



Chapter 4

Development Technology



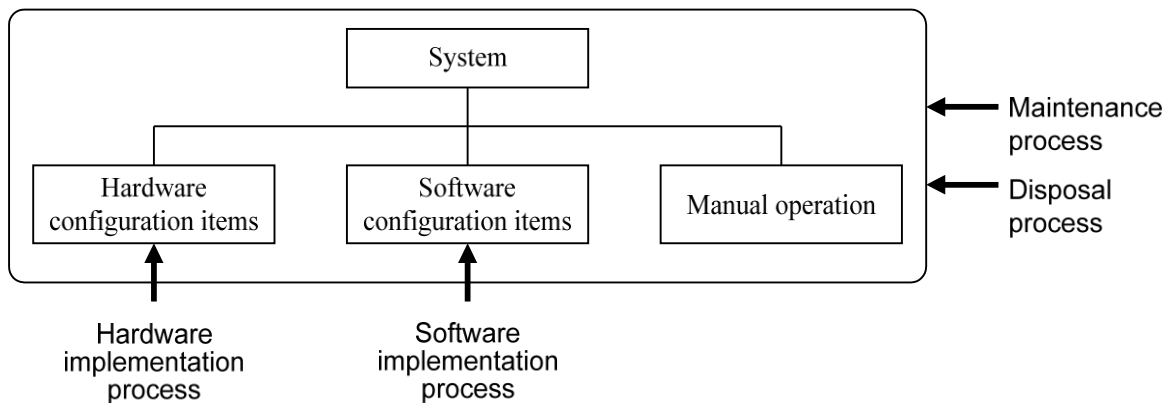
1 System Development Technology (SLCP)

System development technology refers to technology for developing the individual information systems clarified in information system planning.

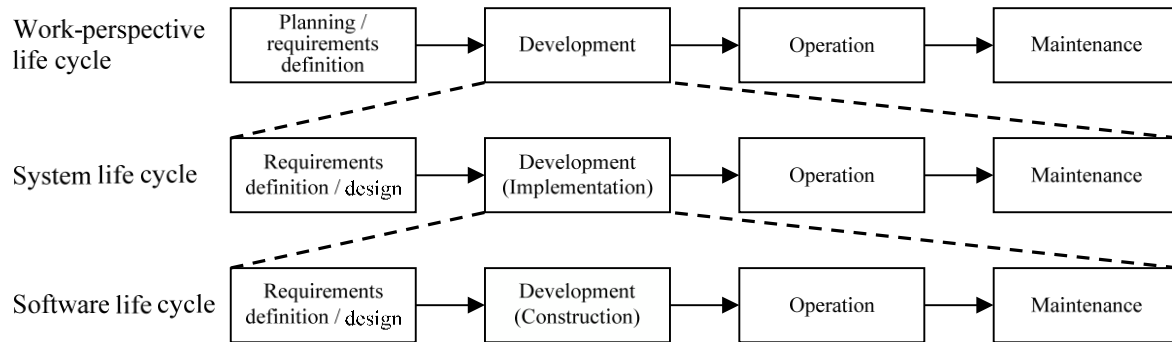
Common Frame 2013 (SLCP-JCF2013) defines processes for which engineers are the main entity within the system life cycle spanning the development to the disposal of a system as follows:

Process Name	Overview
System development process	Clarify system configuration items to be developed.
Hardware implementation process	Develop hardware configuration items.
Software implementation process	Develop software configuration items.
Maintenance process	Maintain developed systems.
Disposal process	Dispose of developed systems.

System development process



On the basis of this concept, the relationships among the work-perspective life cycle, the system life cycle, and the software life cycle can be summarized as shown below. In order to make the relationships easier to understand, feedback from maintenance is omitted here.



This section describes work items and techniques in the system development process, the software implementation process, and the maintenance/disposal process, on the basis of Common Frame 2013.

