE Ver.2



Technical Field

01 Software Development

CONTENTS

Software Development and Management	18		
01 Characteristics of Software	20		
Differences with Hardware	20		
Characteristics of Software	20		
Four Critical Points in Software Engineering	21		
Lifecycle of Software	21		
02 Introduction of Software Development	22		
Software Development	22		
Software – based Technologies	26		
01 Data Structure	28		
Definition	28		
Classification	28		
Selection Criteria for a Data Structure	28		
Utilization of a Data Structure	28		
02 Algorithms	29		
Definition of Algorithms	29		
Criteria for Algorithm Analysis	29		
Analysis of Algorithm Performance	30		
Algorithms for Sort and Search	31		
03 Operating System (OS)	33		
Concept of Process	33		
Thread	34		
		Concurrent Process	35
		Deadlock	35
		Scheduling	36
		Virtual Memory	37
		File System	39
		I/O System	39
		Latest OS Technologies	40
		04 Computer Structure	40
		Components of a Computer	40
		Hierarchy Structure and Mechanism of a Memory Device	41
		Latest Technologies and Trends	42
		Analysis and Specification of Requirements	48
		01 Analysis of Software Requirements	50
		Outline of Requirement Analysis	50
		Elicitation Methods of Functional and Non-functional Requirements	51
		02 Modeling	52
		What is Modeling?	52
		Three Viewpoints in Modeling	52
		03 Structured Analysis	53
		Data Flow Diagram (DFD)	53
		Mini-Specification	54
		Data Dictionary	54

CONTENTS

04 Object–Oriented Analysis	54
Use Case	55
Information Modeling	55
Dynamic Modeling	55
Functional Modeling	56
05 Requirement Specification	56
 Principles of Software Design	 58
01 Principles of Software Design	60
Division	60
Abstraction	60
Information Hiding	60
Stepwise refinement	61
Modularization	61
Structuralization	61
02 Cohesion and Coupling	62
Cohesion	62
Coupling	62
03 Structured Design Techniques	63
Transform Flow–oriented Design	63
Transaction Flow–oriented Design	65

Software Architecture Design	67
01 Software Architecture Design	69
Outline of Software Architecture	69
Procedure of designing software architecture	69
02 Types of Architectures	70
Repository Structure	70
MVC (Model – View– Controller) Structure	70
Client–Server Model	70
Hierarchical Structure	70
03 Methods of Architecture Design Expression	71
Context Model	71
Component Diagram	71
Package Diagram	71
 Object–oriented Design Process	 73
01 Concept and Principle of the Object–oriented Design	75
Objects and Classes	75
Encapsulation	75
Inheritance	76
Polymorphism	76
02 Static Modeling and Dynamic Modeling	76
Static Modeling	76
Dynamic modeling	77

CONTENTS

03 Design Pattern	77		
Singleton Pattern	77		
Factory Method Pattern	78		
Façade Pattern	78		
Strategy Pattern	78		
 Design Concept in the User Interface	 81		
01 Design Concept and Principles of User Interface (UI)	83		
A Need for Consistency	83		
User-oriented Design	83		
Feedback	83		
Identification of Destructive Actions	83		
02 HCI (Human-Computer Interaction)	84		
Concept of HCI	84		
Types of HCI	84		
03 Components of the Graphic User Interface(GUI)	85		
 Programming Language & Code Reuse and Refactoring	 88		
01 Characteristics of Programming Languages	90		
Concept of Programming Languages	90		
Interpreter Language	90		
Compiler Languages	91		
 02 Characteristics of Major Programming Languages	 92		
		C (Programming Language)	92
		C++	92
		Java	93
		Node.js	93
		03 Code Reuse and Refactoring	94
		Concept of Code Reuse and Refactoring	94
		Key Refactoring Techniques	95
		 Software Testing	 97
		01 Concept and Process of Testing	98
		Concept of Testing	98
		Testing Process	98
		How to Design Test Cases	99
		02 Testing Types and Techniques	100
		Testing Types	100
		Testing Techniques	100
		03 Software Build and Distribution	102
		Software Build	102
		Software Distribution	102
		 Software Maintenance & Reverse Engineering and Re-engineering	 105
		01 Software Maintenance	106

CONTENTS

Definition of Software Maintenance	106
Purpose of Software Maintenance	106
Types of Software Maintenance	106
Procedure of Software Maintenance	107
Types of Software Maintenance Units	108
02 Reverse Engineering, Re-engineering, and Reuse	109
Software 3R	109
Outline of Reverse Engineering	110
Outline of Re-engineering	111
Outline of Software Reuse	112
Management of Software Requirements	114
01 Requirement Management	115
Definition of Requirement Management	115
Importance of Requirement Management	115
Purpose of Requirement Management	115
Process of Requirement Management	115
Principles in Requirement Management	116
02 Requirement Specification	116
Requirement Specification Techniques	116
03 Management of Requirement Changes and Traceability	118
Outline of Requirement Traceability	118
Software Configuration Management	121

01 Outline of Software Configuration Management	122
Definition of Software Configuration Management	122
02 Concept Map of Configuration Management and its Components	122
Concept Map of Configuration Management	122
03 Activities of Configuration Management	123
Activities of Configuration Management	123
Effects of Configuration Management	124
Considerations for Configuration Management	124
04 Configuration Management Tools	125
Configuration Management Tools	125
Subversion (SVN)	125
Distributed Version Control System (Git)	126
TFS (Team Foundation Server)	126
Software Quality Management	129
01 Software Quality Management	130
Definition of Software Quality Management	130
Purposes of Quality Management	130
Elements of Software Quality	130
02 Perspectives of Software Quality	131
Users' Perspective	131
Developers' Perspective	131
Managers' Perspective	131

CONTENTS

03 Characteristics of Software Quality and Major Software Quality Models	131
Characteristics of Software Quality	131
Key Software Quality Models	132
04 Methods of Software Quality Measurement	133
05 Software Quality Management	133
06 Activities for Software Quality Assurance	134
Techniques for Software Quality Assurance	134
Software Quality Assurance Procedure and Activities	135
Activities for Software Quality Control and Evaluation	136
 Company A – Taking a glimpse at a business report in setting up a next-generation system	 138
Business Outline	138
Project Background	138
Business Scope	138
Expected Results	139
Execution Strategies	139
Architecture Requirements	140
Requirements of Quality Attributes	140
 Agile Development	 143
01 Concept of Agile Development	145
Agile Background	145

Agile Concept	145
Characteristics of Agile	146
Types of the Agile Methodology	146
 02 Agile Development Methodology – XP	 147
XP Outline	147
Values of XP	147
Practices of XP	148
 03 Scrum	 148
Outline of Scrum	148
Scrum Process	149
Characteristics of Scrum	150
 Mobile Computing	 152
01 Outline of Mobile Computing	153
Characteristics of Mobile Computing	155
02 Mobile Computing Development Process	156
 Cloud Computing	 158
01 Definition of Cloud Computing	159
Cloud Computing vs. Other Types of Computing	160
02 Types of Cloud Computing	161
Classification Based on Service Types	161

Classification Based on Cloud Operation Forms	162
03 Server Virtualization Technology	162
Hypervisors	163
Types of Hypervisors	163
Types of Server Virtualization	164
04 Storage and Network Virtualization Technology	166
Storage Virtualization	166
Network Virtualization	166
 Software Product Line Engineering	 168
01 Outline of Software Product Line Engineering	169
02 Components of Software Product Line	170
03 Software Product Line Engineering Process	170
04 Advantages of Software Product Line Engineering	171

I Software Development and Management

►►► Latest Trends and Key Issues

According to the June 2015 release of Gartner, an American IT research consultancy, the global software market will reach USD 1,2692 trillion in 2014, the portion of software in the overall ICT market is going up every year, and the impact of software has skyrocketed in all industries as they have become 'smarter' and 'big data-based.' Therefore, high-quality software development in the smart IT environment has become a critical factor to determine the system quality.

►►► Study Objectives

- * Able to explain software characteristics and problems
- * Able to explain the background and purpose of software engineering
- * Able to explain software development process models

►►► Practical Importance High

►►► Keywords

- Software characteristics
- Software lifecycle
- Analyzing requirements, design, implementation, testing
- Managing software requirements, managing software maintenance, managing software configurations, managing software quality

+ Practical Tips Consequences of not understanding software development and management

It is often the case that 'software development' is misunderstood as simple 'coding.' However, software development is not just about coding with programming languages. Imagine that you are using blocks to build a house. The first step is to design the foundation and structure of the house, then map out the blocks on their form and characteristics. The house construction is complete after building up the blocks according to the design. What if there is no layout or design? A house with a simple form can be easily constructed. However, if the building blocks are not stacked correctly, the house has to be rebuilt from the beginning, wasting time and efforts. Even after that, there is still no guarantee that the house would be completed correctly. The same logic applies to software development. Developing good software requires a firm understanding of the requirements and knowledge of management methodologies for software development procedures, personnel and timeline. 'Coding' is simply a work for converting human descriptions into easily understandable content by computers, using a program language such as 'C' and 'JAVA.' Thus, coding does not refer to the entire software development.

The importance of software has increased over time as its applications encompass all fields including national defense, finance, communications and household electronics. However, since software is embedded in so many varieties of products, it is difficult to validate the internal content. Moreover, due to its invisibility, progress cannot be easily identified during software development. Therefore, it is very challenging to validate errors during and after development or to find relevant solutions.

In the past, software development projects could be carried out even without a systematic process but just with programmers' experience. However, large-scale projects today involve hundreds of developers and confront numerous issues: communication among programmers, timeline and cost management. This is all due to long development timelines and modifications of obscure and complicated requirements. Therefore, understanding of software development and management methodologies is essential to developing high-quality software.

01 Characteristics of Software

Differences with Hardware

While hardware is a physical system, software is a logical system. So when applying hardware engineering methodologies to software development and management, many issues and additional tasks arise.

Hardware vs. Software

- ① SW is easier to modify.
 - So, developers came to have a practice of pre-coding and post-editing.
- ② SW does not wear or tear even if used for long, but its maintenance cost is higher than HW's.
 - SW reliability cannot be accurately estimated with a HW reliability model, so its maintenance is different from HW's in a different way.
- ③ It is difficult to know the development progress status due to the invisibility of SW.
 - Due to additional staffing to fulfill the SW development timeline, there have been frequent time delays.
- ④ It is difficult to define requirements for SW
 - When 30-page-long requirements are needed to purchase HW equipment of USD 5 million, approximately 150-page-long requirements are needed to purchase the SW with the same price.

- Software developers' creativity determines the development period and performance.
- The 'Law of Increasing Returns' is applied, inducing no additional costs even with additional production (copying) unlike hardware.

Characteristics of Software

- ① Competent human resources are the key to software.

The number of personnel is not in proportion with performance; the personnels' qualitative competency is more important.
- ② Software determines the cost competitiveness of devices.

Well-developed software can generate sufficient output of a product, even with inferior hardware (memories, AP and batteries), thus lowering the product cost.
- ③ Reusability is the key to software.
 - Software is an intangible asset where human beings' creativity and intellectual capacities are integrated, and is also easily copiable.
 - It can be infinitely reused without consuming tangible resources for the sake of additional production.
 - That is why there is a great interest in open source for software, allowing the usage of resources free of cost for anyone.

Four Critical Points in Software Engineering

Software engineering is defined as 'a discipline that systematically, descriptively and quantitatively covers the entire lifecycle of software, ranging from development, operation and maintenance.' Four critical points in software engineering are as follows, through which high-quality software can be produced and delivered with given cost and timeline.

- ① Methodologies
 - Methodologies consisting of project planning and estimation, system and software analysis, data structure, program structure, algorithm, coding, testing and maintenance management
 - Adopting special language orientation (e.g. object-oriented methodologies) or graph transcription methodologies
 - Adopting a series of assessment standards on software quality
- ② Tools
 - Referring to the automation or semi-automation of methodologies used for the sake of productivity or consistency upon performing a task
 - Presence of numerous tools throughout the software development lifecycle (e.g. requirement management tools, modeling tools, configuration management tools, and modification management tools)
 - When tools are integrated so that the information generated by a single tool can be used by other tools, it could be set up as a system to support software development.
- ③ Procedure
 - A procedure enables rational and timely development of software by combining methodologies and tools
 - A procedure defines the order for methodologies applied, output required (documents, reports, etc.), controls that guarantee quality and assist the adjustment of modifications, and milestones that help software managers to evaluate the progress.
- ④ People
 - In software engineering, many factors (establishment, improvement, maintenance, etc.) are operated by people and organizations, so there is a greater dependency on people.
 - Since more diverse issues arise compared with other engineering fields, it is realistically impossible to succinctly and logically align software development in an engineering sense.

Lifecycle of Software

- ① Definition
 - Referring to the whole process ranging from the understanding of the user environment to operation/maintenance
 - A general software lifecycle consists of the following activities: [feasibility review → development planning → requirement analysis → design → implementation → testing → operation → maintenance]

② Purpose

- Calculating the project cost and configuring a basic framework for development planning
- Standardizing terminologies
- Conducting project management

③ Selection of software lifecycle

- Important in tailoring the development process for a project in a company
- Based on risks and uncertainties in system development and understanding of them
- A model selected must be able to minimize risks/uncertainties inherent in a project.
- Representative lifecycle models include Waterfall Model, Prototype Model, Evolutionary Model and Incremental Model.

02 Introduction of Software Development

Activities of software development can be defined according to a software lifecycle.

Software Development

① Requirement analysis

- The biggest challenge in software development is to accurately decide on what to develop.
- Software requirement analysis is to understand users' requirements as the first step.
- It is a critical step to reduce development cost in the entire development process.
- Once investment is made for analyzing, defining and managing requirements in the initial stage, the overall software development period can be shortened and excess of cost and quality reduction can be prevented.

② Design

- While the software requirement analysis process is a conceptual stage, the design process is the first step for physical realization.
- System design determines a system structure consisting of sub-systems, which are allocated to components including hardware and software.
- Design makes a direct impact on quality.
- Poor design lowers stability in a system.
- Doing proper maintenance for an unstable system is difficult.

③ Implementation

- The goal in the software implementation stage is to do programming to fulfill requirements based on a design specification.
- A program must be coded to comply with a specific design or a user's manual.

- One of the most critical tasks in the implementation stage is to set a coding standard, based on which clear coding is done.

④ Testing

- Referring to a whole process of mobilizing manual or automatic methodologies to test and evaluate if a system fulfills designated requirements and if there is any difference between expectations and actual outcome
- A series of work to detect defects in order to secure software quality as the last stage for software quality assurance
- Including the modification task for quality evaluation and enhancement of the developed software

Software Management

Software management activities include the following: defining activities to support software development, responding to problems with software operation and requests for improvement, securing traceability, integrity and project visibility among output generated through the development and enabling systematic operation of software development activities.

① Software maintenance management

- Unlike other systems, requiring many modifications during and after the development
- Developing software to allow such continuous modifications
- Software maintenance management refers to activities of adapting many modifications that occur in the course of using software after it has been delivered – a process to prepare against modifications.

② Software requirement management

- Software design modifications occur more frequently than hardware's since software is intangible and incorrect preconceptions about software requirement management.
- Software requirement management is a system-based activity that extracts, configures and documents requirements from project-related stakeholders, and sets and manages consent for modifications.
- Purpose of requirement management
 - Communication: communication on what to do and why, and what has been modified
 - Collaboration: a means of collaboration for joint execution of tasks
 - Verification: verification to see if everything has been completed as planned

③ Software configuration management

- Activities to create configuration for a system by compiling plans on various outputs (documents, source codes) generated in the course of software development and maintenance, and to systematically manage and control the modifications
- A procedure to control modifications in the course of software development and maintenance is critical to manage the output of software development and obtain high quality software.

④ Software quality management

- Activities to verify if software development activities are aligned with a project plan and suitable for organizational policies
 - Software quality to be defined as the extent of software attributes that can fulfill the requirements
 - Also defined as overall characteristics related to capabilities of software products or services to fulfill explicit or implicit demands
- Characteristics of software quality are mostly functionality, reliability, usability, efficiency, maintenance and portability. Due to diverse types of software and their different usages, it is difficult to classify software quality characteristics in a universal and consistent manner.
 - Functionality: Completeness/Accuracy/Interoperability in implementing features, security, standard compliance, etc.
 - Reliability: Operational stability, ease-of-use in failure recovery, service continuity, data recoverability, etc.
 - Usability: Ease-of-use in functional learning, understanding of input/output (I/O) data, possibilities/consistency to adjust user interface, etc.
 - Efficiency: Response time, resource utilization rate, throughput, etc.
 - Maintenance: Problem diagnosis/solving, possible to modify set-up category selection, ease-of-use in updating, etc.
 - Portability: Conformity with the operating environment, ease-of-use in uninstalling, backward compatibility, etc.
- Depending on the needs, suitable quality characteristics must be selected and used, and accordingly, appropriate verification activities (e.g. review, inspection, testing, simulation) must be conducted.

Example Question

Question type

Descriptive question

Question

Describe the characteristics of the Spiral Model on software lifecycle and its execution stage, and compare it with the Agile Technique.

Answer and explanation

1) Characteristics of the Spiral Model

- It is a lifecycle model where risk management is the key. It uses the incremental technique to finally develop a software by continuously developing a prototype.
- It is the most realistic approach in establishing a massive system with a high financial and technical risk burden where moderate investment based on performance could reduce risks.
- Project development cost and schedule management are critical, especially risk analysis.
- The model itself is complicated, and it has not been fully verified with a new model.

2) Execution Stages for the Spiral Model

- Planning → Risk Analysis → Engineering → Customer Evaluation

3) Comparison with the Agile Model

- The Spiral Model is a document-based development methodology since it is less document-oriented, while the Agile Model is code-oriented methodology for actual coding.
- The Spiral Model aims to manage and minimize risks that might occur in executing a project, while the Agile Model embraces changes without quality reduction, emphasizes collaboration and 'fast product delivery' as a repetitive methodology.
- The Spiral Model is a scalable form starting off from the center along with spiral execution and repetition, so is suitable for large software, while the Agile Model is appropriate for software in small and medium units.

Related E-learning Contents

- Lesson 1 Introduction to Software

II

Software – based Technologies

►►► Latest Trends and Key Issues

Computers are becoming smaller, higher in performance and lower in price. They are used in every aspect of our life. A massive amount of data used in the rapidly changing IT environment along with the emergence of smart devices have a short lifecycle and forms including videos vary. Therefore, such new technologies emerge to utilize and manage the data as big data, cloud computing, mobile OS and parallel system.

►►► Study Objectives

- * Able to explain the definition and classification of data structure and utilize a linear/non-linear structure
- * Able to understand the roles of algorithms, and select appropriate algorithms
- * Able to understand the concept and roles of OS, and explain process management methods, and management techniques of virtual memory
- * Able to understand the computer structure and mechanism, and explain memory hierarchy structure and mechanism

►►► Practical Importance High

►►► Keywords

- Array, list, stack, deque, tree, graph
- Definition of algorithm, performance analysis of algorithm, sort/search algorithm
- Process, process control blocks, thread, parallel process, deadlock, scheduling, virtual memory, file system, etc.
- Storage device, multi-core, parallel system

+ Practical Tips What if software-based technologies are not understood?

In 2003, the northeastern part of the U.S. was hit hard by a blackout. The outage affected a wide swath of territory: seven U.S. states and one Canadian province. The main culprit behind it was the 'Race Condition (a type of deadlock)', a software failure within the XA/21 system that occurs in a multi-step process. Two programs within the XA/21 system simultaneously made the 'write access' to the same data structure, which caused the failure. This damaged the data structure, and as a result, an alarm process that was supposed to alert the failure failed to handle the alarm by falling into an infinite loop, even failing to inform of the failure of alarm handling. The alarm queues increased with no limit, ending up consuming all the available memories within 30 minutes. By this time, the main server stopped working, and the back-up server became the main server as a failover. However, the back-up server could not cope with the infinitely increasing queues, leading to its failure as well.

The above example is a case in point where software developed without proper understanding of software-based technologies generated critical failures, causing an enormous damage in people's life. Software-based technologies are essential to generate continuous performance for creative software development activities for the future. For instance, imagine that you build a house using blocks. You can build an optimal house only by selecting the most appropriate blocks according to the layout in terms of the material, color and shape, etc. However, more importantly, each block's material and characteristics must be fully understood as well as their various usages. If blocks are not used right in line with their characteristics, a house being completed is inevitably vulnerable. Software-based technologies are analogous to these blocks. Various base technologies are used that correspond to the rapid development of the information system. Therefore, optimized high-quality software can be applied to a given environment by applying most appropriate base technologies for software development.

Major software-based technologies include data structures, algorithms, OS and computer structures. Data structure is a critical technology in coding efficiency, helping you to easily decide on what data structure is more appropriate to implement problem-solving. An algorithm enables problems to be handled efficiently, considering throughput time and memory usage. The OS and computer structure consist of various characteristics to support efficient execution of systems: characteristics for the management of storage devices and efficient management of resources, and provisioning of libraries for convenient development of programs.

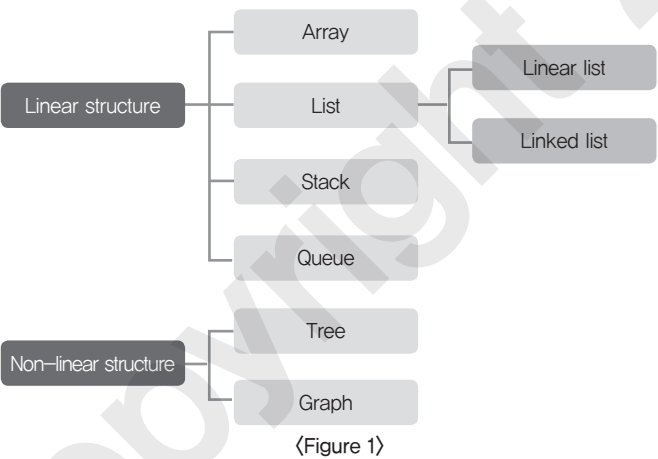
01 Data Structure

Definition

A data structure refers to organized and systematic classification and expression of data based on their characteristics and usages so that various data could be expressed and utilized more efficiently on a computer.

Classification

A data structure can be divided into a linear structure and a non-linear structure. A linear structure is a method of configuring data in a serial connection, while a non-linear structure is the one where the configuration of data is in a particular form of a hierarchical structure or a network structure.



Selection Criteria for a Data Structure

- ① Throughput time for data
- ② Size of data
- ③ Usage frequency of data
- ④ Extent of data renewal
- ⑤ Ease-of-use of a program

Utilization of a Data Structure

A data structure is mostly utilized in data sort and search, and file organization and index. Applications by data

structure are as follows:

- ① List: Implementation of array, DBMS index, such issues as search or sort, etc.
- ② Stack: Interrupt processing, control of the order for a recursive program, return address storage of sub-routines, calculating formulas for postfix notations, the undo feature in text editor, etc.
- ③ Queue: Job scheduling in OS, queue processing, asynchronous data exchange (file I/O, pipes, sockets), usage of keyboard buffers, spool management, etc.
- ④ Deque: Used in stack and queue-related areas as a data structure that takes advantage of stacks and queues
- ⑤ Tree: Issues involving search and sort, parsing of grammar, Huffman code, decision tree, game, etc.
- ⑥ Graph: Computer networks (local area network, Internet, web, etc.), analysis of electric circuits, binary relations, simultaneous equations, etc.

02 Algorithms

Definition of Algorithms

Algorithms refer to the description of a serial processing procedure by stage to solve given problems, and effective algorithms must fulfill the following conditions:

- ① Input & Output: For data needed to execute an algorithm, the number of input must be zero or higher, and after the execution, one or more output must be generated.
- ② Definiteness: Stages that indicate the content and the order of work to be executed must be definite without obscurity.
- ③ Finiteness: An algorithm must be closed after work execution.
- ④ Effectiveness: Processing in all stages must be clearly executable.

Criteria for Algorithm Analysis

- ① Correctness
It is to judge if an algorithm generates proper results within a definite time for feasible input.
- ② Amount of work done:
It refers to the number of times required to execute an algorithm. The amount of work done is measured only based on core operations in the flow, except for the general ones basically included in the entire algorithms.
- ③ Amount of space used:
It refers to the amount of a computer memory usage needed to store data and information while an algorithm is executed.

④ Optimality

The saying, “An algorithm is optimal” means that, considering the environment of a system where an algorithm is to be applied (amount of work done, amount of space used, etc.), there is no appropriate algorithm than this.

⑤ Simplicity or definiteness

It refers to how easily and definitely algorithm expressions have been written. If an algorithm is simple, proving the algorithm correctness, programming and debugging are more easily done.

Analysis of Algorithm Performance

Analysis of algorithm performance is to make general evaluation by estimating space complexity analyzing based on the space required for execution and time complexity analyzing based on the time required for execution.

① Space complexity

It refers to the total storage space needed ranging from executing an algorithm into a program and completing it. It is a sum of the amount of fixed space and the amount of variable space.

- Amount of fixed space: The amount of space needed in a fixed manner regardless of a program, the size of a program including variable and constant numbers and the number of I/O.
- Amount of variable space: A space to store data and variables used to execute a program, and a space to store information on executing features.

② Time complexity

It refers to the total time from executing to completing a program. It is a sum of the compile time and execution time.

- Compile time: It is a fixed amount of time not much related to program characteristics. It is consistently maintained once compiled unless there is a modification in a program.
- Execution time: It refers to a program executing time. Since it depends on the computer performance, it is calculated based on the frequency of executing commands instead of measuring the accurate execution time.

In comparing algorithms, execution time is used most of the time. It is indicated as $O(n)$ by using the Big Oh notation¹⁾ to be represented as time complexity. There are execution time functions such as $\log n$, n , $n \log n$, n^2 , n^3 and 2^n depending on the algorithms. It would be ideal to select an algorithm with the smallest time complexity by writing numerous algorithms for solving problems and calculating each of the execution time functions.

¹⁾ In the expression using the Big Oh notation, the execution frequency is calculated to find the execution time function. A term for 'n' that makes the greatest impact on this function value is selected and indicated on the right-side bracket of 'O' with the coefficient curtailed.

〈Table 1〉 Calculating the execution frequency depending on the changes in the value of 'n' in the execution time functions

$\log n$	\angle	n	\angle	$n \log n$	\angle	n^2	\angle	n^3	\angle	2^n
0		1		0		1		1		2
1		2		2		4		8		4
2		4		8		16		64		16
3		6		24		64		512		256
4		16		64		256		4096		65536
5		32		160		1024		32768		4294967296

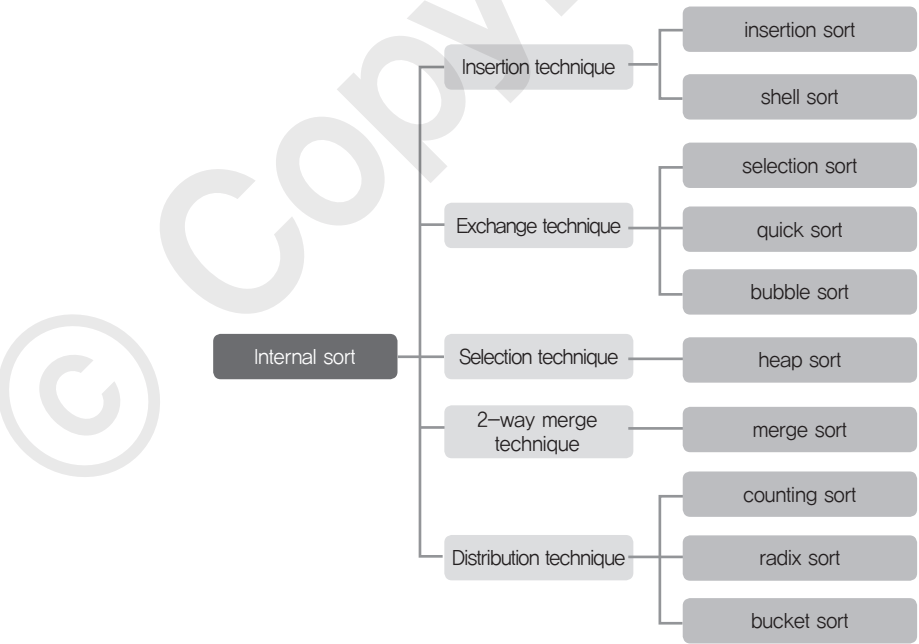
Algorithms for Sort and Search

① Classification of sort

A sort can be divided into an internal sort and an external sort depending on the sorting place. Sorting processes can be chosen based on such conditions as the characteristics of a system used, the quantity, the quantity and the status of data, and the required memory and execution time for sorting.

- Internal sort: Data to be sorted is small enough to be all held in the main memory. The sorting speed is high but the amount of data to be sorted by the capacity of the main memory is limited.
- External sort: Sorting is done in an auxiliary memory for large-scale data. Massive data can be divided into several sub-files and internally sorted. Each sub-file sorted within an auxiliary memory then is compiled, which slows down the speed.

② Types of internal sort algorithms



〈Figure 2〉 Types of internal sort

③ Comparison of the execution time for internal sort algorithms

〈Table 2〉 Comparison of the execution time for internal sort algorithms

Sort techniques	Description	Run time			Additional memory
		Worst	Average	Best	
Insertion sort	Assuming that data has been sorted, and the value is inserted and sorted on the location.	$O(n^2)$	$O(n^2)$	$O(n)$	Not available
Shell sort	Dividing the given data list into sub-files with lengths of particular parameter values, and executing insertion sort in each sub-file.	$O(n\log_2 n)$	$O(n^{1.5})$	$O(n^2)$	Not available
Selection sort	Sorting by repeating by the size (number) of data in finding and moving the minimum value to the left,	$O(n^2)$	$O(n^2)$	$O(n^2)$	Not available
Quick sort	Selecting a random standard using a sorting method designed in the way of 'divide and conquer', and then locating a value smaller than the standard to the left and a bigger one to the right. And then, selecting a random standard, and sorting to the left and right in turns repeatedly, using the recursive call.	$O(n^2)$	$O(n\log n)$	$O(n\log n)$	Not available
Bubble sort	Repeated sorting as exchanges of adjacent data repeatedly occur.	$O(n^2)$	$O(n^2)$	$O(n^2)$	Not available
Heap sort	Sorting by forming the largest heap tree or the smallest one.	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	Not available
Merge sort	Continuously dividing the data in half, sorting it and combining it again while using 'divide and conquer'.	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	Available
Radix sort	Sorting by comparing from the data with the smallest digit in the data.	$O(dn)$	$O(dn)$	$O(dn)$	Available

④ Search

It is a technique to efficiently find items in demand in a data set. Presence of data sorting would determine the technique into two types: linear search and control search. There is hashing to search data by calculating the key value depending on a function. Therefore, optimal search methods must be chosen by considering the array status of the form and data of a data structure. The following is a summary of a search algorithm.

〈Table 3〉 Classification of search algorithms

Classification Method	Data Sort	Type	Content and Characteristics
Linear search	x	Linear search	<ul style="list-style-type: none">• A method of search by comparing each record in order from the beginning till the end• Ease-of-use in writing a program• The bigger the file size, the higher the amount of seek time• The simplest and most straightforward search method• Number of mean comparisons: $(n+1)/2$, Mean search time: $O(n)$

Control search	o	Binary search	<ul style="list-style-type: none">• A method of search by continuously comparing the key and median after setting the upper value(F) and the lower value(L) and finding the medium(M)• Efficient because search is done by halving the number of files to be searched• As the number of records increases, it becomes more effective (in the worst scenario, the number of comparison is one time higher than the mean)• Mean search time: $O(\log_2 n)$
		Fibonacci search	<ul style="list-style-type: none">• A method of search by using a Fibonacci sequential and forming sub-files• While binary search uses division, Fibonacci search uses only addition and subtraction, which makes it faster.• Mean search time: $O(\log_2 n)$
		Interpolation search	<ul style="list-style-type: none">• A method of selecting and searching a spot where a search target is expected to be located, and later on the very spot, linear search is done.• Applicable for search in a dictionary, a telephone registry and an index, etc.• Mean performance of $O(\log(n))$
		Block search	<ul style="list-style-type: none">• A method of consecutively searching the key values within blocks where data to be searched for belong after classifying the entire data into a certain number of blocks• An efficient block size is \sqrt{n}• Easy to write and renew a program• Mean performance of $O(\log(n))$
	x	Binary tree search	<ul style="list-style-type: none">• A method of search using a binary tree• Mean performance of $O(\log(n))$ for input/search/deletion
A way of approaching using a particular function		Hashing	<ul style="list-style-type: none">• A method of search by calculating and finding an address where data are stored using a hashing function• Appropriate for data where input and deletion are frequent

03 Operating System (OS)

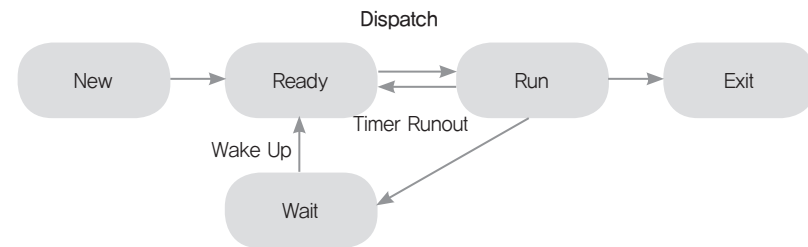
The OS is a system software that plays an intermediary role between computer hardware and application programs. The OS efficiently manages system resources including a processor, a memory, I/O devices and communications devices to support the execution of application of programs. Core components of OS are the process scheduler, memory manager, I/O manager, inter-process communications manager (IPC) and file system manager.

Concept of Process

A process is defined as 'a program that is running', 'a program with a Process Control Block (PCB)', or 'a unit of execution managed by the OS.' It can also be referred to as a 'job' or a 'task.'

① States of process

A process is equipped with a distinguished process status throughout its lifecycle.



〈Figure 3〉 Lifecycle of Process

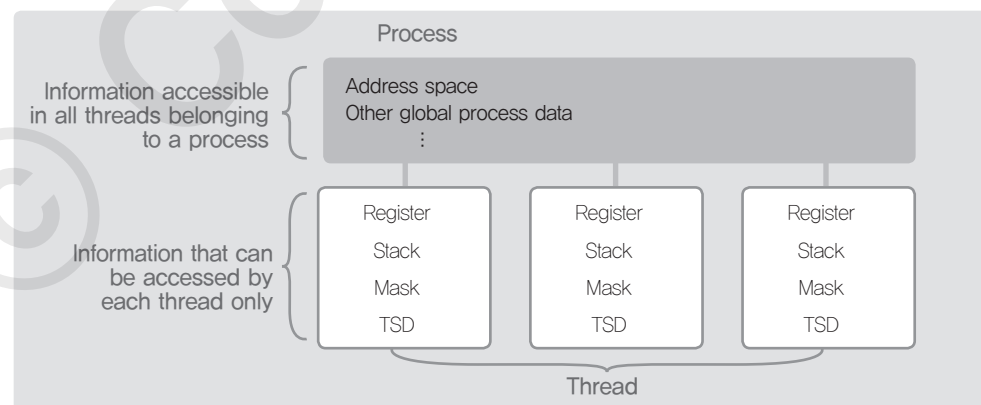
- New: A state where a process has been newly generated but is not executable by the OS yet
- Ready: A state of waiting to have a CPU allocated to execute a process
- Run: A state where a process holds a CPU
- Exit: A state where CPU has been deallocated after process execution is over
- Wait: A state of waiting for some event such as I/O completion after a process is running after being allocated with a CPU

② Process Control Block (PCB)

The PCB is to enable a process to store necessary information for process management. Whenever a process is generated, its unique PCB is generated, and once a process is completed, PCB is removed. The PCB includes the Process Identification Number (PID), process state, program counter (a value referring to a command to be executed next), priorities in scheduling, leverage information and information on main memory management.

Thread

It is a program unit where various resources of a system are allocated and executed as a work unit within a process. As for a thread, unlike process creation, there is no need for the OS to initialize resources to be shared with a parent process. That is why there is a little amount of overhead to create and exit a thread.



〈Figure 4〉 Thread-process relationship

This kind of thread can shorten the response time of an application program by creating a single process in many threads, enhancing concurrency, and raising the hardware/OS performance and throughput of an application program. Since the execution environment is shared, waste of memory can be reduced, and it can support efficient communications using commonly accessible memories.

Concurrent Process

A concurrent process refers to two or more processes doing the processing concurrently. If an exclusive approach to common resources is not guaranteed, there could be a big problem once a failure occurs. Solutions to prevent this type of failure include such methodologies as critical section, mutual exclusion, semaphore and monitor.

① Critical section

It is a section where only a single process is permitted to use resources or data at a particular work point for data and resources shared by many processes in multi-programming OS.

② Mutual exclusion

It is a technique to modify one process at a time in order to execute a task fairly when many processes access common resources. It refers to a situation where if another process demands the already allocated resources when a process uses common resources, the process that demanded the resources would have to wait until the resources are released.

- Solutions to mutual exclusion: Software-side solution (Dekker algorithm, Peterson algorithm), hardware-side solution (interrupt de-activation, test-and-set command, compare-and-swap command)

③ Semaphore

It is to guarantee accuracy and consistency of data so that proper outcome could be generated even if a process executes a task in a critical section. Semaphore is a method to form mutual exclusion by Dijkstra and to control so that access can be made only through the two operations of protected variables P(Wait) and V(Signal).

④ Monitor

It is a special programming technique to implement mutual exclusion in concurrent multi-programming unlike semaphore supporting the OS. It has a structure of concurrency to fairly allocate common resources.

Deadlock

It refers to a phenomenon where two or more processes demand resources held by different processes and the resources demanded are not permanently allocated, so the processes would have to wait infinitely.

〈Table 4〉 Conditions leading to a deadlock

Conditions	Description
Mutual exclusion	A state where one process at a time can use common resources
Hold & wait	A state of demanding another resources while holding the current resources
Non-preemption	A state where resources allocated for each process cannot be forcibly released until the usage is completed
Circular wait	A state where demand for resources among different processes is continuously repeated

〈Table 5〉 Solutions to a deadlock

Conditions	Description
Prevention	A method of preventing a deadlock by removing conditions for a deadlock to occur
Avoidance	A method of appropriately avoiding a deadlock without removing conditions for a deadlock to occur
Detection	A method of allowing the occurrence of a deadlock, and detecting and resolving causes when it occurs
Recovery	A method of resolving a deadlock by restarting a process in a deadlock or returning it to an original state

Scheduling

It is to allocate a process efficiently to the CPU to maximize the usage of the CPU in the OS that supports multi-programming.

① Purpose of scheduling

- Fairness of a process
- Maximization of throughput per unit hour
- Minimization of response time
- Predictable execution time
- Prevention of system overload
- Balanced resource utilization
- No indefinite postponement in process execution
- Prioritization, etc.

② Types and characteristics of scheduling algorithms

- Preemptive scheduling: A way where another process overtakes the CPU (resource) when one process occupies the CPU (resource)
- Non-Preemptive scheduling: A way where the CPU (resource) cannot be allocated to another process until a task is completed once the CPU (resource) is allocated to a process

〈Table 6〉 Types and characteristics of scheduling algorithms

Type	Methods	Features	Classification
FIFO (First In First Out)	<ul style="list-style-type: none">• A mode of allocating the CPU in the order of first in first out it as the simplest scheduling technique• Allocating the CPU in the way of first come first served in the line of resource requesting processes (queue)	<ul style="list-style-type: none">• Inappropriate for conversation types• Simple and fair• Predictable response speed	Not selected
Priority	<ul style="list-style-type: none">• A mode of allocating the CPU to a high-priority process by prioritizing each process	<ul style="list-style-type: none">• Fixed prioritization• Variable prioritization• Purchased prioritization	Not selected
SJF (Shortest Job First)	<ul style="list-style-type: none">• A mode of allocating the CPU in the order of processes with the shortest expected task operating time	<ul style="list-style-type: none">• Advantageous for short work time and large tasks consuming a lot of time	Not selected
SRT (Shortest Remaining Time)	<ul style="list-style-type: none">• A mode of allocating the CPU to a process with the shortest remaining time in the middle of work• Selection-type SJF	<ul style="list-style-type: none">• Same as SJF for work handling but theoretically taking the shortest waiting time	Selected
R-R (Round Robin)	<ul style="list-style-type: none">• Just like FIFO, a process coming in first is executed first, but each process uses the CPU for a designated period of time.• Selection style FIFO	<ul style="list-style-type: none">• Most appropriate for TSS• Same as FIFO if allocated time is big• Context exchange occurring often if allocated time is short	Selected
Deadline	<ul style="list-style-type: none">• A way of making sure that a process is completed within a limited amount of time	<ul style="list-style-type: none">• Excessive overheads and complexities occur because a deadline has to be calculated	Non-preemption
HRN (Highest Response-ratio Next)	<ul style="list-style-type: none">• Complementing the drawback of having to spare excessive time in a big project at SJF	<ul style="list-style-type: none">• Priority = (waiting time + execution time) / execution time	Non-preemption
MLQ (Multi-Level Queue)	<ul style="list-style-type: none">• Processing different tasks in each queue by time slices	<ul style="list-style-type: none">• Each queue using exclusive scheduling algorithms	Preemption
MFQ (Multi-Level Feedback Queue)	<ul style="list-style-type: none">• Processing throughout multiple feedback queues through a single stand-by queue	<ul style="list-style-type: none">• Higher efficiency in a CPU and an IO device	Preemption

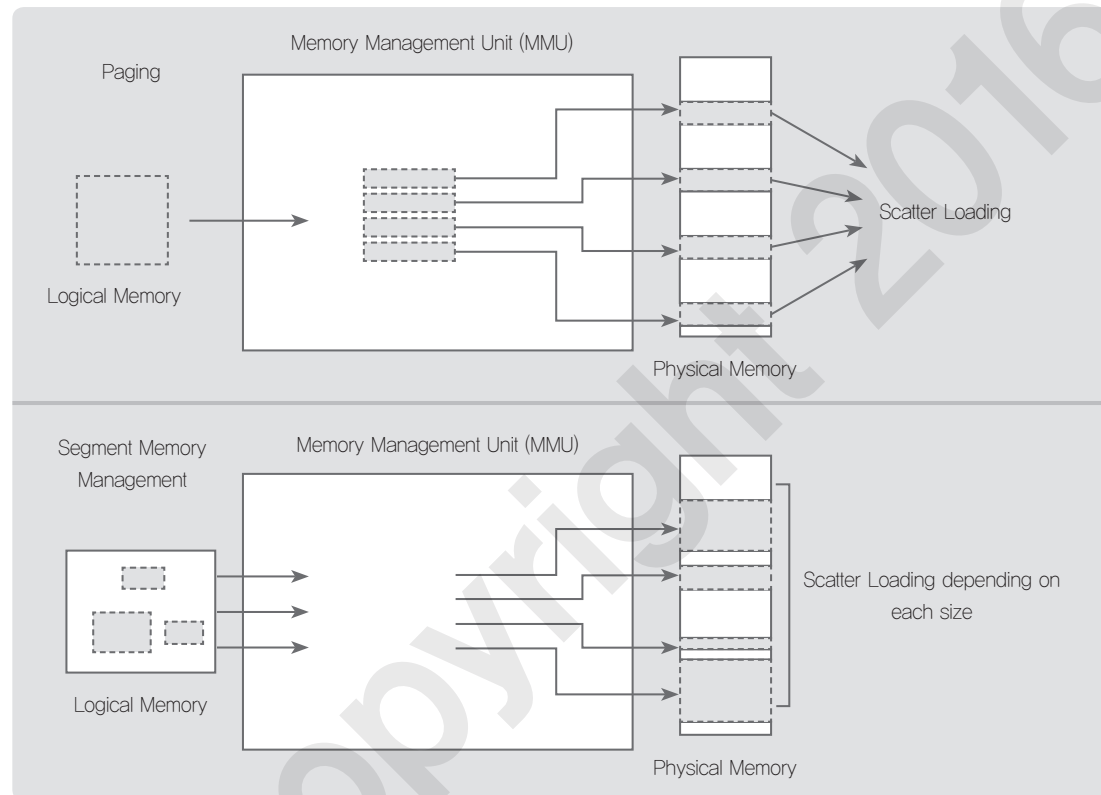
Virtual Memory

As a technique to resolve the limited memory space issue in the OS, a virtual memory enables the main memory lacking enough memory space to be used as a high-capacity memory with much free space. A virtual memory does not physically exist, but logically uses a high-capacity auxiliary memory as main memory. Therefore, it is essential to map the address of a virtual memory into that of a main memory to execute a program stored in a virtual memory.

① Implementation techniques for a virtual memory

For a virtual memory, there is a virtual address referred to in a process, and a physical address referring to an available section in a main memory. Whenever a process approaches a virtual address, a system has to convert this into an actual address through the Memory Management Unit(MMU). There are two techniques depending on the block configuration: paging and segmentation techniques. Sometimes these two methods are utilized as a combination.

- **Paging:** A technique where a main memory is divided into the same size called 'frame', and a task of processing stored in a virtual memory is divided into the same size called 'page' which is loaded into the frame of a main memory
- **Segmentation:** A memory-saving technique where a task of processing stored in a virtual memory is divided into segments, a logical unit of various sizes, and is then loaded onto a main memory and executed



〈Figure 5〉 Comparison of memory allocation for paging and segmentation

② Virtual memory page swap

If a page in a virtual memory for paging is filled up, a free space must be made available before a new memory page is called in from an auxiliary memory. A page swap impacts the efficiency and performance of a system since the system selects a page to be swapped.

- **Optimal:** A technique to swap a page that is not going to be used for long. It is the optimal technique with the minimum page fault rate but cannot be implemented in reality because a process behavior is unpredictable.
- **First In First Out(FIFO):** A technique to swap the first page loaded by tracking the order of loading in the main memory
- **Least Recently Used(LRU):** A technique to swap a page not used for a long
- **Least Frequently Used(LFU):** A technique where a page used the least or not used intensively is swapped

because the focus is on the frequency of usage

- **Not Used Recently(NUR):** A technique to swap a not-recently-used page with a page referred to, based on a tendency of not using it in the near future

③ Factors impacting the performance of a virtual memory

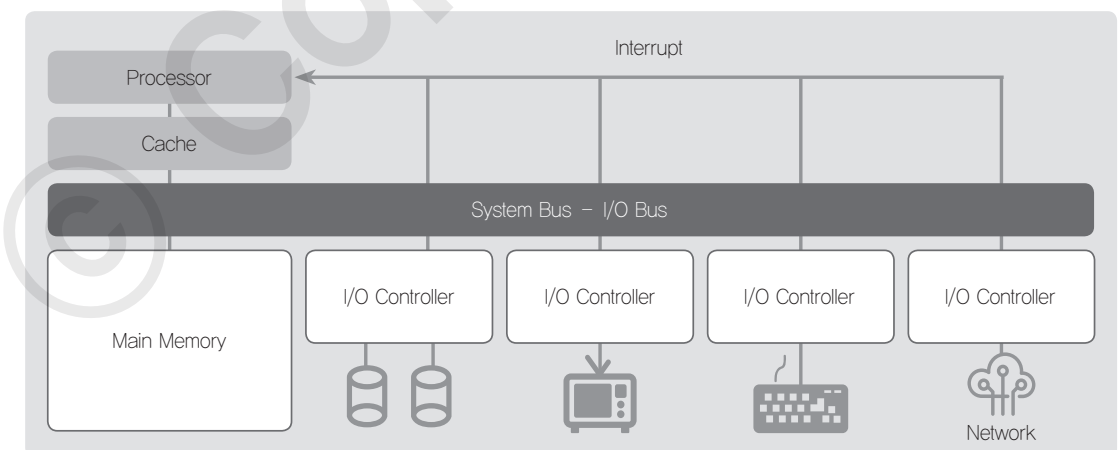
- **Working set:** A set of a page list referred to frequently for a certain period of time to efficiently execute a process. The phenomena of a page fault and a page swap is reduced by placing a working set frequently referred to.
- **Thrashing:** A phenomenon that lowers the CPU utilization rate because it takes longer to swap a page than processing. If the extent of multi-programming is reduced, the CPU utilization is raised or a working set is utilized to prevent thrashing.
- **Locality:** A property of intensively referring to a certain page while a process is executed. It is divided into time locality and space locality.

File System

A file system manages file resources, and data control and access. It provides the following: file management including file storage, reference and sharing; auxiliary storage management; file integrity mechanism to prevent data damage in a file; access methods to access stored data; and file back-up and recovery.

I/O System

An I/O system includes an I/O device and an I/O module (controller). A physical I/O device performs I/O for data and information between an actual processor and computer user. An I/O module provides the following characteristics: control of an I/O device and timing adjustment; communications with a processor; communication with I/O devices; data buffering and error detection.



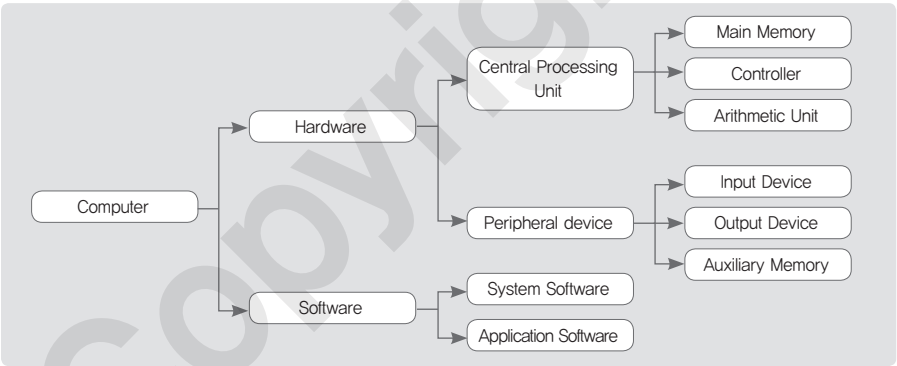
〈Figure 6〉 Basic configuration and an I/O module in a computer system

Latest OS Technologies

- ① **Mobile OS** is an OS that controls a mobile device or an information device.
- ② **Web OS** is a virtual OS or a series of applications that run on the web browser, referring to copying, swapping or complementing the desktop OS environment. Compared with the existing OS, web OS is different in that it is a highly collaborative environment with many people, and applications are run through the web. An iconic example is Google’s Chrome, and others include myGOYA, a free online OS.
- ③ **Embedded OS** is an OS embedded in hardware, supporting a computing environment including robots, household appliances, automobiles and aircrafts. Examples are Embedded Linux, Win CE,NET and VxWorks.
- ④ **Distributed OS** is an OS that manages a distributed computing system equipped with characteristics enabling higher transparency and performance as if many computers logging onto a network are treated as one. Examples include UCB’s DASH.

04 Computer Structure

Components of a Computer



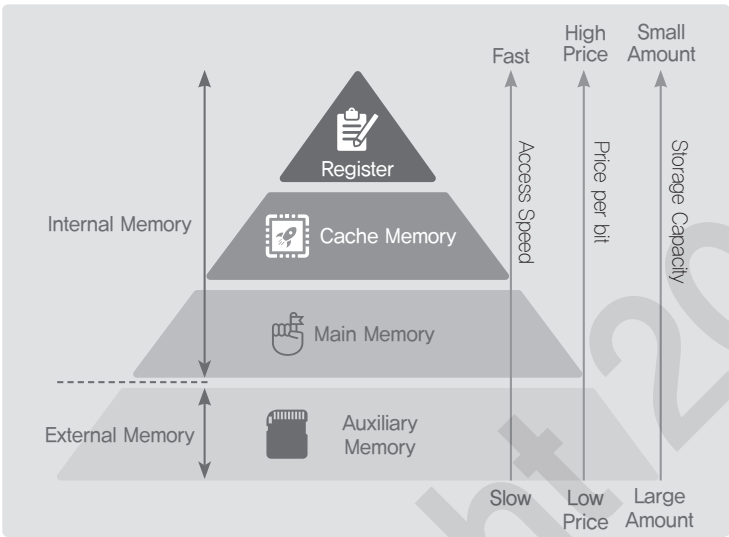
〈Figure 7〉 Components of a Computer

The Central Processing Unit (CPU) reads and encodes a command from a memory and manages data (e.g. reading, processing and storage) if data is necessary in executing a command. The CPU consists of control units, arithmetic units, registers and buses that transfer data by connecting them.

- ① Control Device: Controlling and managing all tasks that happen within the CPU
- ② Arithmetic Logic Unit(ALU): Performing arithmetic operation and logic operation as a core component of the CPU
- ③ Register: A temporary storage to remember such data as a command or the arithmetic medium outcome to be processed within the CPU
- ④ Bus: A common transmission line for connection to exchange mutually necessary information such as the CPU, memory and I/O devices

Hierarchy Structure and Mechanism of a Memory Device

① Hierarchy structure of a memory device



〈Figure 8〉 Hierarchy structure of a memory device

② Evaluation factors for the performance of storage device

- Capacity
- Access time
- Cycle time
- Bandwidth
- Data transmission rate
- Cost

③ Classification and characteristics of a storage device

〈Table 7〉 Classification and characteristics of a storage device

Classification Standard		Explanation	Type
Usage	Main memory device	A memory device that temporarily stores a program or data to be processed by a computer	RAM, ROM
	Auxiliary memory device	A memory device that complements the shortage of storage capacity of the main memory and stores the data semi-permanently using the non-volatile characteristics	Magnetic disk, optical disk, etc.

Classification Standard		Explanation	Type
Physical Storage Types	Magnetic	A memory that remembers binary information based on the directions magnetic flux by maintaining the magnetic property	Magnetic tape, hardware disk, zip drive, etc.
	Optical	A device to record information using laser beams on the surface of an aluminum metallic plate	CD(Compact Disc), DVD(Digital Versatile Disc), BDA(Blu-ray Disk Association)
	Semiconductor	A device to store analogue information using the integrated circuit technology	flash memory
Whether or not data is kept when power is down	Volatile	A memory where all the information is deleted when power is down	RAM-based SSD
	Non-Volatile	A memory where the remembered information is maintained even if power is down	magnetic core, auxiliary memory
Access Mode	Sequential access	Making access sequentially from the beginning to a location necessary for a memory space	Magnetic tape
	Direct access	Making direct access to a location necessary for a memory space	Disk, flash memory, etc.
Whether or not the content is maintained	Destructive	A memory where the stored content is destroyed after reading it	Magnetic core
	Non-Destructive	A memory where the stored content is maintained intact even after reading it	A memory device excluding the magnetic core

④ Addressing Mode

The location where data is stored in the main memory is called an 'address.' Various addressing modes are offered so that a command can be designated by appropriately utilizing limited command bits and the memory capacity can be efficiently used.

- Direct Addressing Mode
- Indirect Addressing Mode
- Implied Addressing Mode
- Immediate Addressing Mode
- Displacement Addressing Mode: Relative Addressing Mode, Indexed Addressing Mode, Base-register Addressing Mode

Latest Technologies and Trends

① Multi-core processor

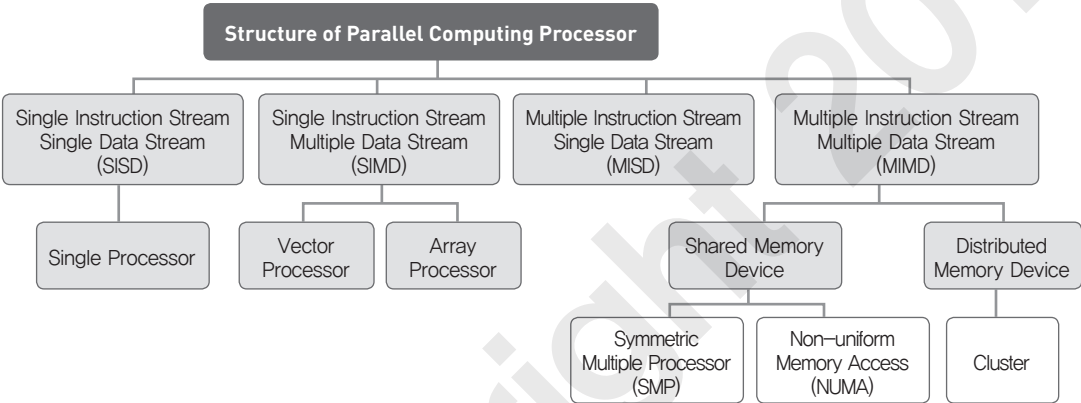
It is a processor with two or more cores like a dual-core or a quad-core. It increases the work speed compared

with a single-core process, and is useful to execute high-performance work such as video encoding and game. However, its weaknesses are high prices and high power consumption.

② Parallel system

Parallel processing refers to the processing of many tasks concurrently by many processors such as I/O channels or processors. Writing a program is somewhat difficult, but throughput speed is high, a memory can be shared, and even a failure in some hardware would cause no harm to the overall system operation. It can be applied to artificial intelligence where voices or images are processed and to a system where a large amount of data is accurately processed in a short time frame including state-of-the-art military equipment such as guided missiles. Such additional tasks as partitioning, scheduling and synchronization are required for parallel processing.

- Classification based on Flynn considering the parallel flow and data flow [7]



〈Figure 9〉 Structure of a parallel computer processor

〈Table 8〉 Classification of a parallel computer processor

Type	Description
SISD (Single Instruction Stream Single Data Stream)	<ul style="list-style-type: none">• A single processor system that processes one instruction and data at a time consecutively• Internal structure of a processor for higher performance: pipelining and superscalar
SIMD (Single Instruction stream Multiple Data stream)	<ul style="list-style-type: none">• A structure executing multiple data concurrently with a single instruction, executing the same operation for multiple data• Also called as the 'array processor,' enabling synchronized parallel processing by an array processor
MISD (Multiple Instruction stream Single Data stream)	<ul style="list-style-type: none">• Each processor executes different instructions from one another, but data that is processed is a single stream. It has not been designed or implemented before.• Unsynchronized parallel processing driven by the pipeline is possible.
MIMD (Multiple Instruction stream Multiple Data stream)	<ul style="list-style-type: none">• Multiple processors run different programs for different data.• Most parallel computers belong to this type.• Divided into the tightly-coupled system and loosely-coupled system depending on the level of data interaction

〈Table 9〉 Classification of processors with processing features [7]

Type	Description
Pipeline Process	<ul style="list-style-type: none">One processor is divided into multiple sub-processors with different features, and each sub-processor processes different data concurrently.With an inter-sub-processor overlap, parallelity comes in with a vertical dependent structure executing tasks by stage
Array Processor	<ul style="list-style-type: none">A processing structure that arrays arithmetic units parallel to process data at a high speed, which is used to calculate vectors or matrixesA processing structure where different processing elements are synchronized in a single control device, and each processing element processes each data concurrently based on a single instruction
Multi-processor	<ul style="list-style-type: none">As the most common model for parallel processing, allocating many independent tasks to many processors in a system and enables task execution concurrentlyNeeds additional features such as scheduling, synchronization, memory management resources and performance management

Example Question

Question type

Performance question

Question

A step-stone bridge made of many stones as shown in the [Image] is the only way to cross the river between two villages, and has three constraints.
Team A is entering the step-stone bridge to head toward a mountain village, and Team B is entering the bridge to go to a village from the opposite side.

- 1) Explain how you would call this state in a paragraph.
- 2) Suggest four conditions that would cause this state (5 points).
- 3) Explain an algorithm to avoid this problem (10 points)

[Image]



[Table for the Question]

Constraints

- 1) Only one person can step on one stone at a time.
- 2) Once you are on the bridge, you cannot go back.
- 3) If both sides have already entered, the team that heads towards the mountain village is given a chance to proceed first.

Intent of the question

To validate if one can infer a state of deadlock through an example, and also accurately explain conditions leading to deadlock and express using the Banker's algorithm or the natural language in order to avoid deadlock based on the understanding of the algorithm for parallel processing.

Answer and explanation

- 1) Deadlock
Team A is entering the step-stone bridge to head toward a mountain village, and Team B is entering the bridge to go to a village on the opposite side. As they meet in the mid-point, they would proceed to go on in opposite directions and retreating is impossible. Therefore, each team would wait for the other to get out of the way, blocking all each other and getting stuck. This is an actual example of deadlock.
- 2) Conditions leading to a deadlock
 - Mutual exclusion: A condition where only one process can utilize resources at a time. Resources occurred by one process cannot be approached by other processes.

- Hold-and-wait: A condition of waiting, since a process waiting for resources is already occupying other resources
- No preemption: A condition where resources occupied by a process cannot be forcibly taken away by another process
- Circular wait: There is a presence of a closed chain among processes: a state where a process occupies a resource blocked in a closed chain with a circle formed within a resource allocation graph, and this resource is demanded by another process in the chain, which is waiting for them.

Once up to this condition is met, sufficient conditions leading to deadlock would occur.

3) Examples of algorithms to avoid deadlock

1. Status check

Condition	Explanations
Stable	A state where all the people can pass through steps and enter the step ramps
Unstable	A state where people are on the steps
Deadlock	A state where people that have entered the steps are in opposite directions

2. Concept of algorithms

- To write algorithms so that people could wait without entering the step-stone bridge if there are people there, in order to avoid deadlock and enter it only when the bridge is completely empty

3. Algorithms (possible to write the natural language)

- 1) Checking out the current status of the step-stone bridge
- 2) Changing the occupation status of the step-stone bridge if it is in a stable state
- 3) Allowing the current team members to enter the bridge
- 4) Handing over the resource after the entrants of the bridge have been emptied out
- 5) Waiting if it is an unstable state

[Banker's algorithm to avoid the general deadlock – Psudo Code]

```
boolean safe(state e)
{
    int currentavail[m] ;
    process rest[<number of the process>]process rest[<number of the process>]process rest[<number of the process>]
    currentavail = available;
    rest = {all processes};
    possible = true;

    while (possible)
    {
        <find a process Pk in rest such that
        claim [k, *] - alloc[k, *] <= currentavail ; >
        if (found)
```

```
{
    currentavail = currentavail + alloc[k, *] ;
    rest = rest - {Pk} ;
}
else
    possible = false;
}
return ( rest == null);
}
struct state
{
    int resource [m]
    int available [m]
    int claim [n][m]
    int alloc [n][m]
}

if (alloc [i, *] + request [*] > claim [i, *]
    <Error> ; /* The total request is bigger than what was demanded.
else if (request [*] > available [*])
    <Process suspended>;
else /* Simulation of resource allocation
{
    <Defining the new Status as follows :
    alloc [i, *] = alloc [i, *] + request [*]
    available [*] = available [*] - request [*] >;
}

if (safe(newStatus))
    <Executing resource allocation> ;
else
{
    <Recovering the original status> ;
    <Suspending the process>;
}
```

Related E-learning Contents

- Lecture 2 Software Engineering
- [Advanced] Lecture 1 Data structure/Algorithm (1) Linear data structure
- Lecture 2 Data structure/Algorithm (2) Non-linear data structure
- Lecture 3 Data structure/Algorithm (3) Sort and search

III

Analysis and Specification of Requirements

▶▶▶ Latest Trends and Key Issues

The importance of object-oriented analysis has been consistently emphasized, and specifications using UML have been much utilized.