1 - 1 System Development Process

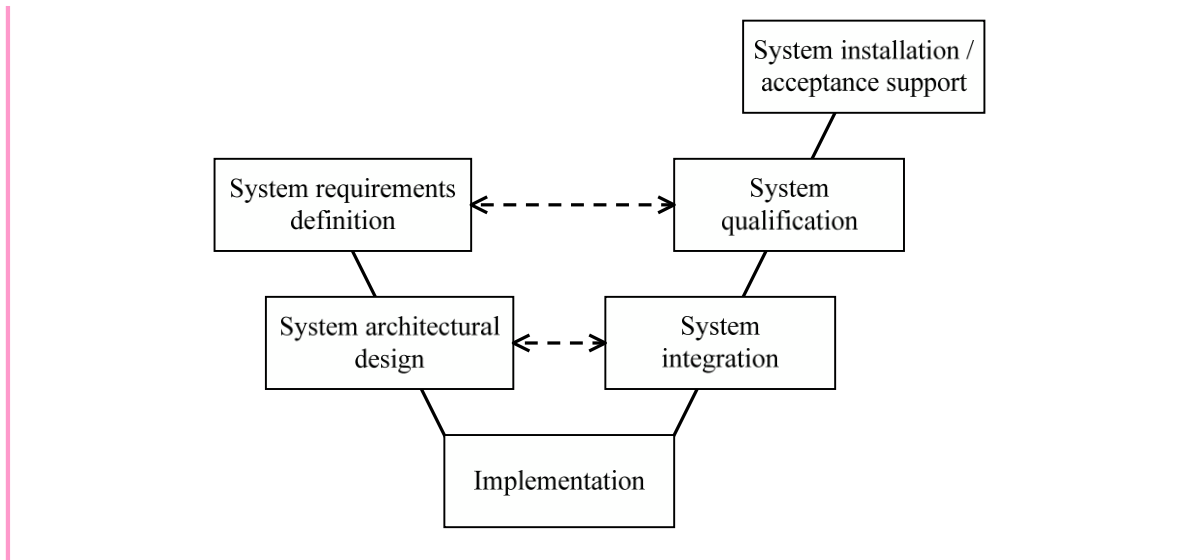
In the **system development** process, the hardware, software, and other configuration items that compose the system to be developed are clarified, and the target system is developed by combining the individual configuration items.

[Objectives of the system development process (excerpt)]

The system development process is aimed at the development of systems, software products, or services. It converts requirements into the system that matches customer needs.

[Processes within the system development process (excerpt)]

- (i) System requirements definition process
- (ii) **System architectural design process**
- (iii) Implementation process
- (iv) System integration process
- (v) System qualification testing process
- (vi) System installation process
- (vii) System acceptance support process



Here, **system installation** refers to the delivery of a system by the supplier (i.e., developer) to the acquirer (i.e., purchaser). Furthermore, **system acceptance support** refers to support by the supplier to enable acceptance of the delivered system by the acquirer.

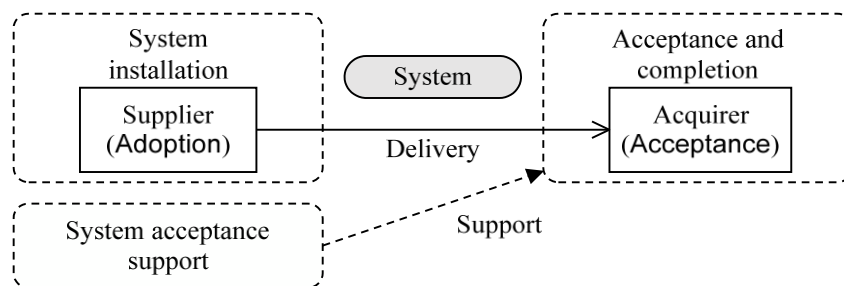


Fig. 4-1 Relationship between system installation and system acceptance support

1-1-1 System Requirements Definition Process

The **system requirements definition** process converts the defined stakeholder requirements to technical requirements (i.e., **system requirements**) that can be used in system design.

[Activities of system requirements definition]

- (i) Definition of system requirements
- (ii) Evaluation and review of system requirements

(1) Definition of system requirements

In definition of system requirements, the specific applications of the system to be developed are analyzed from the stakeholder requirements, and system requirements such as the following are clarified and documented in a **system requirements definition document**.