▶▶▶ Study Objectives

- * Able to understand the modeling techniques required in the analysis of requirements, and validate three perspectives toward the software system (functional, dynamically and information-wise)
- * Able to understand structural analysis and object-oriented analysis and explain their differences

▶▶▶ Practical Importance High

▶▶▶ Keywords

- Functional and non-functional requirements
- Data flow, data dictionary, mini-specification
- Use case diagram, class diagram, sequence diagram, activity diagram

+ Practical Tips Intensity of requirement analysis

The most intensive process in a software development project includes collecting, organizing and confirming customer requirements, which, in fact, is the biggest cause for a project delay and failure.

At a time when there is a greater need for specialists to clearly understand and analyze customer requirements, and new diverse information technologies are leading customers' business, the importance of business analysts capable of planning a business model and making suggestions to customers is emphasized significantly.

Definition of corporate requirements is organized by members of a project T/F representing tasks, requiring the refining of very specific levels of requirements.

The process of analyzing these requirements has been made intensive based on the following reasons:

- Customers are required to provide specific requirements, but customers, in many cases, request developers to organize requirements based on their expertise and experiences.
- Requirements continue to flow into the entire project process, so sometimes a negotiation takes place to stop any additional receipt of requirements.
- When related failures occur, requirements can be used to determine who is to be held accountable depending on the presence of requirements, approval and verification.

01 Analysis of Software Requirements

Outline of Requirement Analysis

① Definition and importance of requirements

Requirements are the most important information to determine the scope and scale of a project, so constant updates are needed through change management tracking throughout the entire project. Requirement analysis is a process of refining requirements of a requestor depending on the scope and characteristics of a system to be developed. Analysis of requirements is a process to enhance the understanding of a given system prior to the designing stage. Validating customers' demands to make software satisfying customer needs is critical to managers and practitioners developing a system and is also the biggest challenge. The key focus in the stage of analyzing requirements is on 'what' in the perspective of customers instead of 'how to'. Analysis of requirements is a process of learning about a user by observing the user's activities. Defining what roles software would perform and what it would provide to customers is significantly different from the issues of development, that is, 'what' to make and 'how'. Analysis of requirements focuses on what features a system would perform in the application perspective in describing the goal of a system without describing how the function would perform in the engineering perspective.

② Classifications of requirements

Requirements can be divided into functional and management aspects, and classification of functional requirements is frequently utilized.

<Table 10> Classification of requirements (functional/management perspectives)

Type	Requirements	Description
Functional Aspect	Functional requirements	• Software features and attributes required to implement a target system • Directly describing features and movements, and behaviors required for a system (output, input, deletion, interface, etc.)
	Non-functional requirements	• Requirements to define the overall quality or characteristics of a system (performance, availability, maintenance, security, etc.)
Management aspect	Continuous requirements	• Inducing problem domains from a model to stable requirements on core organizational activities
	Volatile requirements	• Requirements that change depending on the system development environment and user demand

③ Difficulties in analyzing requirements

Requirements are obscurely written with a lack of knowledge about various domains, or by misunderstanding or wrongly analyzing the demand suggested by a user. Conflicts over description or obscurity in meanings occur as participants would express or analyze problems suggested among each stakeholder in different perspectives depending on their own roles and environment. Factors that hamper communication increase, difficulties arise in sharing tasks and features clearly as the development system gets bigger in scale, many analysts execute tasks jointly, and it gets difficult for adequate communication to occur among participants depending on the

organizational environment and task characteristics, therefore lowering consistency. It is the norm that requirements are modified amid continued evolution, driven by new knowledge and environment acquired through users' demand for change for the system and during the system development stage. Analysis of requirements is difficult because of the following reasons, given such characteristics of requirements:

- Difficult to acquire expertise on problem domains
- Difficult to maintain consistency in analysis due to different perspectives among participants
- Difficulties resulting from higher complexity in communication as a project gets bigger in scale
- Difficulties in tracking management due to continued changes in requirements

Elicitation Methods of Functional and Non-functional Requirements

① Elicitation methods of functional requirements

Elicitation methods might vary depending on the classification of functional requirements, which can be summarized as follows:

<Table 11> Elicitation methods per classified functional requirements

Type	Elicitation Methods
Features	• Validating the purpose (role) of executing a system • Validating the system execution point and the execution mode upon the operation (input, correction, deletion, inquiry, output, etc.) • Validating the timing of changes and expansion in a system
Data	• Input and output data and forms of data • Data accuracy and amount of data injected into a system • Period of data storage
Interface	• Validating the presence of particular media used to transmit data • What would be the input flowing out and in from an outside system? • Is there a particular form of data?
User	• Who is going to use the system? • Validating the user group and computer usage experiences per user group • Validating training needed per user group

② Elicitation methods of non-functional requirements

Non-functional requirements can be broadly classified into resources, performance, security, and quality, etc., and each domain is separated so there could be various elicitation methods.

〈Table 12〉 Elicitation methods per classified functional requirements

Type	Elicitation Methods
Resources	<ul style="list-style-type: none">Validating necessary resources and personnel for system setup and maintenanceCapacities required of developersCharacteristics of hardware to be usedResearch on electric power, temperature and humidity of a systemConstraints in storage space of a development system
Performance	<ul style="list-style-type: none">System speed, response time, throughput rate and size of throughput data
Security	<ul style="list-style-type: none">If there is access control over data and the systemAllocation of rights per user R&R (roles and responsibilities)System back-up period and personnel in chargeCountermeasures against disasters (fire, flood and human disasters)Physical security measures
Quality	<ul style="list-style-type: none">A need for quality characteristics including reliability, availability, maintenance and securityMean time between failure (between MTTF, mean time to failure)Allowable recovery time after the system operation stopsEase-of-use in design modificationsEfficiency in resources usage and methods to measure response time

02 Modeling

What is Modeling?

Modeling refers to a process of simple diagramming or mathematically expressing the characteristics of the system to analyze the target system performance or its operational process. Modeling is used to understand our real world, enabling its expression. Modeling must help validating views of software in various perspectives and software requirements. Modeling outcome becomes a core part of a requirement specification, providing information to move to the progress of the next stage in a project. Modeling outcome is used as a means of conversation between users and developers, being significantly conducive to validate the framework and skeleton of a system needed in the development stage (design, implementation and testing included) while eliciting requirements needed in the initial stage of a project.

Three Viewpoints in Modeling

Software may look different and be used differently depending on viewpoints, which can be expressed in three perspectives.

〈Table 13〉 Three viewpoints in modeling

Viewpoints	Description
Function Viewpoint	A functional model describes a system in a viewpoint where what features are executed by software. The model shows what outcome will be generated for given input, describing arithmetic and constraining conditions.
Dynamic Viewpoint	Describing the system state and causes changing the state (including events and time) by focusing on system operation and control
Information Viewpoint	Used to catch static information structure of software, validating information entities used in a system and detecting the characteristics of these objects, and relationships and relevance between the objects



03 Structured Analysis


Structured analysis is a representative modeling method oriented to processing by functionally viewing software as data and data processing. Its function is to generate outcome by receiving input, and the activity to perform features is called a process. A process refers to a transformational course of receiving and processing input and generating new output, and each process can be defined as input, instruments and techniques used in a process, and output generated as output.

Data Flow Diagram (DFD)

It is an analytical output by validating problem domains in the target analysis system, service and software, and distinguishing a process of input/output for given domains and processing the input, and expressing the overall data flow. The DFD is completed by continuously dividing the highest-level context diagram into a child process as needed. The DFD uses the four following notations for indication:

〈Table 14〉 DFD notations

Signs	Explanations
External Entity 	An external entity including a user exchange information with a system outside the system
Process 	A converter and bubble that processes and converts information within a system. Indicated as a circle or a circular square

Signs	Explanations
<div>Data Flow</div> 	A data flow or a data unit indicating the information flow between processes with the arrow indicating the data flow
<div>Data Source</div> <div><div></div><div>Data Source</div></div>	An information source, file or database system storing data preserved for long

Mini-Specification

A mini-specification is used to describe what features will be executed in the lowest-level process in the DFD where division has been completed. The lowest-level process that cannot be further divided is called 'functional primitive' whose specific explanations are recorded in the mini-specification.

Data Dictionary

The Data Dictionary enables developers or users to conveniently use data by compiling that data shown in the DFD in a single location. The Data Dictionary is a compilation of definitions of data items within the DFD, consisting of definitions and explanations. The Data Dictionary is a metadata source for data on data including the definitions of data items and data flow.

04 Object-Oriented Analysis

Object-oriented analysis refers to an interaction between objects by considering given problem domains as a set of objects. It is understood as process of virtualization of transferring problem domains into a computer world in the same viewpoint toward the actual world. Its advantage lies in increasing reusability of objects and enhancing the understanding of participants. The aforementioned three viewpoints (information, dynamic and functional) are applied by phase to find out objects (or classes demanded in software) and discover attributes and operation of objects. Methodologies for object-oriented development are beneficial in that the identical methodologies and expression techniques can be applied to the entire software development process ranging from analysis, design and programming.

Use Case

Once the Use Case technique, well known for object-oriented analysis is utilized, communication between customers and system developers can be made more seamlessly and customer requirements can be more effectively found out. The Use Case can offer many benefits to various stakeholders in a project including customers, project managers, developers and designers. Customers' active participation can be induced by utilizing the Use Case, and by validating customer requirements quickly, functional requirements in a system can be determined initially, and the relevant outcome can be documented.

In the Use Case, stakeholders can be found out, and they can be classified into homogeneous groups depending on the roles of stakeholders, who are then classified into actors. Each actor has different views and usages for the system. Based on which Use Case, a usage of a system for each actor can be identified. The Use Case represents a usage of use of a system used by each actor. Each Use Case is a set of scenarios for a particular actor to fulfil a function or a duty.

A scenario is written for each Use Case. The scenario refers to a flow and course of events, and may include not only information exchange between the system and actors but also a circumstance, an environment or a background where interactions occur. The Use Case technique represents interactions between users and the system, being capable of clearly expressing what the system is executing, so it is utilized to verify requirements of users.

Information Modeling

Information exchanged between an actor and a system can be found out based on the Use Case scenario. Information modeling can be executed to validate information to be stored and managed within a system by utilizing the information and the output can be represented in the Class Diagram of the Unified Modeling Language (UML). The basic class required in a system can be identified through information modeling. Relatedness between classes can be found out through the inter-class relationships and attributes of each class can be found through information modeling, too. In this stage, the Class Diagram can be objected only where the Class, internal attributes of the Class and relationships between Classes are expressed.

Dynamic Modeling

Using the Information Modeling mentioned above, attributes and relationships of the Class comprising a system have been represented in the Class Diagram, based on which dynamic modeling can be looked into. Dynamic modeling is a process of finding the operation of Classes by taking an interest in the changes in the state or operations of objects comprising a system, or in the interactions among objects.

For the UML, the Sequence Diagram is used to validate cross-object interactions, based on which operations of each Class can be elicited. The Sequence Diagram is normally written for each Use Case. If the Use Case scenario is the one written by regarding a system as a black box, the Sequence Diagram even represents a process of interaction among objects within a system by expanding the Use Case scenarios.

Functional Modeling

Sometimes, various features must be executed to execute the operations elicited from the Sequence Diagram. These features have an internally complicated logic within which they can be expressed as smaller operations. The Activity Diagram enables the elicitation of potential and new operations. As a diagram used to accurately understand the event processing within a Class, the Activity Diagram is used to understand the processing of a complicated process or validate additional operations of a Class.

05 Requirement Specification

Users' requirements and system features must be identified and documented based on the requirement analysis process. The requirements provide basic data to the Scope Management, impacting the entire project stages. Output of the requirement analysis is the Requirement Specification, which is also dubbed as the Function Specification or Target Document. The requirement specification is an important document connecting users and system developers. There are several mantras in software development: "set a goal and act it out", "define requirements and develop them." Their message is to utilize the Requirement Specification as the criteria and standard to test the product performance and quality made from the beginning of the project. For software development, a system is made twice: first, based on requirement analysis, and second, through the actual development.

The Requirement Specification is often written by analysts, but is sometimes co-developed with customers. It must be written for users and developers to easily understand it. It is used as a commitment document between customers (or users) that have made investment for development cost and developers. It would later clearly validate who is to be accountable for potential problems and changes that might occur. To this end, stakeholders related to the system would ideally agree to the requirement analysis outcome (Requirement Specification), sign on it, and move onto the next stage (design, coding and testing).

[Case]

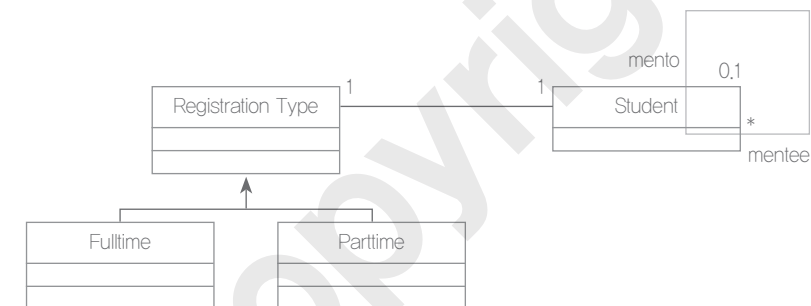
- A) A student can register by choosing one of the two options: a full-time student or a part-time student.
- A full-time student can be converted into a part-time student and vice versa upon follow-up requests.
- B) There are mentor-mentee relationships among students.
- A single student can be a mentor for many students.
 - A single student can have a single mentor, and if unwanted, a mentor does not have to be designated.

* Do's and don'ts upon writing: Use associations related to the minimum number of Class on requirements, indicating inheritance and multiplicity. Attributes and operations, etc. are to be omitted.

Intent of the question

To validate if one has the knowledge on requirement analysis methods and writing the Class Diagram

Answer and explanation



Related E-learning Contents

- **Lecture 5** Analysis of Software Requirements

Example Question

Question type

Performance question

Question

The following [Case] is a part of requirements on a student management system. Write the UML Class Diagram reflecting all of these requirements.

IV

Principles of Software Design

▶▶▶ Latest Trends and Key Issues

The world has now entered an era of infinite competitions over quality, and software is no exception. Software design is closely related to software quality. Making efforts in the analysis of requirements is a part of a process to make a favorable design. In the stage of analysis and design, failure cost of software can be significantly saved as quality management and prevention are made more robust.

▶▶▶ Study Objectives

- ✱ Able to explain the mechanism of software design, its types and descriptions to be considered in software design
- ✱ Able to explain concepts of cohesion and coupling as standards for module design evaluation
- ✱ Able to understand structural design methods and express the description for the designs

▶▶▶ Practical Importance High

▶▶▶ Keywords

- Division, abstraction, information hiding, stepwise refinement, modularization, structuralization
- Cohesion, coupling
- Transform-oriented design, transaction-oriented design, structure chart

+ Practical Tips Rocket explosion due to failure in software design

Software is abstracted into the computer world based on the analysis of requirements in the design process, which must include numerous variables defined in the design, the variable types where variables are utilized, and exception handling resulting from errors. Design defects in software are limited in the process of repeating a test such as the analysis of boundary values. Data backup and dual servers must be designed to stably support this for stable execution.

On June 4, 1996, an Ariane 5 rocket exploded on launch due to the design of a software exception resulting from the mismatch in processing types of variables and inadequate consideration in redundancy design.

The processing variable types were 16-bit integers, but because 64-bit numerics were entered, the processing that occurred in the course of transforming the integers was not fully reflected in the design, leading to the overflow error. As a result, information such as altitude and speed could not be sent to the central control computer.

At a time when SRI-2 ceased to function, SRI-1 in parallel had to be in active mode. However, there was no effect when the identical data were entered in the same architectural design. The Ariane 5 rocket lost control of engine nozzle angle and propulsion, and exploded after failing to orbit.

According to the hazard analysis, three out of seven exception handlings were omitted to fulfill a maximum workload target of 80%, so the software development side was not the only cause of the problem. At the end of the day, high-quality software design cannot be guaranteed by a design per se. High-quality software can be completed only when cases to guarantee the highest testing scope are used in the development and test process, through which unpredictable defects could be completed so that the design could be repeatedly complemented.

01 Principles of Software Design

Division

In the software design stage, user requirements must be continuously divided to reduce complexity in a problem domain, and outcome gained through this must be recombined into an appropriate group, considering the independence and dependence of role units. This is sometimes expressed as the basic design principle of 'divide and conquer.' If a system is considered by dividing it, problems can be more easily solved, and once features are divided or user interfaces are logically divided, solutions can be found more easily.

System components divided in an upper level are generally called 'subsystems', referring to program components that can independently execute features and be compiled. Subsystems refer to the division in the upper level in a system structure. A system designer divides a problem into subsystems, enabling many developers and designers to independently develop different subsystems. Clarity in this course of division aims to operate the entire system by seamlessly integrating subsystems to be developed by other developers.

Abstraction

Abstraction refers to a technique for implementing a product in a higher level, suppressing the more complex details below the current level to gradually approach a problem domain without losing the track of a big flow. Since what is neither of one's interest nor essential but specific is omitted, only the easy-to-handle and essential are expressed. Abstraction is an important principle occurring throughout the entire engineering process, and engineering is a process of moving from a high-abstraction level to a low-abstraction level. Abstraction is divided into data abstraction, control abstraction and procedure abstraction. Knowing the behaviors that occur outside on how components (or modules) interact by dividing a system precedes knowing specific implementation methods within a component. In other words, abstraction refers to dramatic omissions of implementation methods of components and focusing on the outside interface.

Information Hiding

Information hiding is a concept of hiding the internal description of each module and sending a message through the interface only: allowing a module or a subsystem to be designed without any impact of implementation of other modules by restricting the access to internal information. For instance, if modifications occur in a design process, the impact is to be on the minimum amount of modules. In other words, a module is defined by the interface with the outside, and information is said to be 'hidden' if specific information including an internal structure or a progress of a module is hidden in other modules. Information hiding maintains independence among modules.

Information hiding can be made available not just by using a programming language equipped with an information hiding mechanism. It is a basic principle in designing software, which is critical in maintaining independence among components in system design. It hides an internal structure among modules (abstraction) and communicates only with a designated interface without having to know about each other's internal structure. If modifications are required

in a module, the modifications are limited to the data structure within a module and to behaviors that access it, thus providing a basis for easy adaption to modifications and easy maintenance.

Stepwise Refinement

Stepwise refinement is specified by moving down from a program structure to specifics on a module. The level of abstraction goes down in stepwise refinement, and each function is decomposed, suggesting a solution. Refinement consumes a lot of efforts, and enables a specific description to enable system implementation.

Designing is a stepwise refinement process from high abstraction to low abstraction stages. The engineering flow leading to requirement analysis, design and program in problem description is also a stepwise refinement. In a structural analysis technique, the process of starting with a big process and dividing it stepwise into small processes to perform specific features is also a stepwise refinement.

Modularization

In any engineering field, most of the approaches are to divide a system into components. In software, a module represents these components. A module in software is called 'subroutine', 'procedure' or 'function.' A top-down approach is used to modularize a system to divide into functional units and in this case, control layers appear between modules.

Modularization enables a system to be intelligently managed, and solves problems of complexity. In other words, problems are divided into small-unit modules to solve big and complicated ones, and the modules are then divided and conquered. Modularization makes maintenance and modification of a system easy. However, as the number of modules increases, the size of each module decreases, cross-module exchanges increases, system performance drops and overload occurs. Therefore, minimizing cross-module interference for each role, focusing on each of its purposes and enabling it to play roles effectively become an important criteria for module evaluation.

Structuralization

Structuralization of a software system can be acquired by the division process, which is linked to the concept of divide and conquer mentioned earlier. Division takes place as a primary step in the requirement analysis, and becomes more specified in the design stage.

It is the duty of an analyst to find out important elements and features of a system and divide them, and that of a designer is to structuralize the outcome of analysis. How to divide a system is not a simple matter and there is no perfect guideline on the best way to divide it. However, experiences with the existing systems and relevant guidelines can be used by considering the system characteristics.

The existing systems would have several structural frameworks depending on the characteristics of a system. Once these frameworks are used, efforts and time can be saved to make a system that is similar in characteristics.

02 Cohesion and Coupling

It is not easy to accurately define a good design. A good design can be the one enabling efficient programming or the one helping to easily adapt to modifications so that the issue of software evolution could be properly resolved. In order for a design to be evaluated as a good design, its design document as the design outcome must be easy to read and understand, and the impact of modifications in a system must be localized.

Maximizing functional independence and reducing cross-module coupling would be the principle for an excellent design that enables easy maintenance. Object-oriented design is also a design technique improving design quality since it maximizes independence. In this section, let's take a look at cohesion and coupling as facts impacting the quality in the design stage.

Cohesion

Cohesion is a measurement of the maturity of a model to show how cohesive the inside of a module is. In other words, it shows the level of relevance of each component within a module to achieve a common goal. Cohesion represents the level of affinity of components within a module, serving as cement to tie each component. The highest level cohesion is the functional cohesion performed for upper modules while all components within a module perform a single function, and the lowest level cohesion is with the smallest coincidental cohesion where a module consists of irrelevant processing components.

Cohesion refers to the extent of performing a duty per module and a measurement to measure the independence of a module. Therefore, it would be desirable in a design where cross-component cohesion within a module is high, and also ideal if components in a module or a system perform a single logical function or represents a single logical entity (a component of a model). In this case, exchanges with the outside could be minimized.

Once the cohesion of a module increases, cross-module cohesion could be low, and vice versa. The best scenario in designing a software is to enable a high cohesion in modules, and low cross-module cohesion.

Coupling

Modules are correlated with other modules, and coupling refers to the complexity of correlation among modules. If cross-module exchanges occur a lot, and mutual dependence is high, cross-module coupling would go up. If an interface is not accurately configured or if its features are not accurately divided, an unnecessary interface would appear, leading to higher cross-module dependence and higher coupling.

Coupling is a method to indicate correlation among program components, being closely related to independence and cohesion of a module. If two modules perform features perfectly, regardless of the fact they are near each other or not, they are said to be 'completely independent', meaning there is no mutual exchange. The most ideal coupling is the data coupling communicated via cross-module parameters, having the weakest coupling. If one module directly refers to or modifies internal features and data of another module in the descriptive coupling, it would have the strongest coupling level, so must be avoided.

If the coupling is higher, it would be difficult to modify one module independently, causing ripple effects since modifications in one module would have a great impact on other modules. If ripple effects are big, it gets more

difficult to do system maintenance: when impact is on one, it would have the spread of impact on others. If such a phenomenon occurs in system development, it would be a headache for many people.

Therefore, it would be essential to minimize coupling in system design. If modules share variables and exchange control information, cross-module coupling would increase. Parameters are used for programming instead of global variables in the course of learning to program because of this reason (this way, coupling would be lowered).

03 Structured Design Techniques

Data flow-oriented design refers to a process of transferring the outcome of structured analysis, a function modeling technique, to structured design. When structured techniques are applied, a system is logically divided and modularized in the initial design stage, and as top-down refinement take places, top-down specification occurs.

A notation technique used in the structured analysis to represent a data flow is called 'bubble chart' or 'Data Flow Diagram (DFD).' While the DFD used in a structured analysis technique describes data flow and features in a logical sense, the structure chart used for structured design is to specifically represent a structure and design of software.

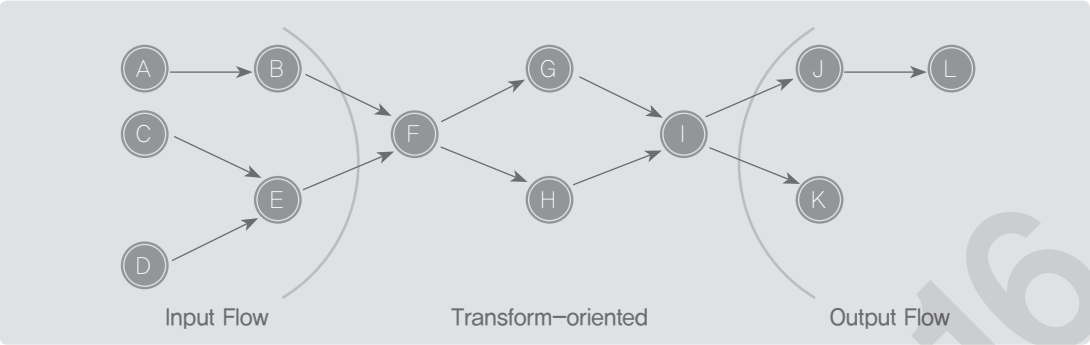
The structure chart is a notation technique mostly used to represent a program structure. The structure chart represents not only a program structure but also other descriptions, showing the data among all modules and control flow, and representing a program control structure like iteration and selection.

The structure chart is a model that shows cross-module data, exchanges of control information, major loops and decision-making, and descriptions on among module systems and modules in the design stage. Data flow and control flow in the structure chart can be distinguished as follows. Data is used to generate or modify new data by being used for calculation. However, control represents the order or conditions for calculation although it is not directly used for calculation. The 'flag,' mostly used in programming, has the value of 'on' or 'off' as a control signal used to imply the occurrence of conditions or to distinguish the boundary.

The structured design suggests a guideline necessary to change a requirement specification into a design document. This type of transform is determined by the type of information flow in a system. The information flow can be divided into two types: transform-based and transaction-based. A design document can be made by utilizing the characteristics of these two types represented in a requirement specification by transferring from analysis to design.

Transform Flow-oriented Design

The transform flow-oriented design is a technique where information is received and processed, and the system to output to the external world as the outcome is mapped into an appropriate computer structure. This technique divides a system into components and generates modules performing basic features and a hierarchical structure among modules. The transform flow-oriented design can divide systems in three ways: the first part refines data usable in a system by receiving the input; the second one processes data; and the third one receives processed information, transforms it into adequate output and generates the output. The system can be threefold: i) the incoming flow, a combination of processes in charge of input; ii) the transform center, a group of processes that process information input; and iii) the outgoing flow, a group of processes generating processed information.



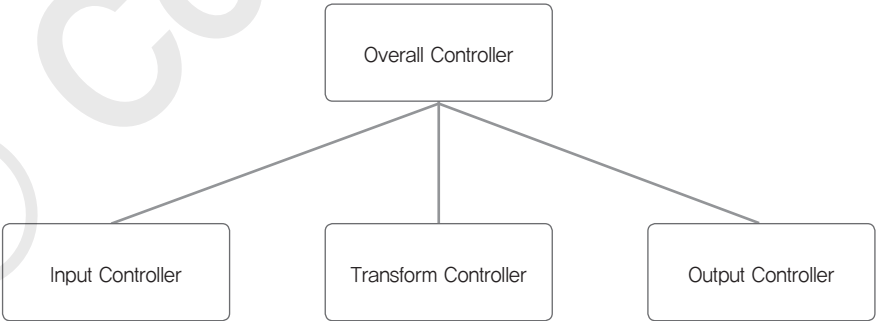
〈Figure 10〉 The incoming and outgoing flow in a DFD

Now, a data flow-oriented program structure is formed. The guideline needed to make a transform-based program structure is based on validating the incoming flow, transform-orientedness and outgoing flow in a DFD. Mapping into a structured chart in a DFD complies with the guideline. In order to make the highest program structure in a transform-oriented DFD, three components can be detected as follows:

- Input controller processing the input
- Transform controller processing the transform
- Output controller processing the output

The input controller receives input data from modules in lower layers, sends it to upper layers, and if necessary, refines the input data and transfers to upper layers. The output controller, meanwhile, receives the output data from higher layers and sends to lower levels. Again, if necessary, the output flow is refined to be sent to a module in lower levels.

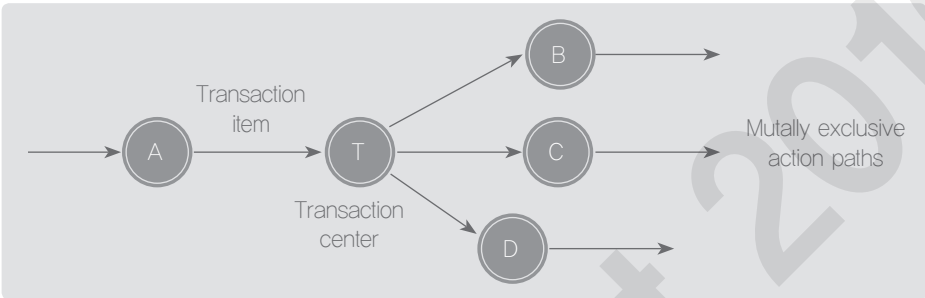
The number of controllers might vary by system complexity, and the number of controllers processing transforms could be determined by the transform-oriented complexity. A simple program structure based on transforms is as follows:



〈Figure 11〉 Tranform flow-based high-level program structure

Transaction Flow-oriented Design

A transaction refers to a situation where data or control signals, etc. induce some action. Transaction flow-based design is made possible when incoming input can be divided into different outgoing flows. A transaction evaluates the input value and flows into one of the many output paths based on the outcome. In this case, the center of the information flow is called the 'transaction center.' The figure below represents these characteristics, and Bubble T is the transaction center.



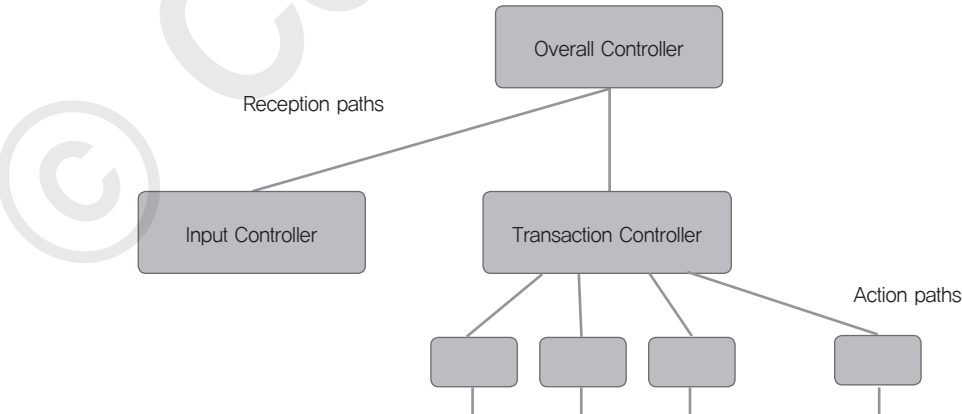
〈Figure 12〉 Transaction flow

If a DFD has a transaction flow, the first thing to do is to validate where the transaction center is. The transaction center is a bubble that receives a transaction as input and is connected to many output bubbles. Therefore, the transaction center has a single input path and many output paths. Each action path (output path) may consist of many bubbles, and have a transform flow or a transaction flow.

Accordingly, a program structure based on a transaction can be made. The structure consists of three components as follows:

- Modules that act based on transactions
- Modules that receive the input
- More than one modules relevant for each action path

A transaction-based program structure is in the following figure:



〈Figure 13〉 A transaction-based program structure

Example Question

Question type

Short-answer question

Question

The following example is a module written in a structured English or structured natural language. What is the relevant cohesion (write down the name of cohesion)?

[Example]

```
file_process (data) {
  ....
  Enter the data.
  Edit the data.
  Store the data.
  ....
}
```

Intent of the question

To check out if one knows about types of cohesion and has the competency to distinguish them

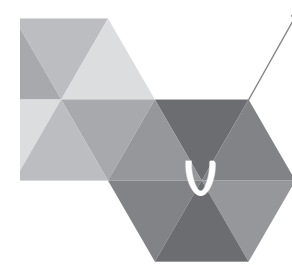
Answer and explanation

Sequential Cohesion

Common data are used and the order of input → edit → store must be maintained, so the order of these three features must not change. Therefore, the answer is 'sequential cohesion'.

Related E-learning Contents

- Lecture 7 Specific Software Design



Software Architecture Design

▶▶▶ Latest Trends and Key Issues

Soft architecture has recently been highlighted to play fundamental, useful and core roles to effectively reflect changes in IT and effectively develop high-quality software in a timely manner. Architecture design is an initial step in the design process to validate how a system is configured and how it interacts among components. It is also an essential stage to manage the system complexity.

▶▶▶ Study Objectives

- * Able to list the basic concept of software architecture (SA) and its components
- * Able to explain representative SA types
- * Able to explain methods to express SA

▶▶▶ Practical Importance Medium

▶▶▶ Keywords

- Modules, components and connectors, subsystems, frameworks
- Repository structure, MVC(Model-View-Controller) structure, client-server model, hierarchy structure, piper-filter structure
- Context model, component diagram, package diagram, deployment diagram

architecture is subject to final approval through evaluation and specification.

In the initial design stage, a system structure must be set to satisfy user requirements. It would be desirable to divide a system to resolve problems. Once a system is divided, complex problems become easy to solve. If features are divided and the user interface is logically divided, solutions can be found more easily.

System components divided in the upper level are called 'subsystems.' Subsystems normally include data and control structures, referring to program components that can execute features independently and be compiled. Moreover, subsystems are distinguished based on available services or features.

Frameworks are supporting units where the knowledge is collected to repeatedly reflect on the architecture design in designing subsystems. Frameworks are general structures extendable to make specific subsystems, and enhance the level of abstraction by providing specific implementation methods.

Architecture design is a process to configure the system composition to satisfy system requirements, describing the cross-component(module) relationship within a program structure by utilizing the outcome in the requirement analysis. The main purpose of structure architecture is to develop a modularized program structure and represent the cross-module control and interface.

02 Types of Architectures

Let's take a look into representative architecture types found in developing software.

Repository Structure

It is a structure where all shared data to be used in a system is stored in a single spot so that all subsystems can share the data in case a subsystem generates data and other subsystems use the data. The structure of a shared repository is appropriate to share a large amount of data.

MVC (Model – View– Controller) Structure

MVC structure is a framework frequently used to design the GUI as an approach to support various expressions to interact with one another: if the expression of one object is modified, all other expressions are renewed. Using this method, modification becomes simplified and reuse becomes easier.

Client–Server Model

The client–server model is a model consisting of a set of servers and clients. It is comprised of servers demanding services and servers providing services. There could be many client instances. Usually, the client–server model is implemented as a distributed system, effectively using the network system.

Hierarchical Structure

A system consists of many layers, and each layer can be defined to provide particular services. Once a hierarchical structure is applied, problem–solving becomes easy. If problems occur, they can be identified gradually in each layer, and equipment becomes interoperable by being standardized. The seven–layer structure of Open System Interconnection in a network protocol developed by the ISO is a representative example of a hierarchical structure.

03 Methods of Architecture Design Expression

Context Model

The Context Diagram describes the domains of a system to be developed and suggests the boundary and interface with the external environment. This regards the system as a big single process before the partitioning process, and shows the relevant input/output data. This model focuses on the interface between the system and the outside environment. In the initial analysis, analysts must first pay attention to what kind of data is exchanged between the user and the system, and what kind of exchanges occur between them. Once the system boundary is established, internal analysis of the system takes place to realize the system. This is analogous to prioritize goals for system development.

Component Diagram

Reusable components or parts that can be made by plugging into well made parts as in other fields to enhance the speed and productivity of software development are called 'components'. The foundation of this technology boils down to reusability, which is an acceleration technology to raise the speed and productivity of software development by reusing possibly the tested parts.

Components must communicate and cooperate with other systems or external devices. In order for components to be guaranteed with interoperability, standards on the implementation and documentation of components must be defined. Combination of components is a process to assemble components, and there are such types as sequential combination, hierarchical combination and additional combination.

Package Diagram

Commercial software developed for multiple users is called the 'package', and subsystems often appear in the form of packages. Once defined as the package, internal specificities of a package are hidden to minimize the interdependence between packages. A package consisting of subsystems is a gathering of functionally related classes.

A package diagram is appropriate to express the software architecture by representing the highest level of abstract subsystems as it shows the interdependence of subsystems. If the subsystems are divided to minimize the interdependence of subsystems, dependence among objects can be minimized and the complexity can be reduced.

Example Question

Question type

Short-answer question

Question

What kind of a structure is it among software architecture types where all the shared data used in a system is stored in a single spot so that all subsystems can share the data?

Intent of the question

To validate architecture types and characteristics

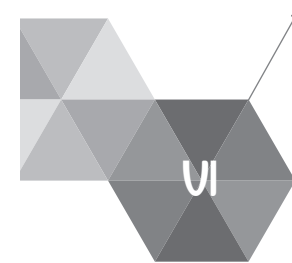
Answer and explanation

Repository structure

The essay-type answer is on the explanation of a repository structure.

Related E-learning Contents

- Lecture 6 Basic Design of Software



Object-oriented Design Process

▶▶▶ Latest Trends and Key Issues

The object-oriented development method has more applications due to its advantage of applying the identical methods and expression techniques in the entire software development process covering analysis, design and programming.

▶▶▶ Study Objectives

- * Able to explain the concept and principle of the object-oriented design
- * Able to execute static and dynamic modelling and express them in the Unified Modeling Language (UML)
- * Able to list the concept of design patterns and representative patterns

▶▶▶ Practical Importance Medium

▶▶▶ Keywords

- Object, class, encapsulation, inheritance, polymorphism, correlation, set
- Class, attributes, relationship, correlation, operation, class diagram
- Interaction diagram (order diagram, communication diagram), state diagram, activity diagram
- Singleton Pattern, Factory Method Pattern, Façade Pattern, Strategy Pattern

✦ Practical Tips Five Principles in the Object-oriented Design: 'SOLID'

- SRP (the Single Responsibility Principle)
Since one class has one method, there must be a single reason for modification. However, problems become complicated and larger in scale in the actual environment, and the actual use is challenging, so it would be better to apply in the form of utility.
- OCP (the Open Closed Principle)
In one class, it should be closed to modification but open to extension. This is related to polymorphism and abstraction, which are the key characteristics of the object-oriented design. It would be better to make improvement through inheritance instead of direct modification in improving features. Abstraction class must be implemented, considering whether or not the logical can be modified.
- LSP (the Liskov Substitution Principle)
It is the most important principle in designing for inheritance. It must be complied with since the IS-A relationship becomes valid for inheritance. In other words, it is a principle related to inheritance where a subclass can refer to a super class.
- ISP (the Interface Segregation Principle)
It is a principle on the exposed method where cross-object communication takes place through the interface of exposed objects. It must be designed to provide a different interface per user. One example is that the interfaces given to an ordinary user and a manager must differ.
- DIP (the Dependency Inversion Principle)
When there are two types of classes, a class that uses another class (Class A) and a class that is being used (Class B), the interface of Class A might be modified. In this case, Class A would have dependence of having to check out the modification. Even though the modification is small, it would be difficult to guarantee completeness and integrity in reality. Therefore, this principle aims to minimize modification possibilities by having the loosest design through interface-based abstraction.

01 Concept and Principle of the Object-oriented Design

In the object-oriented analysis, requirement analysis covers writing the class diagram, sequence diagram, and activity diagram. Requirement analysis usually describes the customer-oriented conceptual aspect without representing the physical side of how to store.

In the object-oriented design stage, the focus is on how to represent user requirements identified in the object-oriented analysis. In the design stage, specificities on implementation are suggested to set specific methods to enable physical implementation of requirements. The object-oriented design is advantageous in that the notations used in the object-oriented analysis are used again so that the design can be made without much modification.

Objects and Classes

A class is a group of similar objects. Objects refer to the things and objects in the real world that exist independently, e.g. Kim Cheol-su (an employees) and Seoul (a place). Each object is described by the group of each attribute, and is distinguished from other objects based on the attribute values. For instance, Kim Cheol-su would have such attributes and attribute values as the resident registration number (i.e. social security number) of '830425-1XXXXXX' and the date of birth of 'April 25, 1983'.

There are sets with similar attributes within a single system. In a university, there are many students, and they share similar information. They share the same attributes (including student ID number, major, GPA, etc.) and each student(object) has the exclusive value of each attribute. Here, the set of objects with the same attributes is called the 'class'. That is, an instance of a class is 'an object'. For instance, such objects as 'Kim Cheol-su' and 'Lee Yeong-hi' belonging to the same class share the same attributes (student ID number, major and GPA) and 'Kim Cheol-su' has '925462', 'CS' and '3.8' and 'Lee Yeong-hi' has '956234', 'Math' and '4.2' as attribute values, respectively for student ID number, major and GPA.

In validating the features of data required in a system, it is more convenient to describe a clustered class instead of describing each object. Clustering similar objects is called 'classification' of tasks. Objects clustered through classification share the same types of attributes, constraints and behaviors.

Encapsulation

The goal of designing is to make a software that is easy to understand and modify. The key lies in designing modules with a high independence. Software tends to show maturity when its components execute features with independence. Functional independence of software components is byproduct in the modularization and information hiding. Functional independence of modules can be maximized by enhancing the completeness of processing unit modules, and minimizing the dependence of cross-module processing.

Once critical concept in the object-oriented development method is encapsulation which can obtain abstraction and higher independence through information hiding, a protection method by grouping attributes and operations. Object-oriented languages have the feature of supporting encapsulation included in the languages.

Inheritance

Each class has its own attributes and operations, and there might be common attributes and operations found among many classes. Generalization is to gather the similarities among classes to define a new class. For instance, professors and students share the common attributes (e.g. resident registration number, name, address, and phone number). In this case, a class of professors and a class of students can be grouped together to define a generalized class of 'people'. In this case, 'students' and 'professors' can be called the subclass of 'people' and people can be called the superclass of 'students and professors.'

In this case, 'people' in the superclass have the common attributes and operations with the subclass of 'students' and 'professors.' 'Students' and 'professors' as the subclass have attributes and operations that are not shared among each subclass. A key feature found through generalization is that common attributes and operations are indicated in the superclass, and the information of the superclass is inherited to the subclass. Inheritance represented through generalization simplifies the definition of classes, enabling one to use the pre-defined class to easily define a new class.

Polymorphism

One feature of object-orientation is polymorphism. Polymorphism refers to a state where operations with the same name have different behaviors depending on the class, and the name of a single function or a single operator can be used for different purposes. Polymorphism in object-orientation is mostly used in the inheritance relationship, allowing for flexibility so that responses can be made in ways exclusive to each subclass for a single operation defined in the superclass. Polymorphism is a concept to invoke the subclass method (the serial order of tasks to be executed per message) through the superclass. It can be divided into overriding that redefines the method defined in the superclass in the subclass, and overloading that defines the method in multiple ways with different parameter types and numbers. On the side of sending a message, only superclass operations are invoked without having to know the objects are of what type(subclass) and depending on the types of objects in run time, the appropriate behaviors in the subclass are determined automatically. The run-time binding is to set behaviors through objects of the subclass in run time. Polymorphism and run-time binding are essential concepts for understanding design patterns.

02 Static Modeling and Dynamic Modeling

Static Modeling

Static Modeling is the concept which is the same as what was explained above. It is to validate static information of objects where the concept of time has not intervened. Information modeling describes data by validating the structure of database used in a system. An information model consists of relationships that validate basic objects required in a system and represent cross-object correlation.

The class diagram of UML is an information model representing a static information structure of a system. Class

Diagram as an information model representing a static information structure of a system is used to show classes needed for the system and their relationship. Each class consists of various attributes and operations representing characteristics of each object. Documents written in the previous stage can be utilized to find out classes, attributes of classes and the relationship among classes. The problem statement and use case scenarios can be used to elicit classes.

Dynamic Modeling

In the previous part, we found out classes consisting of a system, and identified the attributes and the relationship of classes. Now, let's take a look into how dynamic modeling is executed based on the outcome of static analysis. Dynamic modeling is a course of finding operations of classes with an interest in the changes in the states or behaviors of objects and interactions among objects. Static modelling, by contrast, focuses on a static structure of a system.

Dynamic modeling elicits operations of classes through inter-class interactions. Operations of classes are defined to execute features requested by messages of other objects. Normally, the Sequence Diagram of the UML is used to represent interactions among objects. The Sequence Diagram emphasizes how messages are communicated that are exchanged among objects according to the flow of time, and the Communication Diagram is the one that expresses changes in the cooperative relationship among objects and changes in messages. These two diagrams are called 'interaction diagrams.'

The Activity Diagram is a diagram to accurately understand the event processing procedure within classes. It is used to understand the processing of complicated processes or to validate additional operations of classes. It could be helpful to understand activities that occur by use case and the dependence among activities. When operations of specific objects are internally based on a complicated structure, they can be expressed, using the Activity Diagram of the UML.

There are also the Timing Diagram expressing the waiting time when it consists of the objects' waiting state and execution-entry state, and the State Machine Diagram expressing the changes in the state of individual objects.

03 Design Pattern

A pattern is a repeated solution when trying to solve problems. A software design pattern is a standardization of a pattern used frequently in certain circumstances in designing software. It is also a byproduct gained through experiences of developers for a good software design. This byproduct could be an optimized algorithm code or a good structure of classes or modules. The design of an application to be developed can be improved by using a design pattern that is suitable for certain circumstances. A design pattern aims for the reusability of software by improving the quality of a software code.

Singleton Pattern

The Singleton Pattern as a pattern related to the generation of objects guarantees that instances of a certain class

are the exclusive ones (restricting objects of a class into a single object), and provide ways to access this instance. Although the Singleton Pattern has a simple structure, it can be useful in the following circumstances. It is used when many of the objects processing the same resources or data do not have to be made unnecessarily. It is also useful upon having to consecutively do connection processing or thread management in a field preference management class in a program and a network program in the form of pool, and upon having to manage the number of same characters in a game.

Factory Method Pattern

When a certain process is needed to generate objects or the timing of generating objects is not clear, methods that generate objects can be used. Imagine that you make an application program where a file in a folder selected by a user can be read at a time wanted by him or her. Within the folder, there could be different types of files including document files, figure files and video files, and you can tell that there is a variety of programs that have to be connected as many as the number of file types.

Application program objects can be generated in advance so that once the user selects the file, he or she can read it immediately. However, as the number of files types increases, the number of application program objects that are to be generated in advance would go up, ending up as a program overusing memories unnecessarily. In this case, if the Factory Method patterns are used, application program objects can be generated at the time wanted by the user, and application programs can be made so that the user can generate and read the file objects he or she selected.

The Factory Method pattern would define an interface to generate objects, but the responsibility of generating instances is delayed so that the decision on generating what class of instances will be generated could be made in the subclass. This pattern is useful in cases where the base class does all (or most of) the tasks but postpones to the runtime on what objects are to be used.

Façade Pattern

A façade refers to the front side of a building facing a street or a space (a garden or a plaza, etc.), including the entrance as its front gate. The front gate is the area most frequented by people coming in and out, so there comes in the 'information desk' for a big building. Since it lobbies of a hotel or a theater are located toward the front gate, people naturally seek for the front gate when having to be guided in or outside a building or pay the bill.

The Façade Pattern is also located on the very front side of subsystems that are to be used by developers just like the information desk near the front gate of a building. It also plays the roles of enabling objects within a subsystem to be used. In other words, the Façade Pattern structuralizes the subsystem to reduce the complexity of a system and provides access to the subsystem as a façade object. Therefore, the GoF classifies the Façade Pattern as the one related to objects in its scope, which is a structural pattern given its purpose.

Strategy Pattern

The Strategy Pattern encapsulates each algorithm into a single class when various algorithms exist, and enables

them to be substituted. This allows transformations into various algorithms without impacting clients, so clients do not have to transform anything despite changes in algorithms.

Taxonomically speaking, the Strategy Pattern is a behavioral pattern made to distribute responsibilities by making each class in charge of algorithms, and is an object pattern in terms of scope because each algorithm can be changed into dynamic one when each algorithm is needed (in runtime). It is a pattern where the aforementioned concepts of polymorphism and dynamic binding have been applied.

The Observer Pattern belonging to the Behavioral Pattern like the Strategy Pattern is the one that automatically notifies and changes other objects in the dependent relationship when the state of a single objects changes.

Example Question

Question type

Essay-type/Long-answer question

Question

Explain 'inheritance' as a representative feature of object-orientation and its advantage.

Intent of the Question

To validate if one understands the characteristics of object-orientation and explain them

Answer and explanation

The representative attribute of object-orientation is that classes are divided into the superclass (parent class) and subclass (child class), and common attributes are to be reused by the subclass after defining them in the superclass. The subclass additionally defines individual attributes to utilize polymorphism.

Related E-Learning Contents

- **Lecture 6** Basic Design of Software
- **Lecture 7** Specific Design of Software
- **[Advanced] Lecture 4** Design Pattern (1) Creational Patterns
- **Lecture 5** Design Pattern (2) Structural Patterns
- **Lecture 6** Design Pattern (3) Behavioral Patterns
- **Lecture 7** Object-oriented Design (1) Static Modeling (Class Diagram)
- **Lecture 8** Object-oriented Design (2) Static Modeling (Component & Batch Diagram)
- **Lecture 9** Object-oriented Design (3) Dynamic Modeling (Sequence & Communications Diagram)

VII

Design Concept in the User Interface

▶▶▶ Latest Trends and Key Issues

The User Interface is the most important element in a computer system. Its importance is getting higher today as it significantly impacts the user satisfaction with its user-oriented planning and designing.

▶▶▶ Study Objectives

- * Able to understand and apply the design principle for the User Interface
- * Able to explain the Human Computer Interaction (HCI)
- * Able to understand components of the Graphic User Interface (GUI) and apply them appropriately

▶▶▶ Practical Importance Medium

▶▶▶ Keywords

- UI/UX
- HCI (Human Computer Interaction)
- GUI (Graphic User Interface)

✦ Practical Tips Poorly Designed UX Causing Casualties

In the medical equipment field where software directly impacts the life of humans, small defects in medical equipment can bring about enormous outcome, which makes the field more sensitive than any others. When asked which one would be more deadly among physical defects of medical equipment or operational errors of users, most would answer it would be the former, but in fact, it is the opposite. If there is no intuitive explanation about where to attach a pad of the Automated External Defibrillator(AED), taking actions for an emergency would fail or the result might turn out to be the total opposite. These possibilities evidence the importance of UX.



A patient died of excessive radiation exposure due to a defect in Therac 25, a radiation therapy equipment developed in the early '70s. According to the accident analysis, a wrongly designed error message was generated in the process of upgrading that resolved the inconvenience of having to additionally deal with physical setting and subscription processing through manager consoles. This led to the tragedy of the patient death in November who was subject to excessive exposure in radiation treatment at the Ontario Cancer Foundation in July 1985. It was because the radiation therapist was insensitive to the unknown error message of Therac 25 (unable to clearly distinguish the numbers that come after the same message). As the abnormal stop was repeated five times in the course of the patient therapy, the patient ended up being exposed to excessive radiation.

Many companies have recently maximized the customer usability by utilizing the information on the limited screen size in mobile business via smartphones, through which securing more customers and expanding sales. More and more companies tend to acquire specialist units to address a bunch of concerns over UI/UX to provide more customer-oriented invaluable services.

01 Design Concept and Principles of User Interface (UI)

The word 'interface' literally refers to the part that 'faces' with each other 'between(inter)' two objects. The UI is analogous to the window of software, referring to a device or a software enabling a proper interaction between the user and the system. One example of the UI is input/output of a command using a keyboard or a mouse, or selection of a menu to use a particular software. Even with a superior system, the product value would dramatically drop if the UI as the contact point where the user uses the system is inconvenient or difficult to use.

Designing of the interface must have the following points in mind:

A Need for Consistency

The UI must be made to be consistent but in developing a big system, it would be difficult to maintain consistency because many people design and implement the UI. No consistency among UIs will cause a big confusion among users. In order for the UI to be consistent, there must be a standard on the UI before developing it, and even after the development, the UI must be checked to correct errors.

User-oriented Design

The UI is to for the sake of users, so it must be designed in a user-oriented manner. It must be made so that users can control the interface. To this end, input must be simple, and the UI must be suited to user attributes and easy to remember. The output scheme must also be easily understandable and friendly. The UI must be designed so that users can easily learn the 'input and output language' of the interface.

Feedback

The UI must provide meaningful feedback. If users press a wrong button or are to execute an operation wrongly, it must be easily identifiable. Visual or sound feedback helps mutual communications. For a data error, it must be explained where the error occurred in the data instead of notifying with a message that says, "You entered wrong data." The error can be easily corrected as the cursor indicates the wrong data.

Identification of Destructive Actions

It must be checked out if users are to delete data or files before they execute the actions. Normally, data are correlated with other data, and if referential integrity is required, other data might be subject to deletion, which might not be known to users. The UI must be designed so that users do not induce any critical errors. One useful way is to use the 'Undo' feature.

02 HCI (Human-Computer Interaction)

Concept of HCI

Any system can be divided into two parts: one part performing features and the other part in charge of the interface. Software design can also be divided into two: the one on design operations and the one on exchanges with users. The internal design is the part about computer operation, while the external design is the part about exchanges with users. The two designs have the equal importance, and neither of them must be neglected. The UI, in the users' perspective, is the way of conversation between users and their computer, that is, the input/output language of the software for users.

Types of HCI

The UI comes in various styles, which must be properly utilized to design a good interface. This section looks into common styles of the UI.

① Command Language

The command language used to be a common interface type in the initial period when the computer system was adopted. The command language is a formal language like a programming language. For a system operated by the command language, users type in the command language on the keyboard to talk and exchange with the computer. A specific command language is provided per behavior of the computer system, and various options are offered to specify the same types of behaviors. Users that are to use the command language can form commands with the command name and options to execute necessary features.

For instance, the UNIX system provides following commands:

- Display or print the description of 'cat' files
- Change the task list for cd and chdir.
- Change the access permission of the chmod file or the list
- Copy the cp file.

② Menu-driven System

A menu consists of items, and users execute necessary features by selecting an item on the menu. Menu selection is an appropriate interaction method for beginners or intermediate-level users. The menu-driven system is currently used widely. There is no need for users to remember each item, and as long as they can judge what features each item executes, they can use the menu system. Since users select items on the menu system and execute necessary characteristics, keyboard typing is not necessary and errors that might occur in entering a command can be eliminated. The menu system provides a variety of menu items. Designers can design a good interface by utilizing these diverse menu formats.

In the full-screen menu, users whose menu fills up the whole screen have to select menu items to execute the

following features. The following items show the menu of a specific system:

- System management
- Component management
- Product structure management

③ Form Fill-in

When there is a lot of data to be entered, the Form Fill-in Interface can be used, which provides various fields depending on the information to be entered. The Form Fill-in Interface collects related items on one screen, and names the screen representing the whole. Users can directly type in relevant information on the fields, and each field has its own field name. The field name is intuitive, and is easy to use since its format is similar with the one used in daily business. It is also easy to learn since there is not much difference with a manual task. Therefore, the Form Fill-in Interface is appropriate for beginners, and for entering much information concurrently. This type of interface is used a lot in the database system or MIS system.

④ Direct Manipulation

The Direct Manipulation Interface supports the execution of tasks to be achieved by showing a simple work environment and directly manipulating objects. Tasks to be done can be expressed in simple pictures for access at ease, and the Direct Manipulation Interface is commonly found in software using the mouse as in the Window System.

Icons must be made to be easily understood in designing the Direct Manipulation Interface, being suited to purposes. Users must be designed to avoid wrong predictions because icons allow users to guess what tasks are to be performed. Expectations about icon characteristics might differ by culture and custom, requiring caution in dealing with icons.

03 Components of the Graphic User Interface(GUI)

Let's take a look at the Graphic User Interface (GUI) widely used today. There are diverse benefits in using the Window in the UI: other types of information can be concurrently seen, and users can see the screen by exchanging it with one another. Therefore, many tasks can be executed by using the Pull-down Menu. Moreover, the amount of entry can be reduced and mutual exchanges with the system can be efficiently performed. The components supporting the GUI are as follows:

① Menu Bar

The Menu Bar is the part showing such menus as 'Files', 'Edit' and 'Help' that are horizontally listed on the top part of the screen.

② Tool Bar

The Tool Bar consists of icons for frequently used commands of users in such application programs as the Window or the Web Browser.

③ Command Button

There are buttons such as 'OK' button and 'Cancel' button waiting for users' instruction, while executing commands imposed by users on a computer program to start, finish or control some behaviors or features.

④ Toggle Button

This button is used to select one out of two states like the on/off button.

⑤ Dialog Box

It is a special type of the Window indicated by a system or an application program to accept some reactions or entry from users.

⑥ Text Box

It provides a message or a place for users to type in.

⑦ List Box

It shows a list which users can select from and is also called the 'Combo Box.'

⑧ Radio Button

It is used when only one of many options can be selected, and what is selected is indicated with a black dot.

⑨ Check Box

When more than one options can be selected in a group, it can be ticked with a mark(v) in the Check Box.

Example Question

Question type

Descriptive question

Question

Describe the Characteristics of the Form Fill-in Interface among the types of the User Interface.

Intent of the question

To distinguish the types of the User Interface, and understand the features for each type

Answer and explanation

Users can directly enter the relevant information in fields, and each field has its own name, so it is intuitive. Due to its similarity in its format with the one used in daily business, it is easy to use and learn. Therefore, the Form Fill-in Interface is appropriate for beginners, and also to enter much information concurrently. This type of interface is used often in a database system or the MIS system.

Related E-learning Contents

- **Lecture 6** Basic Design of Software

VIII

Programming Language & Code Reuse and Refactoring

▶▶▶ Latest Trends and Key Issues

Many companies use Java and C languages because software and app development has been highly demanded in mobile/cloud recently. These two languages have been mostly used by developers as two major languages in programming for long. These languages serve as the basics in acquiring other programming languages, being used in many business fields, so it would be critical to acquire relevant skills.

▶▶▶ Study Objectives

- * Able to explain the characteristics of programming languages (unstructured language, structured language, object-oriented language)
- * Able to compare the characteristics of major programming languages (C, C++, Java, Node.js)
- * Able to explain the concepts of code reuse and refactoring and representative refactoring techniques

▶▶▶ Practical Importance High

▶▶▶ Keywords

- Programming languages, compiler, interpreter
- Code reuse
- Refactoring, code smell

+ Practical Tips Selection of Programming Languages

Criteria to select programming languages to develop products or services are determined in various ways in the actual business environment. Such criteria include the hardware, OS environment, solutions that are adopted, business characteristics, application of recently developed languages and status of personnel being capable of development. Despite the high usage of the languages, if there is a small number of their developers whose labor cost is high, companies, in fact, would seek for alternative languages.

For instance, if Java is selected for all implementations in a project where systems are set up requiring a lot of batch-type work, users would confront the severe issue of enhancing the performance of batches. If C language is selected only to raise the speed, there would emerge the issues of productivity and a very severe test coverage. Therefore, selection is made based on past experiences in many projects, and even in the same project, various languages and architectures are used. In recent development projects where mobile services have become essential in many areas, the complexity and diversity increase to a great extent.

Software Maintenance and Refactoring

The bottom line in software maintenance and operation units is to minimize failures and to increase the quality of modification development reflecting customer requirements demanded in the changing environment. In the recent business environment, many next-generation projects are implemented for innovating fundamentals and improving the IT service quality through improved business processes, and for executing timely projects to enhance productivity. At a time when such large projects come to an end, the handover process between development units and maintenance units must be cautiously and promptly implemented.

The operation and maintenance units that are handed over with new tasks would experience the aging of systems and a continued increase of complexity in processing logics amid the continued process of reflecting and modifying new requirements responding to the market changes. Against this backdrop, refactoring normally takes place in such forms as prevention of failures and improvement in tasks to achieve a higher performance.

01 Characteristics of Programming Languages

Concept of Programming Languages

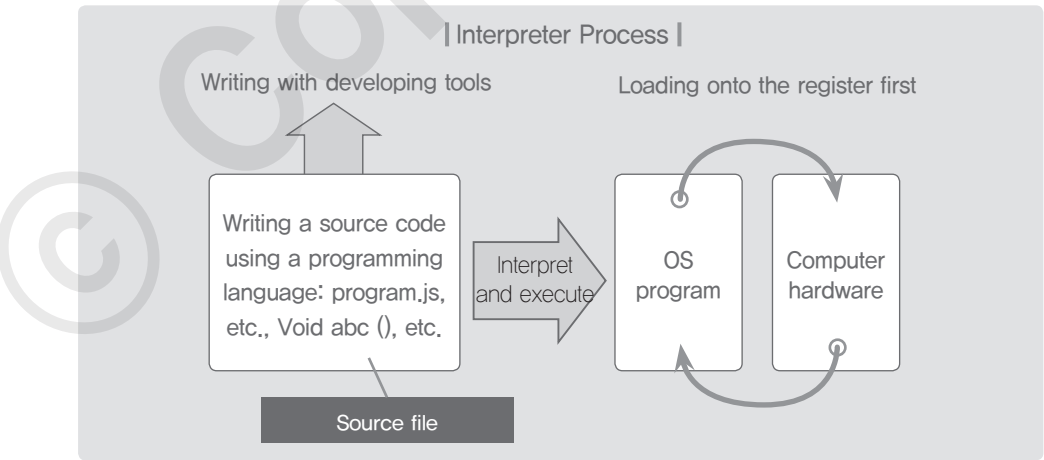
Programming languages: These languages enable users to do programming using languages that are similar with daily languages. There is a compiler or interpreter for a computer to understand languages because programs written in each programming language are interpreted in a machine language. Moreover, each programming language has a different usage, so it is essential to secure target productivity and quality by using appropriate programming languages in each case.