[System requirements]

- Objectives and scope of computerization (e.g., the targeted work, the targeted departments)
- Functions, capabilities, and life cycle of the system
 - Functions are summarized as **functional requirements** in **system functional specifications**.
 - Capabilities are clarified as **performance requirements** (e.g., response time, throughput).
- Business, organizational, and user requirements
 - User work processing procedures, **input/output information requirements**, **database requirements**, **operation requirements**, and other requirements from the business, organization, and users are clarified. In addition, **test requirements** to verify these requirements are also defined.
- Reliability, safety, security, human-factors engineering (i.e., ergonomics), interface, operations, and maintenance requirements
 - **Security requirements**, **operational requirements** (including education, training, and others), **maintenance requirements** (including fault handling and others) are clarified.
- **System configuration requirements**
 - **Execution environment requirements**, **peripheral interface requirements**, and others are clarified.
- Design constraints and qualification requirements
 - Design constraints are clarified as **design constraint conditions**. System configuration requirements can also be design constraints.
 - **Qualification requirements** are standards for verifying that a system is of usable quality, and are documented in a **specification document of a system qualification test**.
- Development environment
- Quality, expense, and expected effects
 - Quality is clarified as **quality requirements**.
- **Migration requirements** and **validation requirements** for system migration

(2) Evaluation and review of system requirements

Two tasks are executed in evaluation and review of system requirements.

(i) Evaluation of system requirements

Defined system requirements are evaluated, in consideration of the criteria of

traceability to acquisition needs, consistency with acquisition needs, testability, feasibility of system architectural design, and feasibility of operation and maintenance.

(ii) **Implementation of joint review of system requirements**

A joint review by the development department (i.e., supplier) and stakeholders (i.e., acquirer) is conducted, and agreement is obtained concerning system requirements. In order to deepen mutual understanding, prototypes may be created to verify feasibility, or simulations may be conducted.

Review refers to verification and evaluation of the activity status and deliverables of a project by concerned parties. Typical types of review and review methods are as follows:

[Types of review]

- **Joint review**

This is a review aimed at maintaining among stakeholders a shared understanding of progress toward agreed-upon objectives and shared understanding for the purpose of aiding confirmation that product development satisfies stakeholders.

- **Design review**

This is a review aimed at discovering errors in design documents.

- **Code review**

This is a review aimed at discovering errors (i.e., bugs) in source code (i.e., source programs). It can also be expected to improve the readability of programs.

[Review methods]

- **Walk-through**

This is a review implemented by a creator and multiple concerned parties. It is implemented primarily for the purpose of early discovery of errors.

- **Inspection** (software inspection)

This is a review conducted by a **moderator** (i.e., a person in charge of implementation). The moderator must bear responsibility for confirmation and correction of errors.

Typical implementing procedures for reviews are: Development of the review documents; implementation of review (i.e., determination of review methods, determination of review evaluation standards, selection of review participants); and reflection of review results in documents. Points for effectively conducting reviews include advance distribution of materials, conducting reviews in a short time, and not including persons in charge of personnel evaluations among the review participants.

1-1-2 System Architectural Design Process

The **system architectural design** process identifies how system requirements should be allocated to system elements (e.g., hardware, software).

[Activities of system architectural design process]

- (i) Establishment of systems architecture
- (ii) Evaluation and review of systems architecture

(1) Establishment of systems architecture

Three tasks are executed in establishment of systems architecture.

(i) Establishing the architecture at the top level of the system

The system is functionally divided into the items of hardware, software, and manual operation at the top level of the system. At this time, it is necessary to confirm that all system requirements be allocated to any items. Hardware configuration items, software configuration items, and manual operations are clarified from these items. The systems architecture and the system requirements allocated to each item are documented in a **system architectural design document**.

- In the functional decomposition, **user work scope** (i.e., manual operation) and others are considered from the perspective of operational efficiency, workload, and activity costs.
- In hardware architectural design, fault tolerant design (e.g., redundancy) is considered along with the **functional allocation and reliability allocation of servers, and configuration is determined**.
- In software architectural design, **in-house development, use of software packages, middleware to be used, and others are considered, and configuration is determined**.
- In system processing architectural design, centralized processing / distributed processing, web systems, client/server systems, and others are considered, and processing methods are determined.
- In database architectural design, **the type of database to be used is determined** from among RDB (Relational DataBases), NDB (Network DataBases), OODB (Object-Oriented DataBases), XML databases, and other databases.

(ii) Development of user documentation (preliminary version)

A preliminary version of user documentation is developed. User documentation is

documentation delivered to users together with systems. Examples include systems operation manuals and business operation manuals.

(iii) **Definition of test requirements for system integration**

Provisional test requirements and a schedule for system integration are defined and documented. The scope of **system integration test**, test plans, test procedures, and other policies are determined. Then, a **system integration test specification document** is developed. This document includes test requirements to confirm whether the system fulfills functionality.