<Table 15> Areas of Usage of Programming Languages

Areas	Major Programming Languages
Flat science & technology (for arithmetic)	FORTRAN, ALGOL60
For business	COBOL
For artificial intelligence	LISP, PROLOG
For system programming	PL/S, BLISS, ALGOL, C/C++, Java
For special purposes	GPSS (General Purpose Simulation System)

Interpreter Language

The interpreter directly changes a source program into a low language without an intermediary process. Instead of making a machine program by interpreting a high language up to the intermediate code and executing the interpreted code using a software interpreter, subroutines which are the features of each intermediate code are invoked for execution. Examples of interpreter languages include LISP and PROLOG.

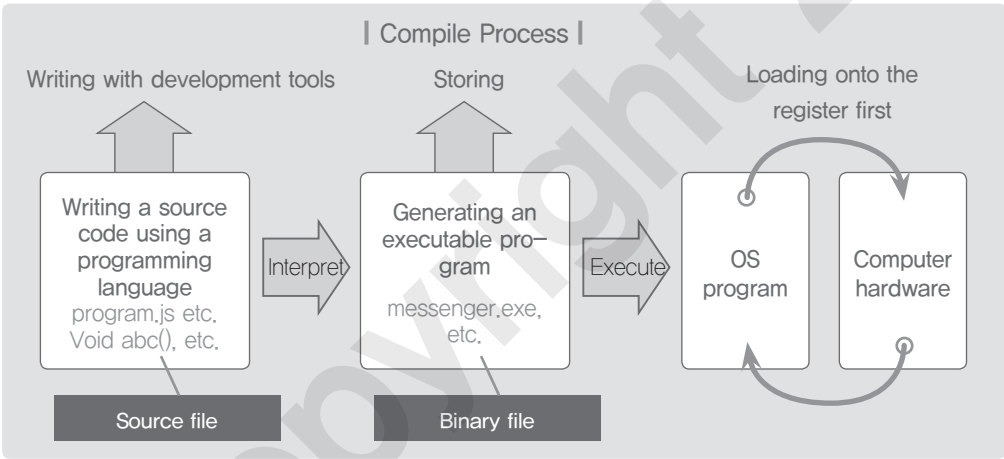


<Figure 15> Interpreter Process

- ① Advantages of Interpreter Languages
 - Executable whenever needed without having to wait for the complete interpretation of a machine languages
 - Saving memories because the form of source language is maintained until a program is executed
- ② Disadvantages of Interpreter Languages
 - Requiring a lot of time to decode a source program upon each re-execution.

Compiler Languages

A compile refers to modifying a certain language into another that has the same meanings. It is to make object modules by interpreting a high language into a machine language, a low language through a compiler and link, load and execute these modules. Examples of compiler languages include FORTRAN, PASCAL, C/C++.



<Figure 16> Compiler Process

- ① Advantages of Compiler Languages
 - Possible to store interpreted object codes
 - Re-executable after compiling
 - Shortening the execution time through fast re-execution after compiling once in a program that is reused
- ② Disadvantages of Compiler Languages
 - Consuming much time to convert into a machine language
 - Causing the waste of memory by interpreting one line of a source program into hundreds of lines of a machine language

02 Characteristics of Major Programming Languages

C (Programming Language)

① Introduction of C

C was developed by Dennis Richie at AT&T Bell Labs who worked on UNIX OS in the '70s. Most of the computer programming languages had their motif from C, and famous OS' such as iOS(iPhone) and Android mostly consist of C.

② Characteristics of C

- High execution speed and ease of use in managing memories fast and promptly
- Possible to implement features with paragraphs due to simple notations
- Procedure-oriented language: program execution is done according to a fixed order
- A program considering arrays, and memories, etc. is needed, so more difficult than Java
- Difficult to port upon changing the execution environment and machine

C++

① Introduction of C++

C++ is suited to system programming because it includes most characteristics of C, and also suited to object-oriented programming because of such characteristics as class, operator overloading and virtual features. If software is developed based on object-oriented programming techniques, the reusability, scalability and maintenance of software would increase, so it is suited to writing large programs, and has high readability.

② Characteristics of C++

- Object-orientation: It reflects the characteristics of object-orientation as a language developed to support object-oriented programming.
- Encapsulation and data hiding: Class as a feature of C++ is commonly used in an encapsulated form, and the structure of the internal mechanism is hidden, so users don't have to know how the class behaves and only have to know how to use it.
- Inheritance and reusability: C++ has reusability through inheritance. Inheritance is a process of making common characteristics in one class, extending them and making a new class. Program reusability is enhanced through inheritance, and productivity of software can be raised.
- Polymorphism: It means that a single name has many forms. For instance, a person named 'John Smith' is not only an office worker in a company but also an instructor in a private institute at night.

Java

① Introduction of Java

James Gosling at Sun Microsystems started to develop Java, a simple debugging programming language controlling household appliances in 1991. It started off as C++ as a language that improved many pitfalls of C++. Yet, with the spread of the Internet in the name of World Wide Web or WWW in 1995, it was applied to the Internet by Hot Java, so a strong output form called 'Applet' gained popularity. Java as a small and simple structure is efficiently changed and executed. Its advantages have been recognized because many features causing errors in the existing languages have been complemented. Java is now the mostly commonly used programming language in apps and on mobile devices. These days, Java is used and developed to make applications for household appliances and Android.

② Characteristics of Java

- Java generated from C++ being grammatically similar with C/C++
- Complexity of C++ being simplified
- Always executing automatic garbage collection
- Object-oriented: a perfect object-oriented language applying the object concept
- Java program being independent on a platform by being executed by a virtual Java machine

Node.js

① Introduction of Node.js

Started to be developed by Ryan Dahl in 2009, it is currently one of open source projects. As a network server supporting the Async/Non-blocking IO of high performance, it behaves based on single threads. It uses programming models based on JavaScript and events. Many Internet companies including LinkedIn, Paypal and Groupon have recently adopted Node.js, converting the internal system.

② Characteristics of Node.js

- Its development structure based on JavaScript has become very simplified, allowing for fast development. In other words, developers that used to develop the front end on the client side through JavaScript can easily do server programming. Companies do not have to divide technical sets of front-end and back-end engineers. Therefore, it has an easy learning curve and technical integration on front-back ends is easy.
- Socket.io-based web push can be easily implemented. Appropriate push methods can be automatically selected and used depending on the types of web browser. This is not a complicated consideration due to abstraction within Socket.io API, but only Socket.io is used. Therefore, developers can implement the push service in such a simple manner.
- Possible to make resources efficient by supporting the non-blocking IO model. Instead of waiting for a response in a waiting state per IO request, the response is processed at a point when events are generated for the completion of IO processing. As a result, threads or processes can be effectively executed and scheduled, enabling resource efficiency.

03 Code Reuse and Refactoring

Concept of Code Reuse and Refactoring

① Outline and forms of code reuse

Code reuse is to use a part of or the entire program written at a certain point to make a different program later. Code reuse is a typical technique to save time and energy consumed in redundant tasks. A library is a good example here.

Companies choose to extract a part of or entire codes from the existing program and copy them to a new program for the sake of fast development. However, a repetition of copy and paste might lead to duplicate coding later.

Methods to accelerate and facilitate reuse include object-oriented programs, generic programming, automatic programming and meta programming.

② Considerations in reuse

- Software development process based on systematic reuse
- Training and quality enhancement for better reusability
- Setup of a reuse environment through initial investment
- Continued improvement and reinforcement of libraries
- Reuse backed by tools
- Evaluation of and scales for software productivity
- Management of information set for reused components
 - Output of targets to be reused in software
 - Architecture, source code, data, designs, documents, estimates (templates) Human interfaces, plans, requirements, test cases
- Composition-based, component-based development Model

③ Outline and Definition of Refactoring

- Code refactoring is to modify the internal structure of software without changing the external behavior of a code to make it easy to understand and change it with little cost.
- There are following reasons that challenge modification when software developers do maintenance for programs.
 - Low readability
 - Duplicated logic
 - Complicated conditional statements
- Refactoring is executed for the following reasons:
 - Improving the software design
 - Raising software readability
 - Making debugging easier
 - Raising program productivity and quality

④ Timing and Procedure of Refactoring

- Timing is important for refactoring as seen in the rule: when coding is done first, it is done from scratch, and when done the second time, it is coded duplicately, and for the third time in doing a similar task, refactoring is done.
- Refactoring is mostly executed when timing is inefficient to add new features to a code, and upon having to change a design in a form that cannot be easily added, debug or review a code.
- Procedure of Refactoring

〈Table 16〉

Target Selection	Maintenance, Inspection, XP Methodologies
Unit configuration	Composite configuration with mentors
Execution control	Change management, configuration management, CCB control
Execution Techniques	Design pattern, AOP execution
Test	Unit/Integration Test, Regression Test
Outcome Compilation	Documentation, update, application of management systems

- In refactoring, when after a small modification (single refactoring), the presence of behaviors is tested and behaviors are executed, users would move to the next refactoring stage: if behaviors are not operated, problems are resolved, and what has been refactored is undone so that the system could be in operation.

Key Refactoring Techniques

① Code Smell: Low readability or duplicated logic in codes

② Causes:

- Programs with low readability of codes (comment, indentation, names of variables/features)
- Programs with duplicated logics (Copy & Paste)
- Programs where complicated conditional statements are included (excessive case statements)

③ Types of Code Smell and major refactoring techniques as solutions

- Duplicated Code, Switch Statements, Comments
- Long Method, Large Class, Long Parameter List, Lazy Class, Incomplete Library Class, Data Class
- Primitive Obsession, Temporary Field
- Parallel Inheritance Hierarchies and Message Chains, etc.

Example Question

Question type

Descriptive question

Question

Code refactoring is the maintenance of convenience, improving the internal structure by raising readability of complicated sources by maintaining the execution outcome of computer programs, thus increasing scalability to enhance efficiency.

Explain the field encapsulation among code refactoring techniques.

Intent of the question

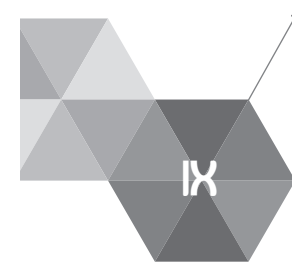
To validate the understanding of types and characteristics of code refactoring techniques

Answer and explanation

Field encapsulation means to hide fields (variables), that is, to access using such methods as Set~ and Get~ without a direct access. This can prevent a mistake of directly configuring a wrong value and help more efficiently in responding to changes in data types. However, having to go through a function would slow down the speed, but in a language such as C++, such features as inline features are provided to enable implementation without speed reduction.

Related E-learning Contents

- **Lecture 8** Programming Language
- **Lecture 9** Code Reuse and Refactoring



Software Testing

▶▶▶ Latest Trends and Key Issues

The portion of efforts and cost for software testing in developing software has recently gone up. IT companies recognize the importance software testing to the extent of running a quality assurance (QA) team or department specialized in software testing.

▶▶▶ Study Objectives

- * Able to explain the concept of testing and compare test case designing methods
- * Able to explain levels (types) and purposes of testing

▶▶▶ Practical Importance High

▶▶▶ Keywords

- Testing process, testing types
- Specification-based Technique, Structure-based Technique, Experience-based Technique
- White-box Testing, Black-box Testing
- Software build and distribution

01 Concept and Process of Testing

Concept of Testing

Testing is a method to detect defects to validate or verify if the behaviors, performance and stability of applications or systems satisfy the expectations of users or customers. General principles of testing are as follows:

- ① Activities to detect the presence of defects
Testing is to detect the presence of defects. It is almost impossible to proof the integrity of software.
- ② No perfect testing
Even for a very simple program, it is impossible to do the testing for all cases.
- ③ Testing to start in the initial development stage
This will shorten the development period, prevent defect prevention activities, and ultimately, reduce cost.
- ④ Pesticide Paradox
According to the Pesticide Paradox, if the same pesticide is sprayed, insects will be immune to it and the pesticide will no longer become effective. Likewise, if the same testing is repeated, it will fail to discover new bugs any more.
- ⑤ Testing being context-dependent
Each area would require different approaches, priorities and the level of severity.
- ⑥ Absence-of-Errors Fallacy
If a developed system fails to cater to users' need and expectations, it will become futile, and the process of finding and debugging is meaningless.

Testing Process

Testing must be adjusted and managed based on a process that covers various components. The testing process and its major activities are as follows:

〈Table 17〉

Process	Major Activities
Test Analysis and Design	<div><div>• Reviewing the test basis</div><div>• Validating testing circumstances/requirements/data</div><div>• Allocating testing techniques</div><div>• Evaluating testability</div><div>• Establishing the testing environment</div></div>
Test Implementation and Execution	<div><div>• Specification of test cases: prioritization, data generation and procedure writing</div><div>• Pre-testing</div><div>• Executing testing and recording results</div><div>• Comparing with expected results</div></div>
Evaluation and Reporting of Exit Criteria	<div><div>• Validating if exit criteria are met</div><div>• Writing the first testing report</div></div>
Test Closure Activities	<div><div>• Validating the output and storing testware</div><div>• Evaluating the test process</div></div>
Test Planning and Control	<div><div>• Setting the test purpose/goal and researching targets</div><div>• Developing test strategies and analyzing risks</div><div>• Establishing strategies and exit criteria</div></div>
Test Estimation and Unit Formation	<div><div>• Test planning and controlling test management</div><div>• Reporting and report planning/design</div><div>• Progress reporting</div></div>

How to Design Test Cases

In designing and specifying test cases as the essence in test designing activities, various designing techniques can be used. Designing techniques are broadly divided into Specification-based Technique, Structure-based Technique and Experience-based Technique as in 〈Table 18〉

〈Table 18〉 Test Designing Techniques

Type	Technique	Explanation
Specification-based Technique	Equivalence Partitioning	A test case designing technique to execute as representative values in the equivalently partitioned domains
	Boundary Value Analysis	A technique to design test cases even including boundary values because it is highly probably to defect defects in the entry values belonging to the boundary of equivalence partitioning
	Pairwise Testing	A designing technique to make a table so that each value needed for testing could form a pair at least once with other values, and executing testing accordingly
	Decision Table Testing	A designing technique for a test case to test a pair of the entry value indicated on the decision table and a stimulus (a cause)
	State Transition Testing	A technique to design the relationship between events, actions, activities, states and state transition based on the State Transition Diagram
	Use Case Testing	A technique to invoke test cases from use cases when a system is modelled as a use case

Structure-based Technique	Control Flow Testing	A designing technique to enable testing of the all possible event flow (path) structures upon execution via components or systems
	Coverage Testing	A test case designing technique to achieve the coverage to the extent where the structure of systems or software is tested by the Test Suite
	Elementary Comparison Testing	A test designing technique of invoking test cases so that testing can be done for pairs of entry values by using the concept of Modified Condition/Decision
Experience-based Technique	Exploratory Testing	A technique to design informal tests using information obtained while executing tests in order to proactively control the test design for testers upon testing and design new and better tests
	Classification Tree Method	A designing technique to execute test cases expressed as the Classification Tree by pairing and executing the representative values of input/output domains

- It is to detect defects within frequently used and testable software (module, program, object, class, etc.) in unit testing, and verify relevant features. Control Flow Testing, Condition/Decision Coverage Testing and Elementary Comparison Test are conducted using the source code.
- For integration testing, structural approaches can be used and applied.
- There are two types of analyses: the static analysis on the internal structure of an implemented source code and find out predefined errors; and the dynamic analysis to find out possible errors in an actual circumstance by executing a program.
- Types of representative techniques for the White-box Testing are as follows:

〈Table 20〉

Type	Description
Structural Technique	To measure and evaluate the logical complexity of a program
Loop Test	To conduct the test limited to the loop structure of a program

② Black-box Testing

- It is also called the ‘functional test’ or ‘specification-based test’.
- It is mostly used in system testing, verifying both functional and non-functional requirements. Functional requirements are verified based on the specification, and the system testing on non-functional requirements is based on the defined specification for non-functional testing of performance, availability and security, etc.
- The testing is focused on whether or not software operates according to the requirement specification.
- The internal structure of software modules/components or systems is not referred to. Functional and non-functional testings are carried out based on the requirement specification and the interface with the outside world.
- Types of representative techniques for the Black-box Testing are as follows:

〈Table 21〉

Type	Description
Equivalence Partitioning	• Testing by selecting testing cases with various input conditions e.g. testing by partitioning into $x < 0$, 0 , x , 100 , $x > 100$ in the scope of $1 \sim 100$
Boundary Value Analysis	• Testing the correctness of outcome based on the boundary values e.g. testing with $x=0$, $x=100$, $x=-1$, $x=200$, etc. in the scope of $1 \sim 100$
Cause-Effect Graphing	• Detecting errors by graphing the impact of input values on the output values
Error Guessing	• Detecting negligible errors based on senses and experiences e.g. checking out without entry values and entering the numbers in the text entry boxes

02 Testing Types and Techniques

Testing Types

Software testing types vary by different levels as in 〈Table 19〉, and in principle, testers, the purpose and the environment must be considered in each type.

〈Table 19〉 Testing Types

Testing Type	Purpose	Tester	Environment
Acceptance Test	Validating the conformance to requirements	Users	User environment
System Test	Validating entire functional and non-functional tests in an environmental similar with the actual one	Testing organization	An environment similar with the actual user environment
Integration Test	Detecting defects in an interface between unit modules	Development/testing unit	Development/Testing environment
Unit Test	Detecting defects within unit modules	Development unit	Development environment

Testing Techniques

For testing techniques, there are the White-box Testing and Black-box Testing whose characteristics are as follows:

① White-box Testing

- It is also called the ‘structural test’ or ‘code-based test.’

03 Software Build and Distribution

Software Build

The software build refers to a process of transforming source code files into an independent software workpiece that can be executed in a computer, or the corresponding outcome. It is one of the software quality assurance activities, and the daily build facilitates the reduction of integrated risks, prevention of low quality, analysis of early defects, progress monitoring and morale boosting among developers.

① Daily Build and Behavior Test

- It is a process to re-compile the entire software products every day, and undergo a series of necessary tests to verify basic behaviors.
- The daily build can reduce such risks as failure in software integration, low quality and low project visibility, thus raising project efficiency and satisfying customers.
- The daily build and behavior test can be used in all projects, being relevant for big and small projects, OS, off-the-shelf software and business systems.

Software Distribution

Software distribution refers to all acts of making a software system to be used, and the distribution process consists of possible transformations among them along with interactions. Such activities can occur on both producer and consumer sides. Because distribution is exclusive to all software systems, it is difficult to define an accurate procedure within each activity. Therefore, 'distribution' must be interpreted as a general procedure that can be defined depending on the specific requirements or features of customers or users. Types and activities of software distribution include release, installation and activation, inactivation, adaptation, update, built-in, version tracing, deletion and retirement.

Example Question

Question type

Performance question

Question

Explain the concept of the Equivalence Partitioning of Black-box Testing, and suggest test case types for input conditions of the information of elementary school students given in the [Example] using the Equivalence Partitioning.

[Example]

- Ⓐ For gender distinction, we use these numbers for boys(1) and girls(2).
- Ⓑ The range of school years is from first to sixth grade.

Answer and explanation

Input conditions are partitioned into over two equivalence types to design test cases in the Equivalence Partitioning, which is useful to determine the number of test cases depending on the given input conditions.

Minimum test cases are written for each test case type:

- Ⓐ Gender=1 or Gender=2, Gender<>1 and Gender<>2: 2 types
- Ⓑ 1<=Grade<=6, Grade<1, Grade>6: 3 types

The Equivalence Partitioning is divided into valid equivalence and invalid equivalence domains, and normal value and abnormal value domains are distinguished to be compiled in the conditions. For particular code values like gender, which are not intervals, it would be mostly two types, and if there are interval values, they can be diversely partitioned depending on conditions.

Scoring criteria

The concept part must include input conditions and over two equivalence partitioning keywords. If there are two or more keywords, and the concept summary is properly done, there is one or less keyword whose full score is 20 points. If the concept summary is properly done, and the 15-point concept summary is not done well, the score is 5 points, and if not, 0 point.

Scoring based on the 15-point full score per test case question.

If a) is the same as the correct answer, 15 points.

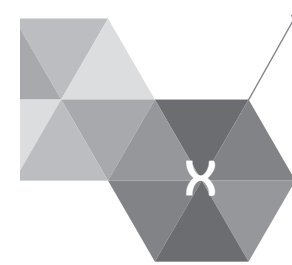
If it is written with 1<= gender <=2, gender < 1, gender > 2, 10 scores, and if it is other than these, 0 point.

If b) is the same as the correct answer, 15 points. If it is similar with the following, 10 points, and if it is other than these, 0 point.

Class Number=1 or Class Number=2 or Class Number=3 or Class Number=4 or Class Number=5 or Class Number=6 or Class Number=7 or Class Number=8 or Class Number=9

Related E-learning Contents

- Lecture 10 Software Testing



Software Maintenance & Reverse Engineering and Re-engineering

▶▶▶ Latest Trends and Key Issues

About 70% of the application lifecycle cost in IT companies is spent in the software maintenance stage. Therefore, a lot of attention is on systematic software maintenance. Activities are proactively underway to extend the actual software usage period through systematic management, and obtain high-quality software by adopting reverse engineering and re-engineering.

▶▶▶ Study Objectives

- * Able to explain the concept and types, process and activities of software maintenance
- * Able to explain the concepts of reverse engineering and re-engineering

▶▶▶ Practical Importance High

▶▶▶ Keywords

- Concept of software maintenance
- Software maintenance process
- Reverse engineering, re-engineering, re-structuralization, re-documentation

01 Software Maintenance

Definition of Software Maintenance

Software maintenance is the last stage of the Software Development Life Cycle (SDLC) as an operation-oriented work stage extending the life of software, referring to a series of works that correct errors, clarify the original requirements and enhance features and the execution capacity. It is the entire software engineering work to enable the removal of defects, enhancement of performance and adaptation to a changed environment after software is delivered to customers.

Purpose of Software Maintenance

The purpose of software maintenance is broadly to improve the software performance and repair damage, and port, correct and prevent damage to enable actions in a new environment. As mentioned above, software maintenance is a critical stage in the efficiency of software lifecycle due to the following reasons:

- A dramatic growth in the maintenance cost even more so than the existing development cost in the total budget for software development and maintenance
- Increases in management tasks including documentation due to highly complicated software
- A greater emphasis on the importance of management due to a long period of use compared with the development period
- Increases in maintenance amid the spread of package development compared with service development

Types of Software Maintenance

〈Table 22〉

Classification Criteria	Type	Description
Reason-based Maintenance	Corrective Maintenance	Error-driven maintenance, processing, execution and implementation, and error identification
	Adaptive Maintenance	Processing for adaptation to changes in a data environment and an infrastructure environment
	Perfective Maintenance	Maintenance for the addition, modification and quality of new features
Time-based Maintenance	Scheduled Maintenance	Periodic maintenance
	Preventive Maintenance	Maintenance for prevention
	Emergent Maintenance	When ratification of maintenance is needed

Target-based Maintenance	Data/Program Maintenance	Processing when needed such as data conversion
	Documentation Maintenance	Processing when needed such as modification of document standards
	System Maintenance	Maintenance of systems

Procedure of Software Maintenance

Software maintenance is usually conducted in the following order:

〈Table 23〉

Maintenance Procedure	Description
Understanding the currently used software	• Program structure analysis, variable/data structure, applications, working knowledge
Requirement analysis	• Types of maintenance, strategy setup (modified/new) • Identification of targets such as changed programs
Validating and modifying the scope of impact	• Impact of the existing function due to changes in software • Modification and correction of programs
Testing/Maintenance	• Document correction, configuration management, maintenance

The ‘requirement analysis’, a major activity in the order of maintenance mentioned above, is to promptly validate how much the demand for maintenance impacts the currently used system. The scope of impact can be easily identified depending on how much ease of maintenance has been already acquired for the software in the existing system. Or, it might require enormous efforts as seen in the above example. Ease of maintenance represents the level of difficulty in software maintenance, and can be divided into following types:

- Ease of use: A feature that represents how easy it is to understand an application or a program
- Modifiability: A feature that represents how easy it is to modify an application or a program
- Testability: A feature that represents how easy a process is to show the correctness of an application or a program

Therefore, if systematic activities take place to improve the ease of maintenance as follows in developing software, user or customer requirements can be continuously met despite changes in the environment, and economical responses can be made to the rapidly rising maintenance cost.

- Analysis Activities: to decide on customer requirements and constraints, and reveal the validity of products to be developed
- Standard and Guidelines: to prescribe various types of standards and guidelines to enable the ease of maintenance
- Design Activities: to emphasize and utilize clarity, modularity and modifiability

- Implementation Activities: to apply the standard structure and coding style
- Supporting Documents: to require guidelines and test manuals needed for maintenance activities

Types of Software Maintenance Units

IEEE/EIA 12207 defines maintenance operators as a unit that performs maintenance activities, sometimes referring to individuals involved in maintenance to distinguish them from developers. Maintenance operators can obtain much knowledge from developers on the software. If maintenance operators meet with developers, or are involved from the very beginning, it might be helpful to ease the maintenance burden. When software engineers have been transferred to other tasks or are not in touch, it would make things more difficult for maintenance operators, who are supposed to support after being handed over with products that are developed (e.g., codes or documents), and to gradually evolve/maintain them throughout the software lifecycle.

The following <Table 24> represents various actors per stage of general maintenance. Each actor conducts given maintenance activities and is accountable for them. The Maintenance Management Board makes the final approval or rejection on a need for maintenance on changes, and finally reviews and approves of the changes that have been executed.

<Table 24> Actor per Software Maintenance Activity

Stage	Activities	Actors
Request	<ul style="list-style-type: none">• Writing the Modification Request Form (MRF)• Writing the Change Request (CR)	Users, customers
Analysis	<ul style="list-style-type: none">• Classifying types of maintenance and determining the severity• Analyzing the maintenance requests and the impact• Deciding on the maintenance priority	Analysts
Approval	<ul style="list-style-type: none">• Approving of a need for maintenance based on the analysis outcome• Approving of the execution of maintenance	Maintenance Management Board
Execution	<ul style="list-style-type: none">• Executing maintenance on the targets• Writing the Software Change Report (SCR)• Modifying related reports	Maintenance operators

<Table 25> explains the types of software maintenance units.

<Table 25> Types of Software Maintenance Units

Type	Description
Unit based on the form of tasks	<ul style="list-style-type: none">• Analyzing user requirements, and designing, implementing and testing systems• Each team conducting separated roles and responsibilities• A need for coordination cost between analysts and programmers although they might be specialized in programming knowledge and techniques
Unit based on the application area	<ul style="list-style-type: none">• Classifying the unit depending on applications• Being equipped with expertise on the development of applicable knowledge

Type	Description
Unit based on the life cycle	<ul style="list-style-type: none">• Distinguishing the unit into development and maintenance parts• Specializing in development and maintenance techniques• A need for coordination cost between analysts and programmers

02 Reverse Engineering, Re-engineering, and Reuse

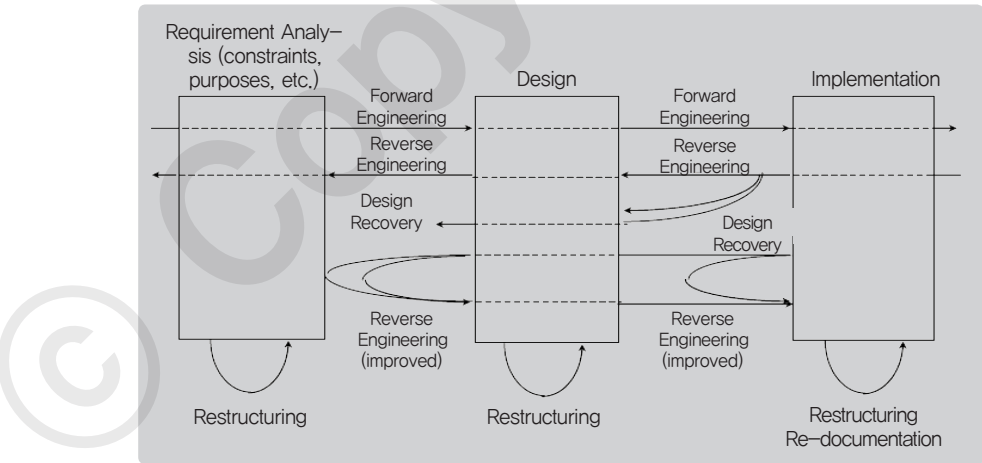
Software 3R

Software 3R refers to techniques to maximize software productivity based on the repository through reverse engineering, re-engineering and reuse.

- ① A need for 3R
 - Reducing cost resulting from maintenance errors and reuse
 - Enhancing productivity in software development
 - Understanding and modifying the system and enabling testability
 - Accelerating the Time to Market for requests for software modifications

- ② Concept Map for Software 3R

<Figure 17> shows how reverse engineering, re-engineering and reuse are used in the development stage, and <Table 26> explains about the 3R.



<Figure 17> 3R Concept Map

〈Table 26〉 Composition of 3R

Composition	Explanation
Reverse Engineering	<ul style="list-style-type: none">Analyzing the software that has been implemented → Designing → Analyzing requirements, being opposite from forward engineeringForward engineering: actualization of abstract concepts, requirement analysis → design → implementation
Re-engineering	<ul style="list-style-type: none">Actualizing into abstract concepts based on software restructured through reverse engineeringRestructuralization: modifying source codes without modifying features (modification of expressions)
Reuse	<ul style="list-style-type: none">Using the software being implemented and used through re-engineering

Outline of Reverse Engineering

① Definition of reverse engineering

Reverse engineering is a type of engineering to reversely trace the system already made, and obtain data such the original documents and designing techniques. This is a series of activities to understand and correct the system and move onto the software maintenance stage. In other words, it is a task of generating information or documents considered as the output of the initial stage of the lifecycle by using the document or documents acquired in the last stage of the software lifecycle. Reverse engineering is opposite from forward engineering of consecutive execution starting from designing.

② Cases requiring reverse engineering

- When it is difficult to do maintenance for the system already in operation
- When changes are frequent, which has lowered the system efficiency
- When re-establishing a task developed as a file system into a relational database
- When downsizing the basic mainframe

③ Advantages of reverse engineering

- Helping to analyze software that has been commercialized or already developed
- Enhancing maintainability by being able to analyze the data and information in the existing system in the designing level
- Storing the existing system information in a repository and facilitating the use of CASE

④ Input/Output of reverse engineering

〈Table 27〉 Input/Output of Reverse Engineering

Input	Output
Data and documents in forms of input/output including source codes, object codes, working process and libraries	Structure chart, data flow diagram, control flow graph, entity relationship diagram

⑤ Types of reverse engineering

〈Table 28〉 Types of Reverse Engineering

Type	Explanation
Logic Reverse Engineering	Eliciting information from a source code, and storing it in a repository on physical design information Obtaining physical design information
Data Reverse Engineering	Correcting the existing database, or porting it to a new database management system

Outline of Re-engineering

① Definition of Re-engineering

It is a representative software engineering technique to improve the maintainability and quality and reduce the maintenance cost by altering and improving the existing software data and features, given that maintenance cost takes up most of the software development cost. Re-engineering is a task of conforming the existing system to the widely used programming standard, reconfiguring it into a high language, or transforming it to be used in other hardware.

Re-engineering can be defined as software examination and alteration to reconfigure software in a new form, even including the implementation of new forms that follow. Although re-engineering is the most radical and expensive alteration, it is sometimes executed for small-scale alternations. It is often used to replace the existing obsolete software instead of improving the maintainability. Normally, after reverse engineering is conducted, forward engineering is conducted again in consideration of alterations.

② Purpose of applying re-engineering

- Enhancing the maintenance in the current system
- Facilitating the understanding and modifiability of the system
- Reducing the maintenance cost and time
- Facilitating the conformance to standards and use of CASE

③ Software re-engineering stage

- Stage of eliciting information from a source code
- Stage of re-engineering, system configuration and verification: recognizing repository information
- Stage of forward engineering and stage of design optimization
- Stage of source code generation

④ Advantages of re-engineering

- A method of handling complicated systems: CASE
- Generation of other views: graphing using re-engineering tools
- Detecting side-effects
- Synthesizing high-level abstraction
- Facilitating reusability

⑥ Methods of applying re-engineering

〈Table 29〉 Methods of Applying Re-engineering

Type	Description
Re-structuralization	A method of gathering software components in libraries, and finding and combining necessary components for new software development
Re-modularization	A method of altering the module structure of systems which is associated with cluster analysis and coupling of system components
Semantic information elicitation	A method of recovering in the document level instead of the code level

Outline of Software Reuse

① Definition of software reuse

Software reuse is a method of configuration to be suited to repeated use to raise the development productivity by standardizing software development-related knowledge (features, modules, composition, etc.).

② Advantages of software reuse

- Reducing TCO¹⁾ in software production
- Effects of sharing and utilization for the production of high-quality software
- Sharing information on system development, and output of other projects

③ Purpose of software reuse

〈Table 30〉 Purpose of Software Reuse

Goal	Description
Reliability	Verified performance in features, stability, speed, etc.
Scalability	Easy to upgrade based on a verified function
Productivity	Overall improved development process e.g. cost and time risks
Usability	Providing the assembly as independent components
Maintainability	Ease of quality enhancement, error correction, operation and upgrading
Adaptability	Ease of applying a new process as independent components

1 TCO (Total Cost of Ownership): This concept considers the investment effects in IT cost used in a company released by Gartner Group, a prestigious consultancy in the U.S. in 1997. In other words, it considers not only initial investment cost but also operating and maintenance cost when companies adopt a computer system.

Example Question

Question type

Descriptive question

Question

Read the following scenario and describe what kind of software engineering techniques will be useful for Mr. K to execute tasks continuously, and state why.

Mr. K was assigned a new task who joined a software package development company as an experienced staff. It was a development project exclusively developed and handed over by a previous software engineer who suddenly quit the company.
The company could afford Mr. K adequate training and time to get to know the work, so Mr. K had to seamlessly carry on his predecessor's task no matter what. He needed design documents such as a structure map to understand the source code as soon as possible, but having searched for documents his predecessor has left, he could only find a simple requirement specification written in the planning department, and some source codes written by his predecessors. Therefore, the only data for reference available were the simple requirement specification and source codes.

Answer and explanation

The concept of reverse engineering is used:

- ① The system source code is analyzed in reverse engineering to generate components(ingredients) of software and their relationship.
- ② It is a process of re-writing the design and requirement analysis information.
- ③ If Mr. K uses this technique, he can reconfigure the data dictionary, data flow, control flow, data structure, entity-relationship diagram, process specification and structure chart based on the source code analysis.
- ④ Information on the written codes is obtained by escalating from the lowest abstraction level to the highest abstraction level, validating software that is currently developed and carrying on the development task.
- ⑤ The information reconfigured by reverse engineering is lower than the ideal level, but it is the only viable option.

Related E-learning Contents

- Lecture 11 Management of Software Maintenance

Management of Software Requirements

▶▶▶ Latest Trends and Key Issues

It is the most basic and important process in the entire lifecycle of applications, and various techniques and tools are developed for efficient management of requirements.

▶▶▶ Study Objectives

Able to understand the concept and the process of managing software requirements, and utilize the management of tracing and modifications

▶▶▶ Practical Importance High

▶▶▶ Keywords

- Process of managing requirements
- Requirement specification technique
- Management of requirement changes

01 Requirement Management

Definition of Requirement Management

Requirement engineering can be broadly divided into requirement development and management. It is to develop requirements defining what to do, identifies if requirements defined to be fulfilled have been reflected properly and continues to manage the initial requirement changes.

Importance of Requirement Management

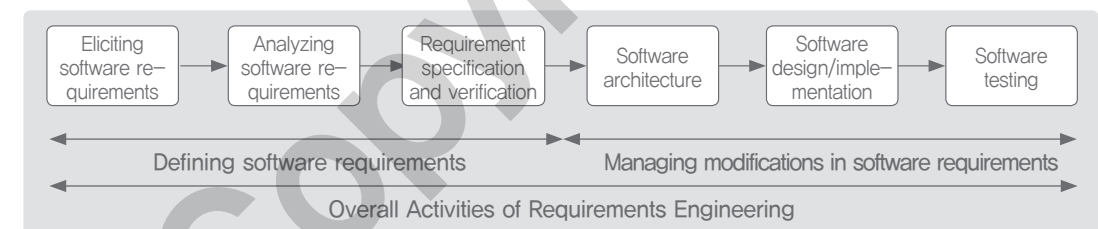
Appropriate management of requirements provides effective means of communication among various stakeholders, prevents delivery delay and budget excess, and enables the execution and management of user requirement specification.

Purpose of Requirement Management

Requirement management aims to accurately validate and satisfy customer needs in the customer perspective and produce high-quality software within limited timeline and duration.

Process of Requirement Management

⟨Figure 18⟩ explains requirement management activities in the entire software development lifecycle.



⟨Figure 18⟩ Requirement Management Activities

⟨Table 31⟩ explains the specific activities of the requirement management process and the output.

⟨Table 31⟩ Specification of Requirement Management Activities

Process	Explanation	Output
Eliciting Requirements	<ul style="list-style-type: none"> • Defining business requirements • Distinguishing participants • Eliciting initial requirements 	Candidate Requirement
Analyzing Requirements	<ul style="list-style-type: none"> • Modelling candidate requirements • Prioritizing requirements • Negotiating on requirements 	Agreed Requirement

Specifying Requirements	<ul style="list-style-type: none">Defining requirement specification criteria / specification statementStoring information on requirement traceability	Formal Requirement
Verifying Requirements	<ul style="list-style-type: none">Reviewing the requirement specification / terminology verification / baseline setting	Baselined Requirement
Managing Requirement Changes	<ul style="list-style-type: none">Controlling requirement changes, traceability and version (track record management)	Consistent Requirement

Principles in Requirement Management

- Principles in managing requirements are as follows:
- Prioritizing the customer value-oriented requirements / Acquiring consent on stakeholder requirements
 - Accurately distinguishing goals of systems required (expectation management / scope management)
 - Analyzing the impact of requirement changes by operating the Change Control Board (CCB) and setting the baseline for each level of change

02 Requirement Specification

Requirements must be defined by specification techniques, and in specifying requirements, correct specifications can be defined in complying with principles.

Requirement Specification Techniques

〈Table 32〉 Requirement Specification Techniques

Type	Specification Techniques	Explanation
Formal Specification	VDM	<ul style="list-style-type: none">Vienna Development MethodState-based graphic specification method
	Mathematical technique	<ul style="list-style-type: none">Providing a verification framework for specification development and systematic systems
Informal Specification	FSM	<ul style="list-style-type: none">Finite State MachineExpressing the state transition with input signals
	SADT	<ul style="list-style-type: none">Structured Analysis and Design TechniqueGraphic-based structural analysis model
	Use case	<ul style="list-style-type: none">User-based modeling
	Decision Table	<ul style="list-style-type: none">Indicating probability and cases for decision-making
	ER modeling	<ul style="list-style-type: none">Expressing the entity relationship

① Principles in Requirement Specification (IEEE 830)

〈Table 33〉 Principles in Requirement Specification

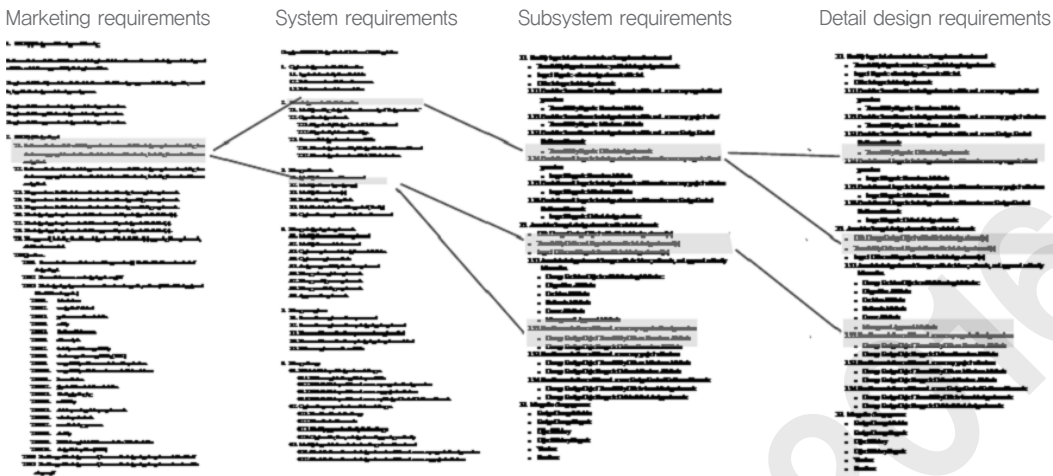
Principle	Explanation
Verifiability	<ul style="list-style-type: none">Requirement specification to be verifiable
Modifiability	<ul style="list-style-type: none">Requirement specification to be modifiable
Clarity	<ul style="list-style-type: none">Requirement specification to be clearly suggested per stakeholder
Correctness	<ul style="list-style-type: none">Requirement specification to be correctly described
Traceability	<ul style="list-style-type: none">Fundamentals and mechanisms of requirements to be traceable
Consistency	<ul style="list-style-type: none">Requirement specification to have no collision among requirements
Completeness	<ul style="list-style-type: none">All major descriptions on functionality, performance, constraints, etc. to be documented
Interpretability	<ul style="list-style-type: none">Requirements: providing consistency in interpretation
Understandability	<ul style="list-style-type: none">Ease of understanding among stakeholders

〈Table 34〉 explains the form of requirement specifications defined in the IEEE.

〈Table 34〉 Requirement Specification (complying with the IEEE 830 Guideline)

Content	Explanation
1. Outline 1.1 Purpose 1.2 Related Departments/Parties 1.3 Scope of Development	<ul style="list-style-type: none">General introduction of softwareDescription of the purpose of software requirement specificationStakeholders reading the software requirement specificationPurpose of the software, restrictions, characteristics, benefits, business strategies, relationship with common goals(Overall explanation about the software)
2. Outline	
2.1 Project Outline	Description on the origin of the software, replacement of the existing system and whether or not the software is an independent product, etc.
2.2 Product Function	Description of the function list only, and specific description in Chapter 4
2.3 User Characteristics	<ul style="list-style-type: none">Description of user characteristics (experience, academic background, tendency, etc.)Description on the frequency of usage per user
2.4 Software Operating Environment	Description on the hardware environment for software to operate in, OS and the software environment, etc.
2.5 Decisions and Distribution	<ul style="list-style-type: none">Constraints with the hardware, memory constraints, usage of specific technologies, multi-language support, securityDesign conventions, programming conventions
2.6 Assumptions and Dependence	Assumptions impacting the software (dependence on outsourced components, dependence technique of other project components)
2.7 Data Requirements	Input/Output data per function in the user perspective
2.8 Use Scenarios	Scenarios in the user perspective to explain the system
3. Interface	
3.1 User Interface	Sample screen image, standard button, standard error message, keyboard short-cut, etc.
3.2 Hardware Interface	Protocol exchange, interactions between the hardware components of the system and software products
3.3 Software I/F	Defining the data or messages entering the software from other software components

4. Functional Requirements 4.1 (Function Name)	<ul style="list-style-type: none">• Description of specific requirements on the features listed in Chapter 2 in the Content• Purpose: purpose of/reasons for the features• Inputs: data entering the features• (Validation check, error corrections)• Outputs: data derived from the features• (process error message, parameter scope, form, shape, distance, volume output)
5. Non-functional Requirements	
5.1 Performance Requirements	Number of users logging in, response time, file/table size, number of transactions per unit time
5.2 Security Requirements	Encryption of data and telecommunication, access control and authority management, etc.
5.3 Software Quality Requirements	Availability, flexibility, interoperability, maintainability, portability, reliability, usability, etc. Description on software quality requirements
5.4 Business Rule	Operating rules, rules complied with when individuals/organizations execute tasks using the software CBP (Current Biz Process) and FPM (Future Process Model)



〈Figure 19〉 Sentence-Specific Traceability

03 Management of Requirement Changes and Traceability

Outline of Requirement Traceability

Once the requirement baseline is set and a project begins, requirements are inevitably subject to changes. Management of requirement changes is a process to officially control all changes that occur based on the baseline of requirements. It provides official control of changes on requirement changes that occur in the project execution process, and provides consistency and integrity on changes resulting from such activities and tracing. Changes in requirements are subject to errors due to the following reasons:

- Errors, collision and inconsistency in requirements
- New understanding and knowledge acquisition on the system of participants
- Changes in the system environment and units
- Occurrence of technical, time and cost issues

Ways to reduce the above errors are to define and configure traceability of requirements. Tracing requirements does not mean the tracing of document-specific requirements but the tracing of sentence-specific requirements within requirement documents. The concept of traceability in the unit of individual sentences is easily understandable if the following figure is referred to:

If there is traceability attached to individual requirements within the development output as above, it is clear how marketing requirements are reflected in the system requirements, and to what subsystems they are allocated and reflected in design. It is possible to validate some points. What would be the estimated additional development efforts due to changes in marketing requirements resulting from users' request for changes? When developers want to arbitrarily change a part of design requirements, what would be the impact on high-level requirements? What are the requirements that are omitted in the intermediate stage? Management of requirement traceability can be effective when requirements are managed by unit. The benefits from traceability management for each requirement unit within all document output requiring traceability management can be summarized as follows:

- Reducing rework by preventing omissions of high-level requirements
- Raising individual requirement quality by securing traceability (clarity, traceability, testability, etc.)
- Analyzing the impact of change
- Effectively collaborating among various departments/companies
- Securing test coverage by acquiring traceability with test cases and raising the product quality
- Securing cross-output consistency through the management of various changes
- Raising productivity with the reduction in miscommunication due to changes

Example Question

Question type

Descriptive question

Question

It is guided that in defining requirements, domains are divided into problem domains and solution domains. Explain the differences between the two domains and explain what problems can be addressed.

Intent of the question

To validate if the procedures defining requirements and their differences are understood

Answer and explanation

In the stage of defining customer requirements, what customers want for software services/products in the perspective of problem domain customers can be accurately described, and in the stage of defining system requirements, how customer requirements can be contained in the system can be described in order to complement the following problems:

1. Accurate understanding of actual problems
2. Scope setting for the system, and accurate understanding of each function consisting of the system
3. Search for optimal solutions due to a lack of design freedom
4. Avoidance of development oriented toward developers and solution providers in explaining the solution –oriented system

Related E-learning Contents

- Lecture 12 Management of Software Requirements

XII

Software Configuration Management

▶▶▶ Latest Trends and Key Issues

As software configuration management is perceived as a critical element impacting the quality, its importance has gradually been highlighted, and the use of CASE tools for configuration management has been emphasized. Configuration management in the distributed environment has become a spotlighted issue too.

▶▶▶ Study Objectives

Able to explain the concept and activities of software configuration management and to utilize its tools

▶▶▶ Practical Importance High

▶▶▶ Keywords

- Software configuration, Software configuration management, configuration management element, procedures to control software change
- Control of software configuration access and synchronization
- Version management tool, configuration management tool

01 Outline of Software Configuration Management

Definition of Software Configuration Management

Software Configuration Management(SCM) refers to a series of activities developed to manage changes in software. Software configuration management is to control by finding the causes for software changes, validate if changes are made appropriately and notify them to related operators. As the stage is applicable for the entire software development process, it can be conducted not only in development and maintenance in order to reduce the entire cost for software development and guarantee the minimization of many risks rampant in the development process. Here, the word ‘configuration’ encompasses programs made in the software development stage, and documents and data explaining the programs. Software configuration management is a management technique to systematically manage various outputs generated in the software development process. In other words, it is to manage all behaviors related to records of change for all components generated from the software development stage up to maintenance.

〈Table 35〉 explains general problems when configuration management is poorly done

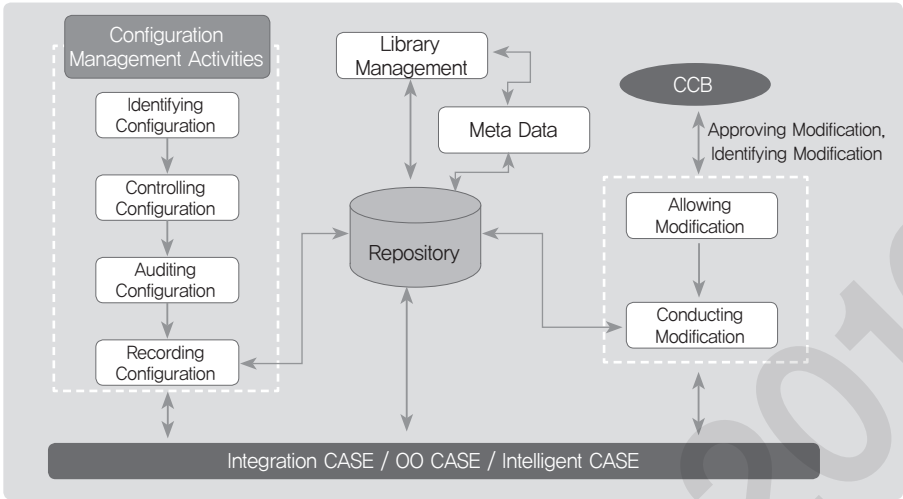
〈Table 35〉 Background for a Need for Configuration Management

Causes of the Problems	Description
Poor Visibility	There is no visibility for software as an intangible output.
Difficulties in Control	It is realistically difficult to control invisible software development.
Poor Traceability	It is difficult to trace the entire process of software development.
A Lack of Monitoring	It is difficult to continue on with project management due to poor visibility and traceability.
Endless Changes	Endless user requirements

02 Concept Map of Configuration Management and its Components

Concept Map of Configuration Management

〈Figure 20〉 conceptually explains the entire activities and mutual relationships of configuration management.



〈Figure 20〉 Concept Map of Configuration Management

② Components of configuration management

The below table explains major items as components of configuration management.

〈Table 36〉

Components	Description
Baseline	Timing of technical control of each configuration item, and timing to control all changes
Configuration Items	Basic targets that are officially defined and described in the software lifecycle
Configuration Products	Configuration management targets where officially implemented entities are realized in the software development lifecycle Technical documents, hardware products, software products
Configuration Information	Configuration Information = Configuration Items + Configuration Products

03 Activities of Configuration Management

Activities of Configuration Management

Configuration management activities consist of validating, controlling, auditing and recording configurations, and are explained specifically as follows.

〈Table 37〉

Activities	Description
Validating Configurations	<ul style="list-style-type: none">• Distinguishing targets for configuration management and numbering items to be managed• Objectives to validate configuration items: facilitating the process of defining document structures to be clear and predictable, and also traceability and management based on information records• Description to validate configuration items: products/various documents/configuration item number
Controlling Configurations	<ul style="list-style-type: none">• Reviewing and approving requests for changing software configuration and controlling them so that they could be reflected in the defined baseline• Managing requirements for change/controlling changes/operating configuration management units and supporting to control configuration for developers and outsourcing companies
Auditing Configurations	<ul style="list-style-type: none">• Serving as a means to determine integrity of software baseline• Successfully setting the software baseline by successful configuration auditing• Reviewing to see if changes in the baseline are in line with requirements• Verification, validation
Recording Configurations	<ul style="list-style-type: none">• Recording various statuses and execution results on software configuration and change management, and writing a report by managing the database

Effects of Configuration Management

Once activities of configuration management are applied to software development and management, the following effects can be obtained:

〈Table 38〉

Type	Description
Manager Side	<ul style="list-style-type: none">• Providing the criteria for systematic and effective management of a project• Facilitating the project control• Securing project visibility and guaranteeing project traceability• Suggesting the baseline for quality assurance
Developer Side	<ul style="list-style-type: none">• Minimizing the impact resulting from software change, and facilitating management• Techniques for software quality assurance• Enabling appropriate change management for software• Improving maintainability

Considerations for Configuration Management

The following points must be considered for efficient configuration management:

- Forming an appropriate operating unit and utilizing tools specialized in management
- A need for continued management and management criteria and for problem-solving measures
- Appropriately tailoring the extent of configuration management for a smaller project
- Setting items for configuration management and implementing all changes based on official negotiations
- Cautiously implementing changes for the software in operation

04 Configuration Management Tools

Management of source codes is one of the critical points in developing software. The Version Control System (VCS) or the Source Code Management (SCM) System is required which is a system to store and manage source codes for managing various versions of records and changes and for collaboration among team members.

Let's have a look at the flow of the system for software configuration management. First, there is configuration management that executes version updating using shared folders, etc. based on file systems in the local part. In the next phase, network-based client/server type management systems came to be used, making distributed management highly popular. Tools for configuration management for each type are as follows.

Configuration Management Tools

The following table explains the types of representative configuration management tools, and their application techniques and characteristics. SVN and Git as key open source configuration management tools, and TFS, a commercialized tool from Microsoft will be explained from the next chapter.

〈Table 39〉

Type	Tools	Note
Shared Folders	RCS, SCCS	
Client/Server	Subversion(SVN), CVS, Perforce, ClearCase, TFS	Centralized Version Control System
Distributed Repository	Git, Mercurial, Bitkeeper, SVK, Darcs	Providing distributed repositories Offline development work Advantageous for a multiple number

Subversion (SVN)

The Subversion is a management system of an open software version. It is called the 'SVN' named after a command used in a command-type interface. It has been developed by CollabNet since 2000 to replace the CVS that had restrictions.

Currently, it is being co-developed with the global developer community as the highest-level project of Apache. The Subversion follows the server-client model. The server can be included within a computer being used, or be placed in a separate computer connected to the network. The Centralized Version Control System like the Subversion is used by multiple clients as they check out files from the central server.

As for the Centralized Version Control System, codes are stored in a store server only. When developers download a source code, and edit/store it, the descriptions are reflected in the Centralized Version Control System. In other words, there is a master version (latest version) source code stored always in a server, and when a server is disconnected or network access is impossible, the code cannot be committed or the latest code cannot be downloaded. As a result, when all the data is lost due to a problem in the hard disk of the central server, there is no way to recover

it. The biggest pitfall is that the version management server must be concentrated in a single spot, and is entirely dependent on the server account.

Distributed Version Control System (Git)

With the Distributed Version Control System like Git, the client does not have to download the last snapshot directly in copying the entire version control system. Even for a problem with a server, the server can be restored with a copy of the client. With the Distributed Version Control System, a source code is not only stored in a central server, but in local PCs of multiple servers or developers, and each becomes a source repository. The source code stored in each source repository is not of the same version and the code is stored in different branches. In other words, Server A can store Versions A, B, and C and Server B can store different branches such as Branch A, C, and D. In short, the branch version of each repository is all different, and the place that is brought in by accessing a source code is all different, so there is no such concept as 'master version' in the system.

The source code version management can be highly flexible because repositories can be separately developed in team or function units, or repositories can be separately developed in release version units. There is no such concept as a central repository, so despite a failure in a particular VCS system, as long as the VCS one uses is not problematic, the development can be continued on. As mentioned earlier, the VCS can be installed in local PCs of developers, so with no network connectivity, development can be carried on. Because the source code is stored in not only the central server but also multiple servers and PCs in a distributed manner, despite any damage done to a repository due to a server failure, other servers or PCs would store all the source codes and histories, so the Distributed Version Control System is easy to recover compared with the central server system.

TFS (Team Foundation Server)

The Team Foundation Server (TFS) is a product of Microsoft that provides the following features: management (team foundation version control), reporting, requirement management, project management (agile software development, Waterfall model), automated build, lab management, and test and launch management. TFS can be used as a back-end in numerous integrated development environments, but is designed to provide optimized advantages when used as a back-end for Microsoft Visual Studio or Eclipse (Window and non-Windows platforms). TFS is an integrated repository based on the SQL server. It can store all types of information related to software development activities of development teams including source codes, output and development activities for them to achieve effective collaboration. Source control in TFS is a version management tool that has nothing to do with the existing Visual Source Safe. The main difference is that security configuration and authority configuration have been strengthened in TFS as SQL-based storage types. TFS also has improved stability due to the execution of database transactions, and its scalability has become stronger so that it can be configured to a large team of thousands. TFS includes a strong version control system to cater to requirements for source control for a company as a whole, so it has been developed to provide high-speed security access to guarantee reliability of the data available in this version. TFS has not only check-in, check-out, and a standard version control mechanism for version control and branch/merge but also features to solve specific problems of large-scale distribution such as shelving (a function to store changes without executing the entire validity check-in) and dynamic check-in policy.

The following explanations are on key features provided by TFS:

- Version Control – managing source codes requiring version management and other execution files
- Work Item Planning and Management – continued tracing of defects, issues, requirements, tasks and scenarios
- Project Management Function – supporting for planning and tracing by forming a team project according to the software process that users can tailor and aligning with Microsoft Excel and Microsoft Project
- Team Build – providing a process and a workflow to build into executable products
- Data Collection and Reporting – supporting to evaluate and make decisions on a team project based on various information acquired from TFS
- Team Project Portal – providing a collaborative workplace to enable seamless communication for team projects
- Team Foundation Sharing Service – providing shared infrastructure services that are critical to tool developers and scale-up providers although end users might not be able to see them

Example Question

Question type

Short-answer question

Question

The example shows a circumstance that occurred as a result of an activity omitted in the process of configuration management.

There was no new defect even after source codes were corrected and tested to resolve two cases of defects detected while doing the system testing. Correction was made again along with testing, but new defects were additionally detected, and this phenomenon repeatedly occurred. Thus, to do it again from the beginning, one wanted to go back to the state prior to the defect correction, but the source code was repeatedly corrected, and due to no records of change, it was impossible to return to the original state.

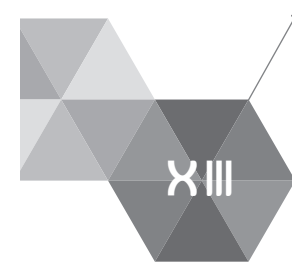
Answer and explanation

Change Control or Change Management

The above description is slightly detached from the definition and scope of change control. However, given that the concept is not to correct the original file and maintain it before completing the change, that is, completing the test for changes and quality assurance, it is not a problem even if change control is written in the answer. Similar with versioning or version control, but it cannot be the right answer because it is an act conducted to manage each version after completing a version.

Related E-learning Contents

- Lecture 13 Software Configuration Management



Software Quality Management

▶▶▶ Latest Trends and Key Issues

Recently, many companies have stayed away from the rule of thumb-based management to develop and manage high-quality software, establishing a quality management system in a systematic and quantitative manner and making consistent investment.

▶▶▶ Study Objectives

Able to explain the concept and activities of software quality management

▶▶▶ Practical Importance Medium

▶▶▶ Keywords

- Software quality
- Software Quality Model

01 Software Quality Management

Definition of Software Quality Management

Software quality management refers to activities to control and guarantee the quality of all activities taking place throughout the application lifecycle the output as a result of these activities, consisting of software quality guarantee and software quality control activities. Software quality management is a comprehensive quality activity by deciding on the quality goals, policies, responsibilities and roles of software, and conducting quality planning, guarantee, control and improvement within the quality system. It means an operating scheme to control management driven by statistical numeric through development activities so that software product quality could be maximized with the given amount of cost input under the same conditions.

Purposes of Quality Management

Purposes of software quality management can be summarized as follows:

〈Table 40〉 Purposes of Quality Management

Purpose	Description
Technical Evaluation	Predicting appropriate estimation standards and software quality
Resources Evaluation	Estimating appropriate resources and cost
Process Evaluation	Controlling the application lifecycle process
Product Check	Conducting routine test, checking out the output, comparing with other products (adopting packages)

Elements of Software Quality

Key elements of software quality can be broken down as follows.

〈Table 41〉 Elements of Software Quality

Purpose	Description
Operation	Correctness, reliability, efficiency, scalability, ease-of-use, integrity, maintainability
Correction	Maintainability, portability
Adaptation	Testability, reusability, interoperability, maintainability

02 Perspectives of Software Quality

Software quality might vary by stakeholder, so it is essential to fully check out quality expectations and requirements from all stakeholders before executing projects in evaluating the quality.

Users' Perspective

- Mostly taking an interest in the software usage, performance, and usage effects
- Evaluating software not knowing the internal aspect of software and how it is developed

Developers' Perspective

- Taking an interest in not only the quality of final products but also that of intermediate ones
- Using different scales for different stages for quality evaluation in each stage in the development stage
- Including the quality feature perspective needed by software maintenance personnel

Managers' Perspective

- Taking an interest in the overall quality instead of particular characteristics of quality
- Requiring to impose a weight to reflect business requirements in each feature
- Requiring to harmonize management criteria such as deadline delay and cost addition with quality enhancement

03 Characteristics of Software Quality and Major Software Quality Models

Characteristics of Software Quality

Characteristics of software quality refer to those of software, and are used as quality evaluation items for software. ISO 9126 is a representative standard for quality characteristics, and can be classified as follows in 〈Table 42〉:

〈Table 42〉 ISO 9126 Quality Characteristics

Quality Characteristics	Description	Quality Sub-characteristics
Functionality	• A software product capability providing features that meet explicit and implicit requirements	Suitability, correctness, interoperability, flexibility, security
Reliability	• A quality to maintain the prescribed performance level in using prescribed conditions • A software product capability to enable users to prevent errors	Maturity, fault tolerance, recoverability
Usability	• A software capability to enable users to easily understand, learn and prefer	Understandability, operability, learnability
Efficiency	• The extent of performance provided to input resources • The extent of the consumption of resources required to execute required features	Execution efficiency, resource efficiency
Maintainability	• A capability that can be changed including software correction or improvement according to the operating environment, requirements and functional specifications	Interpretability, stability, ease of change, testability
Portability	• A capability for software to be ported to other hardware or software • The extent of software's capability to be ported to another environment	Adaptability, conformity, workability in porting, substitutability

Key Software Quality Models

Software quality models can be referred to in evaluating a product or a process, and considering it as a model case to be executed in a company, thus improving the quality. 〈Table 43〉 as follows classifies the quality standard and models in the product and process perspectives

〈Table 43〉 Key Software Quality Models

Type	Product Perspective	Process Perspective
Characteristics	Product measurement, product verification, product identification	Improving and inspecting software
Methods	Evaluating functionality, reliability, usability, efficiency, maintainability and portability	Evaluating process (procedure) conformity
Standards and Models	ISO/IEC9126,14598,12119, 25000SQuaRE	ISO 9000, ISO/IEC 12207, SPICE, CMM, CMMI
Advantages	• Applicable to all types of software • Objectification of specialized judgment	• Applicable to many types of products • Shortening the test time and reducing certification cost
Disadvantages	• Consuming cost and time of routine test • Difficult to evaluate latest software	• Impossible to guarantee the quality comparatively • Difficult to apply to innovative software

04 Methods of Software Quality Measurement

Software measurement is a method of quantifying software codes or a development process. Software measurement(metrics) can be done through the following methods.