(2) **Evaluation and review of systems architecture**

Two tasks are executed in evaluation and review of systems architecture.

(i) **Evaluation of systems architecture**

The established systems architecture is evaluated, with consideration given to the criteria of traceability to system requirements, consistency with system requirements, **appropriateness of design standards and methods used, feasibility of software items fulfilling their allocated requirements, and feasibility of operation and maintenance.**

(ii) **Implementation of joint review of system architectural design**

A joint review by the **development department (i.e., supplier) and stakeholders (i.e., acquirer)** is implemented, and agreement is obtained concerning system architectural design.

1-1-3 Implementation Process

The **implementation** process implements the specified system elements (i.e., hardware, software). The software implementation process for implementing software is described in “1-2 Software Implementation Process.” Here, the hardware implementation process for implementing hardware is described.

[Overview of the hardware implementation process]

The purpose of the hardware implementation process is to produce a specified system element implemented as a hardware product or a service.

In the hardware implementation process, specified behavior, interfaces, and implementation constraints are transformed into a system element (i.e., a hardware configuration item) to be implemented as a hardware product or a service.

1-1-4 System Integration Process

The **system integration** process integrates system elements (including hardware configuration items, software configuration items, manual operations, and other systems, as necessary) in order to produce a complete system that satisfies the system design and the stakeholders' expectations described in the system requirements.

[Activities of system integration process]

- (i) System integration
- (ii) Evaluation and review of test readiness and system integration

(1) System integration

Three tasks are executed in system integration.

(i) Development of a system integration plan

Hardware configuration items, software configuration items, manual operations, and any other required systems are integrated, and an integration plan for turning these into system items is developed. In the integration plan, test requirements, procedures, data, responsibilities, plans, and others are to be documented.

(ii) Implementation of system integration test

Software configuration items are integrated, with hardware configuration items, manual operations, and other systems as necessary, into the system. System integration is tested against requirements according to the sequence of development, and the integration and the test results are documented in a **system integration test report**.

- Following the **system integration test specification document** defined in system architectural design, a test environment and test data are prepared and implemented according to the test plan.

(iii) Updating user documentation

User documentation is updated as required, including corrections to operation manuals.

Note: **System integration test** refers to testing that combines deliverables developed separately, in order to validate proper operation (i.e., that interfaces are correct). Integration testing is described in “1-2-5 Software Integration Process.”

(2) Evaluation and review of test readiness and system integration

Three tasks are executed in evaluation and review of test readiness and system integration.

(i) Preparation of system qualification test

A set of tests, test cases (e.g., inputs, outputs, test criteria), and test procedures are developed and documented for conducting system qualification test.

(ii) Evaluation of system integration

The results of system integration are evaluated, with consideration given to the criteria of test coverage of system requirements, appropriateness of test methods and standards used, conformance to expected results, feasibility of system qualification test, and feasibility of operation and maintenance. The system is modified and documentation is updated, as required.

(iii) Implementation of joint review of system integration

A joint review by the development department (i.e., supplier) and stakeholders (i.e., acquirer) is conducted, and agreement is obtained concerning system integration.

Note: Concepts concerning evaluation of testing results are described in “1-2-4 Software Construction Process.” Here, it is sufficient to understand the review as confirmation that all testing results are according to expectations for system requirements.

1-1-5 System Qualification Testing Process

The **system qualification testing** process ensures that the **system is ready for delivery** by testing the compliance of implementation for each system requirement.

[Activities of system qualification testing process]

(i) System qualification testing

(1) System qualification testing

In system qualification testing, seven tasks are performed.

(i) Implementation of system qualification testing

System qualification testing is conducted in accordance with the qualification requirements specified for the system in the system requirements definition. It is ensured that the implementation of each system requirement is tested for compliance and that the system is ready for delivery. The qualification test

results are documented in a **system qualification testing report**.

(ii) **System evaluation**

The results of system qualification testing are evaluated, with consideration given to the criteria of test coverage of system requirements, conformance to expected results, feasibility of operation and maintenance, and appropriateness of the test methods and standards used. If the result of evaluation reveals areas that do not fulfill system requirements, **tuning** is conducted and documentation is updated, as required.

(iii) **Implementation of joint review of system qualification testing**

A joint review by the development department (i.e., supplier) and stakeholders (i.e., acquirer) is implemented, and agreement is obtained concerning system qualification testing.