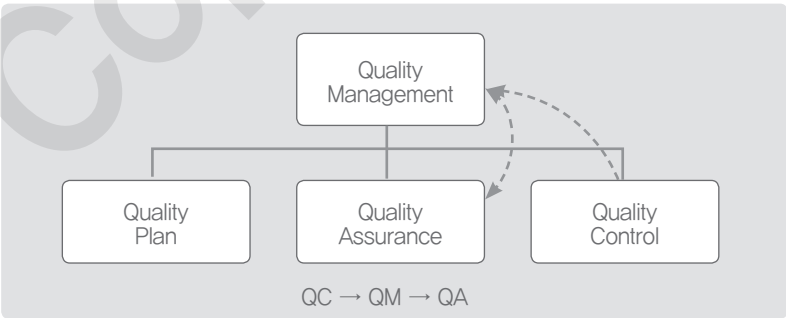
- Direct Measure: a method of measuring the size of software as a bundle, measuring cost, timeline, effort, Lines of Code (LOC), memory size and error, etc.
- Indirect Measure: a method of measuring the quality of software based on the directly estimated value, measuring features, quality, complexity, reliability, maintainability, efficiency, etc.

Types of software metrics are as follows:

- Quality Metric: explicitly and implicitly indicating the software quality metric on user requirements
- Productivity Metric: focusing on the output of the engineering procedure
- Technical Metric: focusing on software characteristics including a logical complexity or the extent of modularization
- Size-oriented Metric: directly measuring and collecting the size of output based on the software engineering procedure
- Human-oriented Metric: collecting information on efficiency including developers' attitude, tools or methods
- Feature-driven Metric: information based on the Indirect Measure, including quality, reliability, complexity and maintainability

05 Software Quality Management

The software quality management system can be broken down into planning, execution and control stages, and each stage determines necessary activities and subjects, and enables quality activities to be continuously executed.



〈Figure 21〉 Software Quality Management

The following 〈Table 44〉 explains components and major activities of software quality management.

〈Table 44〉 Breakdown of Software Quality Management and the Major Activities

Type	Concept	Activities
Quality Plan	Validating the standard of quality to be executed, and determining execution methods	<ul style="list-style-type: none">Validating requirements in the quality systemValidating the quality assurance proceduresIdentifying the quality control proceduresDefining operating proceduresMaking the quality management checklistWriting quality management plans
Quality Assurance	Executing quality assurance to see if software products conform to requirements in a third-party perspective	<ul style="list-style-type: none">Reviewing project outputReinforcing preventive measures for quality accidentsReviewing project procedures
Quality Control	Autonomously executing quality activities in the software lifecycle	<ul style="list-style-type: none">Monitoring quality outcomeAnalyzing differences with planned quality levelsEstablishing correction plans if necessaryDocumenting correction activities and updating the plans

- Clearly define problem domains.
- Avoid discussions on solutions or improvement measures
- Limit the number of participants and make sure to prepare for a review.
- Develop a checklist on the output to be reviewed.
- Allocate resources, time and timeline in advance.
- Make sure that all reviewers take training in advance.
- Share reviewed outcome among reviewers.
- Check out the review process and outcome.

Types of official technical review include the Walkthrough and Inspections, which are all executed in forms of meetings.

The Walkthrough is to do a small-scale review on software modules or source codes as a technical evaluation meeting that can be executed in each software development stage. The purpose is early detection of defects, and the detected are documented. Defects that are detected are not addressed in a review meeting, but are compiled for post-meetings. Meeting participants must be 3~5, and the review time can be made effective if it is less than two hours. Review data are distributed in advance for people to review in advance which is supposed to take less than 2 hours.

The inspections are more standardized and formalized forms of the Walkthrough, reviewing and improving the output quality in the software development stage. The Inspection Team consists of 1~4 members that are trained on the field. Inspectors execute the process by using the checklist on the inspection items.

Software Quality Assurance Procedure and Activities

For software quality assurance, the execution procedure must be defined and systematically executed. First of all, quality assurance plans are set, which are then subject to reviews among stakeholders. Mutual consensus is made on quality activities and metrics, and the execution outcome are documented, reported and notified so that the execution outcome of quality activities can continuously monitored and appropriate corrective measures can be taken.

Key software quality activities are as follows besides technical review activities:

- Verification: an activity to inspect if each stage of software analysis and design is correct, or if source codes are defective
- Validation: an activity to inspect if defining and analysis of requirements has been properly done so that proper software products can be developed in line with the operating environment of customers or users
- Certification: an activity where users or customers, or experts on behalf of users officially validate the software quality
- Software Test: an activity to execute a program to detect defects
- Debugging: an activity to accurately diagnose and correct the causes for defects that are detected
- Audit: an activity based on the project audit to evaluate the conformity with the project standard process, requirement documents and project plans. The audit is executed as defined in the standard process, enables observation on conformities with standards or plans, and requires corrective measures, if necessary.

06 Activities for Software Quality Assurance

Software quality assurance is a planned and systematic task executed in the entire development stage to validate if a particular software conforms to predefined requirements. Activities for software quality assurance set quality goals by thoroughly validating software characteristics and requirements from the starting point of software development. In the development stage, standardized review activities take place to check out if quality goals have been met. After development, debugging and testing occur.

There are the following necessities for software quality assurance:

- Enhancing productivity by satisfying user requirements as much as possible
- Detecting quality problems in the development stage and removing them
- Ensuring deadline conformity, and product robustness and scalability
- Saving cost, improving productivity and increasing reusability

Techniques for Software Quality Assurance

The key technique for software quality assurance is the Formal Technical Review (FTR), an activity to guarantee software quality executed by developers. In order to assure the software quality, developers must conduct technical review for software. Technical review for software detects problems or defects that might occur in the development stage including planning, analysis, design and implementation, thus improving the software quality.

The following is a guideline on official technical review:

- Only focus on the review of source codes and output
- Define the agenda and execute it.
- Minimize debates and counter arguments

Software quality activities can enable the acquisition of the quality levels that need to be systematically executed throughout the software lifecycle. Since software quality activities are not one-time ostentatious ones, it must be aligned with the software development process to establish systematic quality planning and executing it continuously.

Activities for Software Quality Control and Evaluation

Software quality management consists of software quality control and quality assurance. Software quality control is an essential process in order to maintain the quality levels expected in software in the process of software development, operation and maintenance. By contrast, software quality assurance is a process to support the quality control process to guarantee the reliability of a particular software to related stakeholders. While software quality control is executed within software development units, operation units and maintenance units, software quality assurance is executed by a third party that mostly has an objective stance. Thus, for software quality management, only when quality control is organically aligned with quality assurance and executed accordingly, can the quality levels in demand be effectively achieved.

Example Question

Question type
Multiple choice question

Question
The quality of good software can be distinguished in the perspectives of clients, users and maintenance operators. Choose the most appropriate software quality standard in the perspective of maintenance operators.

- ① Functional accuracy, portability, ease of use and productivity
- ② Efficiency, productivity, re-usability and ease of comprehension
- ③ Portability, re-usability, interoperability and reliability
- ④ Flexibility, portability, reliability and ease-of-use

Answer and explanation
Answer : ③
Software quality standards can be distinguished in the perspectives of clients, users and maintenance operators.
Clients': minimum cost, productivity, flexibility, efficiency, reliability
Users': functional correctness, understandability, ease-of-use, consistent integration, efficiency, reliability
Maintenance Operators': portability, reusability, maintainability, interoperability, reliability

Related E-learning Contents

- **Lecture 14** Software Quality Management

Company A – Taking a glimpse at a business report in setting up a next-generation system

01 Project Outline

Business Outline

- ① Project Title: Next-Generation Comprehensive Information System Set-up
- ② Project Period: from the contract signing date to XX (date), 201X
- ③ Project Cost: approximately KRW XX million (including VAT)

Project Background

- ① Requirements to reshuffle the information system
 - The current system cannot make prompt and flexible responses to changes in the business process.
 - There is a lack of information linkage among internal systems.
 - Efficiency in the information system has dropped due to the absence of a standardized business process.
- ② A lack of enterprise-wide business process system
 - There is a need to establish an information system based on an efficient and standardized business process.
- ③ Absence of strategic corporate management
 - There is a lack of mutual conformity of IT with the vision, strategic goals, execution strategies, business process and personnel.
- ④ A lack of linkage among unit systems
 - Since the current information system was individually developed due to the demand, there is a lack of linkage among unit systems.

Business Scope

- ① Establishing the next-generation corporate management system
 - Personnel, wage, budget, accounting, purchase, contracting, asset, lease, training, EIS, EDMS, e-payment
- ② Establishing the UC-based EKP
 - The EKP is to be re-established to be user-oriented by strengthening knowledge management features, personalization services and integrated search features. It therefore enhances convenience in usage and supporting prompt business processing.

Expected Results

- ① Making business more efficient and securing information reliability
 - Enhancing cross-system information conformity based on development in consideration of business linkage among individual systems
 - Raising business efficiency by complementing business features and reinforcing business support by expanding the domains of e-payment
 - Raising efficiency in business processing by improving the process and simplifying unnecessary administrative process
- ② Raising user convenience and shared usage of knowledge
 - Raising convenience in usage by directly accessing the services in demand through individualized menus
 - Raising the user satisfaction levels by providing differentiated information and services depending on individual tasks
 - Enhancing task applicability by being able to concentrate on high value-added tasks with shortened time to acquire information necessary for tasks and to easily acquire and process the knowledge that has been built up
 - Ensuring the acquisition of knowledge by applying the process of managing the knowledge that can be applied to actual tasks, instead of merely registering or inquiring knowledge

Execution Strategies

- ① Establishing a field-oriented system
 - Facilitating a field-oriented system development by expanding staff communication channels including forming and operating a field-oriented working-level unit to set up a user-oriented system and conducting a survey
 - Forming a field-oriented working-level unit for reinforcing quality assurance by constantly complementing problems in the process of work analysis, design and development by applying latest analysis design techniques based on architecture development strategies, and also for minimizing on-site trials and errors in opening the system
 - Establishing a user-oriented system by expanding the scope so that staff can be directly engaged in development through various surveys and their participation in tests on major features
- ② Prioritizing in-house development
 - Initiating the development of an optimized in-house information system to elastically respond to internal and external changes in the environment, given that there are frequent modifications in units and tasks

Architecture Requirements

〈Table 45〉 Architecture Directions

Architecture Components	Current State	Directions
Programming Languages	COBOL + JAVA	JAVA
Application Servers	IBM Mainframe-based	JAVA operating environment
Tier Architecture	<ul style="list-style-type: none">• User screen: Java-based web• Processing logic: COBOL-based• Database: relational DBMS	<ul style="list-style-type: none">• User screen: Java-based web• Processing logic: Java-based• Database: relational DBMS
Operating Platform	Mainframe + Unix server	Mainframe
Security	Authentication management through Single Sign On	Introducing the EAM (including authentication management and screen authority management through the Enterprise Access Management-Single Sign on)

Requirements of Quality Attributes

Non-functional conditions of the next-generation system describe applications excluding infrastructure and satisfy quality attributes suggested in ISO 9126.

〈Table 46〉 ISO 9126 Quality Attributes

Quality Attributes	Description	Quality Sub-characteristics
Functionality	<ul style="list-style-type: none">• A software product capability providing features that satisfy explicit and implicit requirements	Conformity, correctness, interoperability, flexibility, security
Reliability	<ul style="list-style-type: none">• A capability to maintain the prescribed performance level in being used for prescribed conditions• A software product capability for users to prevent errors	Maturity, fault tolerance, recoverability
Usability	<ul style="list-style-type: none">• A software product capability to enable users to easily understand and learn, and prefer	Understandability, applicability, learnability
Efficiency	<ul style="list-style-type: none">• The extent of performance provided to resource input• The extent of requiring necessary resources to execute required features	Efficiency in execution, resource efficiency
Maintainability	<ul style="list-style-type: none">• A capability to be modified including correcting and modifying software in line with the operating environment, requirements and functional specifications	Interpretability, stability, ease-of-change, testability
Portability	<ul style="list-style-type: none">• A capability for software to be ported to other hardware or software• The extent of software's capability to be ported to another environment	Adaptability, consistency, workability in porting, substitutability

02 Requirements for Developing the Next-Generation Information System

- ① Requirements for the methodologies for developing the next-generation information system
 - Developing it by using the architecture-based latest development methodology (Component-Based Development (CBD), etc.)
 - Developing the next-generation information system based on an iterative development process

03 Analysis of the Next-Generation Information System and Design Requirements

- ① Requirements for analyzing requirements
 - To develop the analytics for the next-generation information system based on object-oriented analysis modeling
 - To be able to partially use a structural analysis model to reflect the characteristics of the work process for the next-generation information system
 - To write the analysis output for the next-generation information system based on the UML modeling output
- ② Requirements for system design
 - To write the next-generation information system by reflecting 'Architecture Requirement'
 - To write the next-generation information system based on object-oriented design modeling
 - To seek to establish the UX-based user-oriented interface to achieve 'Strategies for Establishing the Field-oriented Information System'

04 Software Execution and Testing

- ① Programming language
 - To develop the software by using the object-oriented language of JAVA, a programming language of the Architecture Requirements in the previous paragraph
- ② Core reuse
 - To develop the next-generation information system based on components to increase the code reusability
 - To develop in the direction of code reusability by utilizing refactoring in the development stage
- ③ Software testing
 - To verify and inspect the software components and sub-systems developed by applying the testing strategies, test plans and design process, various testing techniques and levels starting from the project planning stage in order to assure quality in the next-generation information system.

05 Software Management

① Information system maintenance

Applying reverse engineering and reengineering process in order to enhance maintainability after developing the next-generation information system

② Management of information system requirements

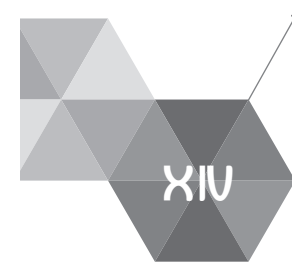
Applying the requirement management process and guaranteeing bi-directional traceability to develop a user and field-oriented system

③ Information system configuration management

Applying the configuration management system in developing the information system to maintain configuration of the next-generation information system, and applying the configuration management process during and after development

④ Information system quality management

Tracing the ISO 9126 requirements in quality attribute requirements among 'Architecture Requirements' to develop a high-quality information system throughout the entire project period, and managing if quality management goals are met



Agile Development

▶▶▶ Latest Trends and Key Issues

The speed of changes in the recent business environment is getting higher and higher. Companies' top competitiveness is now to acutely adapt to and change the rapidly changing business. As such, amid the greater importance of business acuity in the corporate environment, there emerged methodologies to secure acuity in software development and visibility in development progress. The Agile methodology came to emerge against this backdrop, whose importance has rapidly risen. The Agile Development Methodology did not start as denial of the value of the traditional software development methodology or output. Its starting point lies in improving inefficiency in modeling beyond the inherent goal and value from the existing style.

▶▶▶ Study Objectives

- * Able to explain the concept and characteristics of Agile development and types of its methodologies
- * Able to explain the principle of characteristics and execution methodologies of XP, a representative methodology in Agile
- * Able to explain the characteristics, roles, output and process of Scrum, Agile's representative management theory

▶▶▶ Practical Importance High

▶▶▶ Keywords

- XP, SCRUM, TDD

+ Practical Tips

The speed of changes in the recent business environment is getting higher and higher, and companies' top competitiveness is now to acutely adapt to the rapidly changing business environment. With the conventional software development method that focuses on stringent process-based roles and activities, efficiency in development cannot be easily secured. Therefore, by using the frequent check, test and distribution of the Agile Development Methodology to complement these issues, an effective, productive and measurable development environment can be secured. The Agile Methodology can allow for the following effects:

- Risk Management – A project team can identify problems early through the small incremental release method and adapt to changes easily. Clear visibility in the Agile development provides information to make important decisions in the initial stage of development.
- Quality – As tests are developed, acute software development is possible that can be inspected in the execution code level, and product integration can improve. A project team can change the software if necessary at any time, and the team can detect quality problems early resulting from particular changes through the existing testing environment and unit tests.
- Quick Launch – Iteration of Agile Development implies that features are gradually transferred. Products can be continuously evolved and released in the market fast.
- Visibility – The principle of Agile development recommends the user participation throughout the very cooperative shared approach with product development. In other words, it is helpful in validating if project progress and updates are effectively managed that are to provide superb visibility to major stakeholders.
- Flexibility / Acuity – As for Agile development, flexible responses to changes can be made. As development progresses, its duration gets fixed, and requirements become crystal clear and can be further developed. To this end, customers must take part in the development process for proactive negotiations.
- Proper Products – The capability of Agile that evolves and embraces changes lies in team building to make proper products. While conventional development focused on successful project development, Agile must aim to develop a project in a proper manner.

01 Concept of Agile Development

Agile Background

In the mid- '90s, software development methodologies based on the Agile methodology started to emerge. The Agile methodology came into being as a light methodology by staying away from the existing heavy-toned, regulatory methodology. The Agile methodology used to be known as the 'lightweight method', and it ended up being called 'Agile' with the Agile Manifesto. From then on, Agile began to have a meaning as a development method for software development. Compared with software engineering that kicked off in 1945, the Agile methodology is a new domain with a history of only 20 years or so, which is now making a great progress.

Agile Concept

In January 2001, the Agile Alliance made the Agile Manifesto which is an important manifesto that has been discussed as the basic principles and spirit behind Agile software development. The description of the Agile Manifesto is as follows.

<Table 47> Manifesto for Agile Software Development

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on the right, we value the items on the left more."

- 12 Principles behind the Agile Manifesto
 - ① Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
 - ② Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
 - ③ Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
 - ④ Business people and developers must work together daily throughout the project.
 - ⑤ Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
 - ⑥ The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
 - ⑦ Working software is the primary measure of progress.
 - ⑧ Agile processes promote sustainable development. The sponsors, developers, and users should be able to

maintain a constant pace indefinitely.

- ⑨ Continuous attention to technical excellence and good design enhances agility.
- ⑩ Simplicity – the art of maximizing the amount of work not done – is essential.
- ⑪ The best architectures, requirements, and designs emerge from self-organizing teams.
- ⑫ At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Characteristics of Agile

Agile-based software development is iterative and incremental. In order to effectively proceed with these types of development, such techniques as self-organizing and cross-functional teams are used. Agile practices are derived from the process of realizing the value and principles of the Agile Manifesto. The values and principles of the manifesto must be looked into to properly understand the pursuits of the methodology.

① Key Differences between the Waterfall Methodology and Agile methodology

- Plan-oriented vs Customer-oriented

The Waterfall Methodology is to make a timeline for the entire project before starting it and execute it based on the schedule. The Agile methodology, meanwhile, is to devise a plan for a project, but given the uncertainties of the project, it is planned based on the descriptions that are important to customers at this point or those that are confirmed instead of making an unrealistic plan.

- Big Bang Release vs. Small Release

While the Waterfall Methodology is to release all features at a time when a project ends, the Agile methodology iterates small releases continuously in small intervals called 'iterations.' Customers can promptly check out if requirements are reflected through this method.

- Output-oriented vs Software Behavior-oriented

In the Waterfall Methodology, whether or not a project is executed properly is checked out by validating if the output has been written for each planned stage. In the Agile methodology, meanwhile, what is critical is if software properly behaves, and how well it has been developed in line with requirements. Due to these characteristics, it is misperceived that if the Agile is applied for development, there is no need to make a document. It must be understood, however, that it is not that a document is not made but its output can be made in various forms of output suited to different circumstances.

Types of the Agile Methodology

Various Agile methodologies came into being with the launch of the Agile Manifesto, among which, the globally adopted Agile methodologies are Scrum and eXtreme Programming (XP). XP was mostly used at the beginning but as Scrum came to be popular, XP is now used along with Scrum. Recently, the Lean software development methodology has rapidly emerged as a favorable option where Toyota System's Lean Manufacturing System is to be applied to software development, and is now used mostly with Scrum.

Major Agile methodologies are as follows.

- Scrum, Ken Schwaber/ Jeff Sutherland
- Adaptive Software Development (ASD), Jim Highsmith
- Feature Driven Development (FDD), Peter Coad/ Jeff DeLuca
- Dynamic Systems Development Method (DSDM), Dane Falkner
- Crystal Family, Alistair Cockburn
- eXtreme Programming (XP), Kent Beck/ Erich Gamma
- Lean Software Development Methodology, Tom Poppendieck/Mary Poppendieck
- Agile Unified Process (AUP), Scott W. Ambler

02 Agile Development Methodology - XP

XP Outline

eXtreme Programming (XP) is a methodology established based on the lessons that many engineers spearheaded by Kent Beck had learned through projects in the late '90s. It is a light-weight development method suited to small- and medium-sized development units. As for XP, defining it as 'a methodology' is sometimes controversial because it is significantly linked to development techniques such as Test Driven Development (TDD), Daily Build and Continuous Integration. In Korea, it is the norm that a part of the technique is applied especially by small-scale development teams. In more and more cases, such additional Agile development methodologies as Scrum are applied instead of sticking to XP only. XP consists of its values and practices to achieve these values upon its application, and requires a principle to maintain a balance of the two. XP is a type of an iterative development methodology. There are numerous iterations in the project, and tasks are iterated based on test outcome.

Values of XP

Five values are suggested in XP:

- ① Communication: The most important element in software development for each team is communication and a typical cause in project failures is communication errors. Problems must be solved through communication among customers, developers and managers while reinforcing team building.
- ② Simplicity: There is a question that lingers all the time: "What is the simplest of all?" It must be ensured that unnecessary complexities are removed for a simple and clear design.
- ③ Feedback: Making incremental improvement is more effective than pursuing completeness at a time when the output or values of the past become futile amid the flow of changes. Feedback can be produced as much as possible in the fastest manner within the manageable scope for a team.
- ④ Courage: Believing that modifications are reflected and delivered to customers as fast as possible, one must bravely address requirement and technical changes. Courage to proactively address problems pursues simplicity, and courage to make specific suggestions reinforces feedback.
- ⑤ Respect: A project cannot be properly underway if there is no respect for other team members based on the four values above.

Practices of XP

XP suggests 12 practices in <Table 48> besides the five values.

<Table 48> Basic Practice Methods for XP

Type	Practice	Explanation
Development	Simple Design	Voice call using the wired/wireless Internet network and mobile telecommunication network
	Test-Driven Development	Make a test before writing a code and automate it by using a testing tool.
	Refactoring	Improve the design of the existing code by removing the duplication and complexity of code.
	Coding Standard	Make a coding standard for effective communication.
	Pair Programming	Execute development tasks with two developers sitting at a computer together.
	Collective Code Ownership	Make sure all developers in a team are jointly accountable for a source code so that codes can be corrected at any time by anyone to make them better.
	Continuous Integration	Execute integrated tasks to finish tasks.
Management	Planning Game	Devise the entire project plan and cycle plan in consideration of business and technical aspects, and maintain updates through execution and feedback.
	Small Release	Distribute executable modules as fast as possible so that customers could see how software is run frequently.
	Metaphor	Describe the entire picture of a system using easily understandable figures and stories.
Environment	40 hours/week	Do not work for over 40 hours a week to maintain good quality.
	On-site Customer	Ensure that customers that have used the actual system stay on the development field.

<Table 49> Types of Scrum Roles

Role	Explanation
Product Owners	<ul style="list-style-type: none">• Making a product backlog which is a product feature list, and managing the tasks of adjusting the priority or adding new items• Playing core roles in establishing Sprint plans. But product owners are not recommended to be engaged in a team's operation as much as possible.
Scrum Master	Striving to remove obstacles against a team's tasks, and supporting a team to execute development by abiding by principles and values
Scrum Team	Usually consisting of 5~9 people, and devising features to be developed during Sprint by using the user story

Scrum Process

The Scrum process consists of the following three:

- Sprint: referring to an iterative development period of 1~4 weeks according to the calendar
- 3 types of meetings: Daily scrum, Sprint plans, Spring reviews
- 3 types of output: Product backlog, Sprint backlog, Burn down chart

3 types of output can be summarized as follows in <Table 50>.

<Table 50> Scrum Process Output

Roles	Explanation
Product Backlog	It is a priority list of features to be included in products and a product supervisor mostly decides on priorities on behalf of customers. Features defined in a product backlog is called the 'user story.' Estimation of the user workload is based on the standard called the 'story point.'
Sprint Backlog	It is a list to be developed during a single Sprint, and the task is defined as the user story and work to complete it. Each task size is estimated based on time units.
Burn Down Chart	A graph showing the remaining workload to complete development and the remaining workload for each iteration are represented as the 'story point.'

Three types of meetings in the Scrum process can be summarized as follows in <Table 51>.

<Table 51> Scrum Process Output

Roles	Explanation
Sprint Plans	Setting goals for each Sprint, and selecting items to be executed in the Sprint from a product backlog. Operators are allocated for each item, and task-specific plans are devised.
Daily Scrum	It is a daily meeting to share project updates for 15 minutes. Every team member takes part to talk about what they have done, what they are to do and problems.
Sprint Review	It is a meeting to check out task progress and outcome if Sprint goals are met. Scrum Team demonstrates the outcome generated during Sprint to participants and receives feedback. It would be desirable if the demonstration on all tasks executed during Sprint and participants could participate. Here, Scrum Master can conduct a 'project retrospective' to find out what went well, what was regrettable and needs to be improved.

03 Scrum

Outline of Scrum

Scrum is an Agile methodology for project management and is a representative form of the estimation and adjustment-based experiential management technique. It is derived from an article titled "The New Product Development Game" at HBR by two professors, Ikujiro Nonaka and Hirotaka Takeuchi in 1986. The method was introduced to software development by Ken Schwaber and Jeff Sutherland in 1995 who called it 'Scrum.' Scrum has the following three roles in <Table 49>.

Characteristics of Scrum

Characteristics of Scrum are as follows:

- **Transparency:** It might be difficult to accurately identify at what state the current project is, if everything is going as planned, and what problems there are, Scrum effectively identifies the state or problems of a project using such techniques as Scrum Meeting, Burn Down Chart and Sprint Review.
- **Time Boxing:** It is possible to concentrate on proceeding with a project by constraining the time needed to carry on with Scrum. Daily Scrum is carried out within a limited timeframe of 15 minutes every day, while Sprint Review is periodically executed for each iteration.
- **Communication:** Numerous efforts are poured in to facilitate cross-team member communication. Sharing problems confronted by developers in Daily Scrum, and discussing the level of difficulty/time in implementing the user story using the Planning Poker is one of the characteristics to facilitate communication among team members.
- **Empiricism Model:** Scrum has its own process models, and many techniques emphasize the experiences of individuals taking part in projects. Because each project has unique circumstances and characteristics, the existing standardized process is not sufficient to track the circumstances of these projects. Scrum recognizes that the basic structure remains the same but Scrum might differ by team in reality.

Example Question

Question type

Short-answer question

Question

Scrum as an Agile methodology for project management consists of three roles. What is the name of 'a role that removes disruptions for a team's tasks, sticks to principles and values and helps it to carry on the development?'

Intent of the question

Understanding the roles of Scrum

Answer and explanation

Scrum master

Related E-learning Contents

- **Lecture 15** Agile Development

Mobile Computing

▶▶▶ Latest Trends and Key Issues

With the emergence of Android and iPhones, the impact of mobile computing has gradually intensified. The penetration rate of mobile computing has skyrocketed with strengths in mobility, convenience and accessibility, and the impact of computing, especially in social deals and SNS, dominates the desktop computing. Companies have recently and strategically emphasized the mobile computing-based mobile service in order to acquire competitive edges in business.

▶▶▶ Study Objectives

- * Able to explain the concept and characteristics of mobile computing
- * Able to understand strategies to set up mobile computing and the process to develop mobile apps

▶▶▶ Practical Importance High

▶▶▶ Keywords

mobile computing, mobile application, mobile computing service

+ Practical Tips

The mobile transaction volume for all product categories in e-commerce in Korea has skyrocketed, and in the fashion field, over 50% of transactions take place on mobile platforms as of 2015. In non-fashion fields, the mobile transaction volume is over one third, and the mobile transaction volume and growth rate outperform those of the desktop.

For domestic mobile devices in Korea, the portions of Android and iPhone are dominant. The mobile portion in e-commerce will be even higher driven by high-performance smartphones, a superior network environment, convenience in payment via smartphones and customized curated services for mobile users. Consequently, companies planning to launch B2C services are tapping onto 'mobile-first' strategies. More companies provide the mobile service even before developing services for the desktop. The importance of mobile computing in the B2C market will be higher and this trend will be more intensified down the road.

01 Outline of Mobile Computing

The mobile concept refers to a capability to enable ubiquitous portability and exchanges of data using mobile telecommunication devices without wire connection and to execute digital data exchange in real time on the move using portable devices. Characteristics of mobile computing include mobility, personality and presence. One of its element is the ubiquitous environment enabling free Internet access with 'three any's': anytime, anywhere and any device. There are four components that make the mobile service environment: devices, network, platforms and content.

- Devices: mobile phone, PDA, smart phone, DMB device, PMP, eBook
- Network: mobile telecommunication, satellite communication, wireless LAN, portable Internet, etc.
- Platforms: mobile operating system, VM (Virtual Machine), browser
- Content (application programs and data):
 - o Mobile information

- o Mobile entertainment
- o Mobile communication

Mobile content refers to digital content used in a mobile telecommunication environment including texts, sounds, videos and games. Characteristics of mobile content include communicability, instant connectivity, localization and personalization. However, the size, functions and quality of content are limited due to constraints with device functionalities and performance. Mobile content can be used relatively for a short time span and the usage environment is also limited. Mobile content can be divided into four types: information service, mobile entertainment, mobile communication and mobile commerce.

- Information service: informative content
 - o General information including news, weather and sports
 - o LBS including maps, transportation information and GPS
- Mobile entertainment: entertaining content
 - o Games, ring tones, characters, VOD, DMB
- Mobile communication: sending messages or emails, etc. via mobile devices
 - o Text messages, multimedia messages, instant messages, video calls, community services, SNS
- Mobile commerce: e-commerce including ads, payment and shopping
 - o Mobile ads, mobile payment, mobile shopping

The value chain in the mobile market consists of the following: device makers, carriers/telco's, solution providers and content suppliers, etc.

- Device makers
 - o Designing/Manufacturing mobile phones, smart phones, and devices for DMB and PMP
- Carriers/Telco's
 - o Mobile carriers in Korea: SK Telecom, KT, LG Telecom
 - o Satellite DMB service is provided by TU Media, and terrestrial DMB services are provided by six companies including KBS
- Solution Providers
 - o Platform developers: developing VMs like WIPI, iPhone, Android and mobile browsers
 - o Developers of multimedia component technologies: audio, video, graphics, etc.
 - o Developers of retail management technologies: providing content security and protection, etc.
- Content Providers/SPs:
 - o Directly executing services by developing mobile content
 - o Providing developed content to mobile carriers

Technological development of mobile phones is from first-generation simple (or normal) phones to second-generation feature phones to third-generation smartphones. Smartphones are almost like mobile computers. The first-generation simple phones only enabled voice communication at first, were in black and with low resolution specifications embedded with real-time OS. The second-generation feature phones were mostly convergent with MP3 phones, DMB phones and camera phones. They had such specifications as colored LCDs and high resolution, and were embedded with real-time OS. The third-generation smartphones implemented the same features including Internet, email and fax like PCs. With a large LCD touch screen and a general-purpose OS are embedded, they play the roles of mobile computers equipped with such features as information search online and data transmission. In the world of smartphones, 'mobile user experiences', that is, content capable of communicating with five senses of users are spotlighted, being used through gesture recognition, multi-touch, vibration feedback and tactile feedback. Content services enabling stimulating users' tactile sense and vision so that they could be immersed into content are being

developed beyond merely responsive content in the mobile environment. Types of mobile OS include Symbian OS, Windows Mobile OS, iPhone OS, Android, Blackberry OS and Palm OS.

Recently, mobile computing has been emphasized as a means of strategy to secure corporate competitive edges. Companies need to, first and foremost, think of ways to utilize mobile computing to steadily secure their competitive edges. Mobile computing is proactively utilized to seek for corporate survival and growth, and secure competitive edges in the competitive market. It serves as an essential survival competency for companies instead of being a target of investment.

Characteristics of Mobile Computing

It would be essential to look into characteristics of mobile computing in order to strategically tap onto it.

- Ubiquity

Ubiquity is the most outstanding advantage of using mobile devices. Using wireless devices in forms of smartphones or communicators, users can do information search and use communications service anytime and anywhere.
- Reachability

Reachability or accessibility is a major feature for communication among users. Users with a mobile device can log in anytime, but reachability can be limited to designated people or time for particular cases. Reachability is gaining more and more importance in line with recent trends such as individualization and diversification.
- Security

Wireless communication security technologies are specialized in the form of secure socket layers within the closed end-to-end system. In Europe, a higher level of security is guaranteed in the wireless environment compared with the wired Internet network because the smart card attached to a device and the Subscriber Identification Module (SIM) card are to be authorized proprietarily by users that hold them.
- Convenience

Convenience is an attribute of wireless devices, and many related features are made available based on users' needs as the data storage capacity gradually increases. With the emergence of diverse wireless devices, the usage will be made more convenient. The characteristic as such will be in line with the technological advancement of hardware, and will be further strengthened through an improved device size, increased capacities for batteries and storage, and diversified wireless devices and features
- Localization

This feature can upgrade features of wireless devices by coupling location information to wireless services. Once users' location at a certain point is identified, appropriate services can be offered to arouse a desire to have transactions with users. Various services can be provided including sending hotel information in a city to people that have arrived at an airport, which is available now in some regions, and sending discount coupons for restaurants and clothing stores which users might be interested in via wireless Internet when they are at a certain space in a specific shopping mall at a particular time.
- Instant Connectivity

Users can immediately access the Internet anytime and anywhere via wireless devices. Communication services in such packet modes as GPRS and IS-95C will be enabled, so a mobile phone could be used to access the Internet conveniently and fast without having to connect to a communication network for log-in.

- Personalization
It is to provide customized content befitting individual users based on the information already provided to them or the pre-determined content. Currently, a limited scope of services including the validation of purchase through credit cards are available. However, the level of personalized information and transaction handling can be upgraded through a wireless portal site, while wireless devices can serve as essential instruments in daily life.

02 Mobile Computing Development Process

Mobile application development is significantly different from ordinary web development because an integrated mobile development process is applied.

- Possible to be used anywhere and at once (integrated to be used in all mobile devices and OS' where Apple iPhone, RIM Blackberry, Window Mobile, Google Android and Farm are included)
- Supporting all devices and channels which customers use
- Newly launching and upgrading mobile applications within weeks or months, if necessary
- Resources and outsourcing to achieve predictable development cost and an efficient cost structure


Major companies develop applications every three months to be distributed to all channels. 'UI design' is located between 'design' and 'programming.' UI development is to connect the design image in the affective domains and manipulability, and serves as a bridge between the subjective and the objective, and between the affective and the logical in a process. It is a series of process starting with ideation, and then leading to verification of feasibility, production of icon design and resources, design of UI and architecture, programming and completion and distribution of an application. It represents that the development task has been completed by connecting the beginning with the end. In the next section, the development process is explained in more details.

- Ideation (defining product concepts)
Ideation is a stage of planning/designing the concept of an app, user research, research analysis, and competitor app analysis. It is the most essential stage of cross-departmental idea exchanges and in this stage, once an accurate product concept is confirmed based on the requirements of user research or customers/ internal departments, specific planning is required and cross-departmental idea exchanges. Overall strategies are devised on what forms an app would have and through what forms it would be sold. The bottom line is that the app concept/specifically defined content is communicated to all participants.
- Design
Once the app concept and specifics are defined, a big frame of the app must be designed. The app concept might differ as development proceeds, so the design might be subject to change after designing it first.
- UI Design
In order to develop the mobile UI for a particular app project, information of a platform of target devices, specific information and a design guideline can be identified, design strategies can be set, and prototyping can be executed many times. Accordingly, it is reviewed if the concept can be reflected or implemented.
- Programming (Development and Test)
Programming is the actual development stage where the participation of staff involved in marketing/planning/ sales is relatively low, it is always needed to share individual builds in the middle of development, and have meetings/validation through the developed builds so that the tasks are not carried on in a wrong direction. It is

critical to execute an app quality test to secure app stability and to provide a favorable app to actual users. If a preliminary user research has been conducted, it would be desirable to conduct a user research so that users could actually use an app that has been somewhat completed.

- Feedback
Once an app is completed after development and testing, it would be subject to promotion and sales. If the customer feedback is positive in a country after it started to sell there, it can be sold in many other countries. One of the myths is that mere translation of a menu would still gain the same popularity in other countries, but since each country has different cultures and lifestyles, simple translation of a menu is likely to fail. It is desirable to take each step at a time as if to make a new app from the beginning.


Example Question

 **Question type**
Multiple choice question

 **Question**
Describe all the characteristics of mobile computing.

- ① Mobility
- ② Personality
- ③ Presence
- ④ Localization

 **Intent of the question**
To understand characteristics of mobile computing

 **Answer and explanation**
1, 2, 3, 4

Related E-learning Contents

- **Lecture 16** Mobile Computing

Cloud Computing

▶▶▶ Latest Trends and Key Issues

The conventional IT environment that used be based on high-performance serversmostly represented by the mainframe and the Unix server has rapidly replaced by a cloud computing environment with virtualization, presence and linear scalability. In particular, the importance of cloud computing is likely to go up, especially when the enormous growth of SNS and online services, big data and IoT have been established as keywords to describe Cloud computing must be fully understood to promptly respond to the rapidly changing business environment and provide continued scalability in the IT environment.

▶▶▶ Study Objectives

- ✧ Able to distinguish and explain the characteristic, concept and types of cloud
- ✧ Able to explain the virtualization technology which is the backbone of cloud computing and its components
- ✧ Able to distinguish types of the virtualization technology in virtualizing servers for cloud computing and explain the differences

▶▶▶ Practical Importance High

▶▶▶ Keywords

Cloud computing, types of cloud, virtualization

+ Practical Tips

Developing services fast in the rapidly changing business environment and scaling out the system promptly depending on market responses is recognized as a critical corporate competitiveness. Once cloud computing is used, acute infrastructure can be secured, and it is possible to secure and scale up resources in a scale in need at a point in demand. Companies that use a cloud can effectively maintain the infrastructure operating personnel, and maximize the efficiency in system management. Cost-efficient infrastructure for scale-out and scale-in can be secured at a time in need with the minimum number of infrastructure operators. In case companies prepare for a global service, they can utilize the infrastructure of providers of cloud computing so that hub services per continent can be effectively configured. Start-ups can configure a system acutely with minimum cost using a public cloud in a cloud environment and scale it out depending on the business circumstances. Conventional companies can also secure operating efficiency by configuring a hybrid environment of the existing computing resources and a cloud. Cloud computing is recognized as an essential technology that is acute and scalable and forms efficient infrastructure.

01 Definition of Cloud Computing

Cloud computing was developed to efficiently utilize a surplus computer capacity in individual servers and flexibly respond to the uncertain service demands. Prior to the emergence of cloud computing, users had to own and manage resources to utilize computing resources. However, it has been shifted to the mode of using computing resources provided via Internet in a virtualized form when needed in the recent cloud environment. The key point in a cloud is that because computer resources are provided in an invisible state as they are accumulated in a network, users can do a job using computer resources in demand anywhere without knowing the complexities therein. Various services started to be launched in 2008 or so to utilize computing resources online, and the word 'cloud computing' to encompass it all came to be used. Consequently, users could use services they wanted from the Internet without expert knowledge on technical infrastructure available. Expected results in adopting cloud services are as follows.

- Cost: CAPEX¹⁾ reduction, OPEX²⁾ increase, TCO³⁾ reduction
- Period: shortening the development period and the product development cycle
- Operation: reducing the operating personnel and strengthening efficiency of resources
- Products: boosting the product concentration level

The words ‘cloud service’ and ‘cloud computing’ are usually used as synonyms but the Telecommunications Technology Association defines the two words as follows:

〈Table 51〉 ‘Cloud Computing’ and ‘Cloud Service’

Terminology	Definition
Cloud Service	On-demand outsourcing IT service providing a user-oriented cloud computing environment
Cloud Computing	A computing environment where IT resources are leased via Internet based on virtualization and distributed computing technologies and charged to the extent of usage

Cloud Computing vs. Other Types of Computing

① Grid computing and cloud computing

Grid computing defined as ‘a structure to virtualize and integrate resources including a computer or data in a network, and to dynamically generate a virtual computer, if necessary.’ Grid computing and cloud computing are similar in that they use a distributed computing structure and provide virtualized computing resources. Yet, while grid computing uses all computer resources in the Internet, cloud computing only uses enterprise-owned computing.

〈Table 52〉 Distinctions between Grid Computing and Cloud Computing

	Grid Computing	Cloud Computing
Location of a Computer	Geographically distributed and managed by separate units	Geographically distributed but managed by a central unit
Composition of a Computer	Presence of heterogeneous types	Mostly same models
Standardization Organization	Available	Not available
Technical Standard	Standards available on resource management, scheduling and data management security, etc.	None
Interconnectedness	Critical	Not considered
Usage	Applications with a high level of parallelism including scientific/technical calculation and large-scale arithmetic operations	Widely used including web applications

1 CAPEX: Cost spent to generate future profits
2 OPEX: Operating cost
3 TCO: The total cost including the cost for system adoption up to maintenance

② Utility computing and cloud computing

Utility computing is a type of providing computing resources by billing the amount based on the amount of computer resources used. Billing on the resources used is a commonality of utility computing and cloud computing. Cloud computing is a developed version of abstraction on computing resources based on utility computing.

02 Types of Cloud Computing

Classification Based on Service Types

Cloud service can be divided into IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service) based on the types of service.

① IaaS (Infrastructure as a Service)

IaaS is to provide infrastructure resources (server, storage, network) via the Internet network. Infrastructure resources provided at IaaS are done so in a virtualized environment but the virtualization environment is not essential for IaaS. The form of providing physical infrastructure resources that are not virtualized is called ‘bare-metal.’ IaaS users must manage domains beyond the OS directly. It is similar with users receiving hosting from the existing service provider.

② PaaS (Platform as a Service)

PaaS is a type of service where the concept of SaaS has been applied to the development environment as a type of using a necessary development and operating environment in the form of service without having to establish a development or operating environment. PaaS considers the application development environment and the service distribution and operating environment as the service scope. PaaS provides the network infrastructure and a runtime to execute applications. The scope of management for users includes applications and data that are on top of the application runtime.

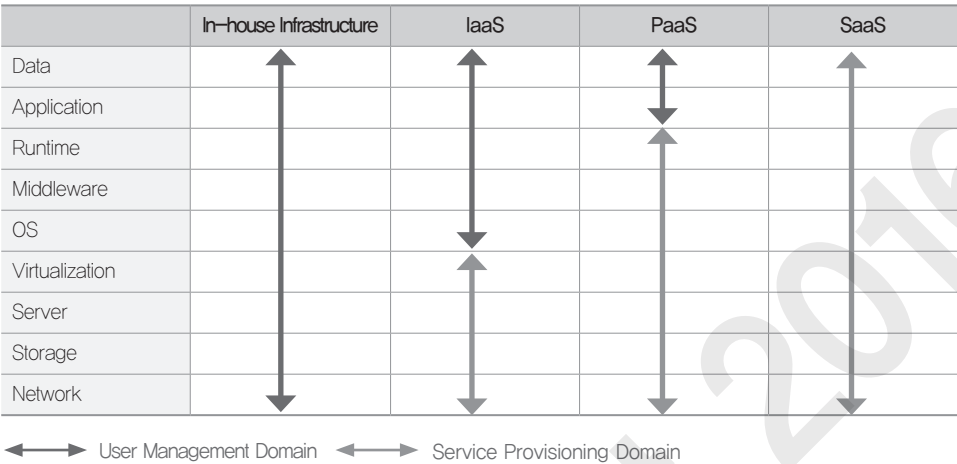
For instance, PaaS can be used that provides MySQL and Apache Tomcat in development a new Java web application using MySQL as database so that a development/operating environment can be prepared for promptly while enabling prompt application development/services.

③ SaaS (Software as a Service)

SaaS is a service to provide software as a model where the software made available at the center can be transferred to users through such clients as web browsers. SaaS users can use the necessary functions without having to know where the software being used is located, what OS is used and through what language, it has been developed. Examples of using SaaS include Google Docs and Salesforce.com. Usually SaaS has the following characteristics.

- Access via web browsers: accessing via web browsers instead of installable software
- Cost based on the amount of usage: paying cost to the extent of software usage
- On-demand: possible to immediately use necessary software

- Optimization of IT demand: no issue in terms of managing and scaling out IT infrastructure
- The scope of management for users and service provides for IaaS, PaaS and SaaS are as follows in [Figure 22].



[Figure 22] Scope of User Management by Cloud Service Type

Classification Based on Cloud Operation Forms

Cloud services can be classified into three types as follows based on the forms of service operation and the scope of disclosure.

〈Table 53〉 Types of Cloud Services Based on the Scope of Disclosure

Cloud Service	Explanation
Public Cloud	A cloud service accessible through the Internet access, being opened to unspecified individuals online
Private Cloud	A cloud service allowing for access to a limited group of users by establishing a cloud service in a closed network in companies and institutions
Hybrid Cloud	<ul style="list-style-type: none">• A combination of a public cloud and a private cloud, using both an externally disclosed cloud service and a closed one• Interoperability is very important so that a service distributed to a public cloud could be migrated to a private cloud and vice versa.

03 Server Virtualization Technology

It is the norm for the existing computing environment to have a single OS dependent on a single hardware, and run an application on the OS. Virtualization as a mode of virtualizing computer resources refers to a technology that enables multiple computer resources to seem like a single server, or a single computer resource to seem like multiple

computer resources. Once the server virtualization technology is used, the utilization rate and management efficiency of computer resources can be maximized because it is possible to operate more than one OS on a single computer. Normally a cloud service uses the virtualization technology and provides computer resources that have become abstract. In particular, x86 Server Virtualization Technology has been rapidly disseminated driven by the mature technology and market, logically paving the way for a cloud computing environment for sharing and distribution of computing resources beyond physical limitations.

Hypervisors

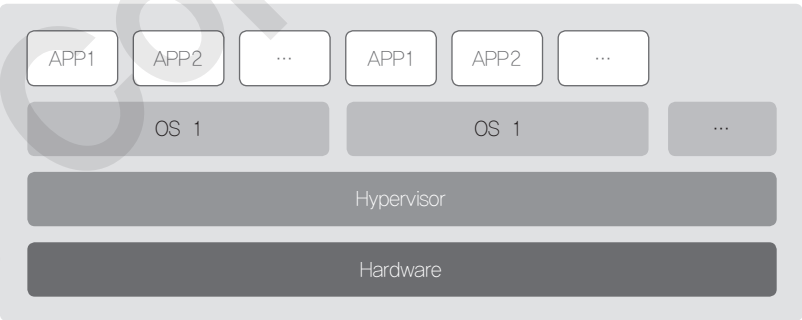
A host OS refers to a server that physically exists in server virtualization, and the virtually provided server is called a 'guest OS.' Server virtualization technologies can be distinguished based on how to allocate physical resources that a virtual server could look like a physical one. A technology to virtualize a physical server is called a 'hypervisor', which is a logical platform for server virtualization. A hypervisor is also called a 'virtual machine monitor(VMM).' A hypervisor can be classified depending on types of installation and types of virtualization.

Types of Hypervisors

Hypervisors can be classified into native and hosted hypervisors depending on the types of installation. Native types are installed directly in hardware, while hosted types are installed on top of an OS like an ordinary program [Figure 3].

① Native Hypervisors

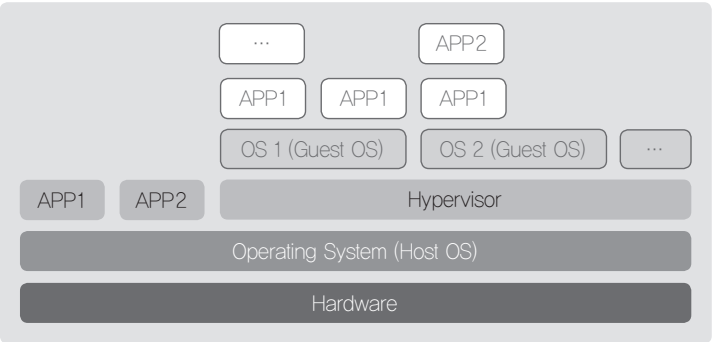
It is a type where a host OS is not needed because the Virtual Machine Monitor (VMM) is directly installed on a physical hardware. Since a host OS is not needed, resources can be saved compared with the virtualization mode where a host OS is needed. The native type is also called Type1. Virtualization technologies in a native type include Xen, Citrix' XenServer, VMWare's ESXServer, IBM's Power Hypervisor and Microsoft's Hyper-V and KVM.



[Figure 23] Native Hypervisors

② Hosted Types

A hosted-type hypervisor is a software installed on the existing OS. These types of hypervisors include Microsoft's Virtual PC, VMWare's Workstation and Oracle's VirtualBox.



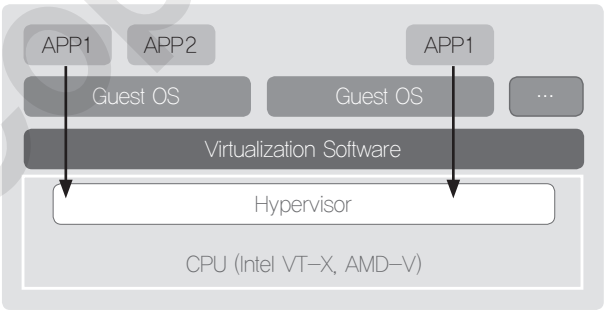
[Figure 24] Hosted-type Hypervisor

Types of Server Virtualization

Depending on virtualization modes, server virtualization technology can be broken down into Full Virtualization, Para-Virtualization and OS-level Virtualization.

① Full virtualization

It is recognized that a guest OS directly owns and access a hardware as it is fully virtualizing the hardware, but in fact, a virtual server uses the resources where a hypervisor did emulation for the hardware. When a guest OS is to dominate the hardware, a hypervisor has to process this command separately. In order to fully emulate the hardware resources except for the CPU, a CPU to support virtualization (Intel’s VT, AMD–V) is needed. The performance has significantly increased because the hypervisor included in the CPU does processing fast. Since full virtualization is processed in the CPU, the guest OS does not have to be modified. Therefore, there is no constraint on the guest OS, Linux-based guest OS and the Windows Guest OS can be run at the same time. Normally, a native-type hypervisor is referred to as full virtualization.

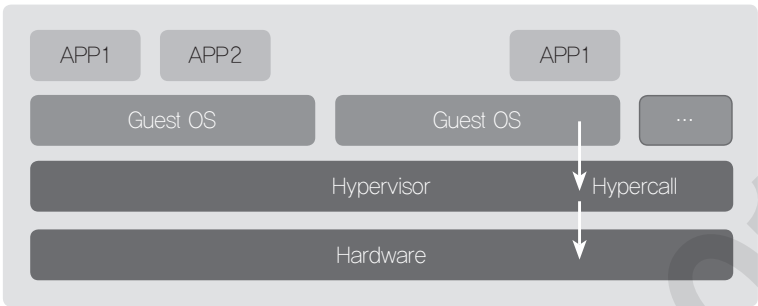


[Figure 25] Concept Map of Full Virtualization

② Para Virtualization

Para virtualization is different from full virtualization of hardware, and in order for a guest OS to access hardware resources, it has to pass through a hypervisor. It is to use a hypervisor API without hardware emulation. As for the OS of a guest OS, its kernel must be partially modified to use the hypervisor API. The advantage is that a high performance is guaranteed because the control is through the hypervisor without

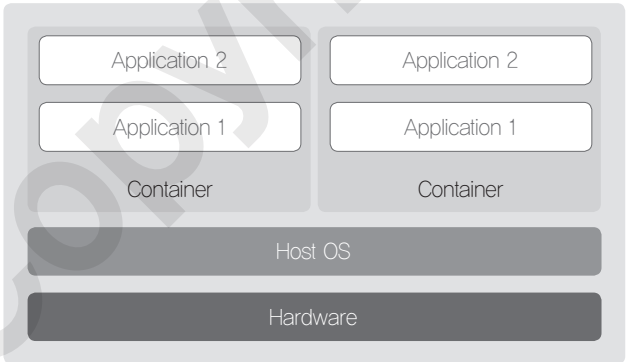
going through hardware emulation. Yet, the pitfall is that a part of the kernel of the guest OS must be modified, so only the open source OS can be used as a uest OS.



[Figure 26] Concept Map of Para Virtualization

③ OS-level Virtualization

It is a virtualization technology to make believe as if the same OS is being used in an OS. It uses a virtualization technology included in the OS without using a hypervisor. Currently, a container technology that allocates and isolates the resources included in the OS. Representative technologies here include Solaris Containers, FreeBSD Jails and Linux Docke. Since the OS’ container feature is used without using a hypervisor, a guest OS must be identical to a host OS. Containers that get to be on the OS are basically allocated with such resources as memory, storage and network. Each of them is operated between containers independently.



[Figure 27] OS-level Virtualization

Container-type virtualization technology, compared with the existing virtualization technologies, is light and superior in portability. In Linux Containers(LXC), there are the ‘kernel space’ to manage physical resources and ‘user space’ to execute the user process in the OS. Container-type virtualization technology is operated by dividing the user space into 10, and limiting the resources seen in each user process.

〈Table 54〉 Hypervisor Virtualization vs. OS-Level Virtualization

Item	Hypervisor Virtualization	OS-Level Virtualization
Hardware Independence	Fully independent within VM	Using the host OS
OS Independence	Fully independent from the host OS	The host OS and the guest OS are the same.
Performance	<div><div>• High overhead</div><div>• Applying hardware virtualization for a higher performance</div></div>	No overhead
Management	Separately managed per VM	Centralized management of shared software
Applications	<div><div>• Heterogeneous integration</div><div>• Linux & Windows integration</div></div>	Resources integration of a single OS environment
Virtualization Technology	Xen, MS Virtual Server, KVM	Solaris, LXC (Linux Container), Docker

OS-level virtualization (container virtualization) has the following strengths.

- Fast start and closing: very fast compared with hypervisor-based virtualization
- High level of integration: minimizing resources to be consumed because there is a single OS upon operating many containers
- Low overhead: isolating the user space without emulation
- Supporting application containers: supporting container configuration per application

OS-level virtualization has the following weaknesses.

- Dependent on the host OS
- Impossible to form a container-specific kernel

04 Storage and Network Virtualization Technology

Storage Virtualization

A method called a ‘thin provisioning’ is used instead of a storage space so that the minimum required space in the initial stage can be virtually allocated to implement a service. Moreover, an environment to use heterogeneous storage integration is made available.

Network Virtualization

L2/L3/L7 switching, network firewalls and security device that used to exist in the form of hardware appliances are implemented as virtual machines, and networking resources are to be separated and manipulated internally through virtualization in a single shared physical environment.

Example Question

Question type

Essay-type question

Question

Server virtualization technologies are divided into full virtualization and para virtualization depending on virtualization modes. Select a mode of virtualization mode where kernels have to be modified, and explain advantages of the virtualization model that has been selected.

Intent of the question

To understand the operating mechanisms of full virtualization and para virtualization

Answer and explanation

1. A mode of virtualization where kernels have to be modified: para virtualization
2. Advantages of para virtualization: higher in speed because a hypervisor directly controls resources instead of the OS controlling them

In explaining an answer in Question 2, that the hypervisor directly controls resources and is fast must be included.

Software Product Line Engineering

▶▶▶ Latest Trends and Key Issues

Amid the increases in the usage frequency and dissemination of software, it has become more important to raise the software quality and productivity. To this end, similarities and differences in the functional aspect belonging to similar domains must be analyzed to secure core assets and improve on the development period and quality in developing software through the assets. This is what the software product line engineering is all about. If software product line engineering is applied, such benefits as quality improvement, higher productivity and shortening of product launch time can be achieved.

▶▶▶ Study Objectives

- * Able to explain the concept of software product line engineering
- * Able to explain advantages of software product line engineering

▶▶▶ Practical Importance Low

▶▶▶ Keyword

- software product line engineering

+ Practical Tips

Various attempts are made to shorten the period for software development and secure higher quality at the same time in the rapidly changing business environment. Software product line engineering is a software development paradigm that considers both the reuse techniques, and business profits and technical strategies. Software product line engineering is a technique to adopt a production line engineering in manufacturing companies and strategically pursue the reuse in large units aligned with business strategies.

If the concept of the reuse repository concept is lacking in software product line engineering, the already implemented features would have to be repeatedly developed whenever new products are developed, and architecture would have to be designed anew each time, wasting corresponding time and cost. By contrast, if software product line engineering is applied, there would occur the development time and cost to set up a reuse repository, but over the long term, the reuse rate would increase, being a merit in terms of time and cost.

01 Outline of Software Product Line Engineering

The software penetration rate and complexity have rapidly increased recently. The development period for high-quality software would be relatively higher. Raising the quality and productivity of software would require the adoption of reuse repository so that the component reuse rate could go up. Software product line engineering came into being as a development methodology in the '80s, aiming to maximize the productivity and quality of developing software by developing, managing and reusing common features such as core assets of software by analyzing the common aspects and variable aspects of functionalities in many systems in a domain so that component reuse rate could be boosted. Software product line engineering is a methodology to satisfy high quality and productivity of software by focusing on reusability. Software product line engineering boils down to the addition of domain engineering to the CBD methodology. Software product line engineering makes development easier by standardizing CBD components as domain units. While CBD is simple and suited to the broad implementation of applications, software product line engineering is more suitable for package software or embedded software.

02 Components of Software Product Line

The software product line consists of three core activities: core asset development, product development and product management. Developing core assets is a development stage of discovering commonalities of products through repeated execution and of enhancing productivity in developing the existing products. The product development stage is to utilize core assets in developing core common features in products to be developed, thus shortening the production period and systematically executing product development. Lastly, 'management' here refers to technical/organizational management including having a proper organizational structure, allocating resources, mediating and supervising, training, giving rewards and devising plans for the organization. Software product line engineering consists of two steps: domain engineering and application engineering. Domain engineering is to make product line assets by analyzing commonalities and differences of products included in a particular domain, consisting of architecture design and component design. In architecture design, design decisions are made in a broad sense, while in component design, design decisions are made in a narrow sense. In component design, reusability and assembly are considered to refine architecture components, enabling products planned for a product line to be assembled from reusable asset components. Application engineering consists of activities to produce particular products which customers want by using product line assets made in domain engineering. Codes of final products are generated by analyzing product requirements based on customer requirements, using corresponding software product line assets, and comprising of components and architecture.

03 Software Product Line Engineering Process

Software product line methodologies consist of two steps: domain engineering making product line assets (PLA) by analyzing commonalities and differences, and application engineering producing particular products customers want using the PLA. Domain engineering is also divided into domain modeling and component modeling processes. Domain modeling consists of the following: context analysis to configure the scope of domains, feature modeling to analyze commonalities and differences of systems within domains in the feature aspect; operational modeling to model internal features and behaviors of systems within domains, and architecture modeling representing the upper design structure of systems within domains. Meanwhile, component modeling is a course of developing reusable components commonly used among systems within domains. It consists of the following: candidate object identification finding objects as basic units of components, object component development defining the interface and implementations of objects based on candidate objects, and reusable component development enabling the developed object components to adapt to different environments. In application engineering, final software is developed through the five-step process: feature selection, object selection and response, model testing, architecture selection and code generation. Feature selection is a process of eliciting user requirements, and object selection and response is a process of eliciting an overall analysis model by configuring a response relationship between objects elicited through the feature selection and requirement analysis modeling. In the stage of model testing, the analysis model elicited is simulated, and errors in the model are traced and corrected through logic testing. When this stage is done, the next stage is to select a design model. Software systems in various structures can be developed by selecting specific architectures fitting user requirements among many reference architecture models elicited from domain engineering.

Lastly, the final application software is developed through a code generation stage that automatically generates codes based on the predetermined architecture platform.

04 Advantages of Software Product Line Engineering

Since software product line engineering induces broad reusability, the reuse rate in organizations where this methodology has been successfully applied is over 80%. This, at the end of the day, can save efforts and cost. In organizations where software product line engineering is successfully applied, the cost saving effect is over 50%. Software product line engineering will bring about the following benefits.

- Higher productivity in development
- Higher quality in software
- Shorter product development period
- A smaller number of personnel required

With the reuse rate going up, such benefits as the efficiency in handling defects and saving maintenance cost can be achieved. That is why not only large enterprises including Nokia, Cummins, Siemens and Thales but also SMEs prove the success of software product line engineering in various domains.

Example Question

Question type

Multiple choice question

Question

Select all the benefits in applying software product line engineering.

- ① Higher quality
- ② Higher productivity
- ③ Technology internalization
- ④ Saving the preparation time for product launch

Intent of the question

To check out if one understands the concept of software product line engineering

Answer and explanation

1, 2, 3, 4