(iv) **Updating user documentation**

User documentation is updated as required, including corrections to operation manuals. When system operators update user documentation, the necessary information is to be provided to the system operators.

(v) **Support for audits**

Audit refers to independent evaluation to assure that selected deliverables and processes conform to the corresponding requirements, plans, and agreements. In support for audits, the audit activities are supported, and the audit results are documented.

(vi) **Preparation of deliverable systems**

Deliverable systems are updated and prepared for the process of supporting system installation and system acceptance.

(vii) **Preparation of systems to be taken over in operation and maintenance**

From the created deliverables, deliverables to be taken over in operation and maintenance are organized and prepared.

In system qualification testing, whether the system has been achieved (i.e., implemented) according to system requirements is verified through testing listed below.

Name of Test	Content of Testing
Functional test (Functional requirement test)	Verifies whether functions defined in system requirements (functional requirements) are fulfilled, by using data used in business operations.
Non-functional requirement test	Verifies whether non-functional requirements other than functional requirements of the system (i.e., software) are fulfilled.

Performance test	Verifies whether performance requirements (e.g., response time, throughput) defined in system requirements are fulfilled.
Load test (Stress test)	Increases the volume of processed data or the number of programs executed simultaneously, in order to verify the maximum processable data volume or continuous operable time.
Security test	Verifies that security requirements defined in system requirements are fulfilled. In order to intentionally find issues of a security breach, verify strength against unauthorized access. Tests including penetration testing are performed.
Exception test	Verifies that appropriate operation (e.g., error processing) is performed, by using data that would be processed as exceptions in business operations.
Regression test	Verifies whether modifications to the system because of specification changes or others have impacts on non-modified portions of the system. In general, this is conducted in operation/maintenance.

1-1-6 System Installation Process

The **system installation** process installs a system that meets the agreed system requirements in the **production environment** (i.e., the actual operational environment).

[Activities of system installation process]

- (i) System installation

(1) System installation

In system installation, two tasks are performed.

- (i) **Development of a system installation plan**
Support the acquirer in drafting a plan for installation of the system in the production environment, and document an **installation plan** as designated in the contract.
- (ii) **Implementation of system installation**
Assist the acquirer with installation of the system in accordance with the installation plan, and document the installation events and results.

In (i) described above, ahead of the drafting of the installation plan, the following two requirements are considered: **migration requirements** that outline how installation into the production environment and system migration is implemented; and **installation requirements** that outline points of note in data preservation (e.g., backups), impacts on business operations, and others, and how the installation schedule and installation structure is set. On the basis of the results, the acquirer must be supported in drafting a feasible installation plan, according to **installation acceptance/rejection criteria**.

In actual system installation described in (ii), the initialization of the system, software, and database is performed and the execution environment is prepared according to procedures defined in the installation plan, and after that, the system is installed. Ideally speaking, these installation operations are to be performed under the leadership of the user department and system operations department of the acquirer, with the supplier providing user support.

1-1-7 System Acceptance Support Process

The **system acceptance support** process supports the acquirer in confirming whether the system fulfills system requirements.

[Activities of system acceptance support process]

(i) System acceptance support

(1) System acceptance support

In system acceptance support, three tasks are performed.

(i) Support for the acquirer's acceptance review and acceptance testing

The acquirer's acceptance review and acceptance testing are supported. These are conducted in consideration of the results of the previously conducted joint review, software qualification testing, system qualification testing, and the audit. Their results are documented.

(ii) System delivery

The completed system is delivered along the lines of the acquirer's readiness for acceptance based on the contract.

(iii) Support for education and training of the acquirer

Initial and continuing training and support are provided to the acquirer as specified in the contract.

Acceptance testing refers to testing, which is performed by the acquirer according to

acceptance criteria, to verify that the delivered system fulfills system requirements. The acquirer performs acceptance testing and other tests according to acceptance procedures. If the results fulfill **receiving inspection** criteria, accepts the system as a finished product.

In “(ii) System delivery” described above, user documentation is also delivered. The user documentation includes the system operations manual / business operations manual that includes operational rules for the work that the acquirer should perform in operation, and the **user manuals** that are composed of system usage documents / software usage documents and tutorials (i.e., explanations of functions and operational procedures using examples).

1 - 2 Software Implementation Process

In the **software implementation** process, the requirements are defined and software specifications are determined, for each individual software configuration item to be developed, and then the software is developed.

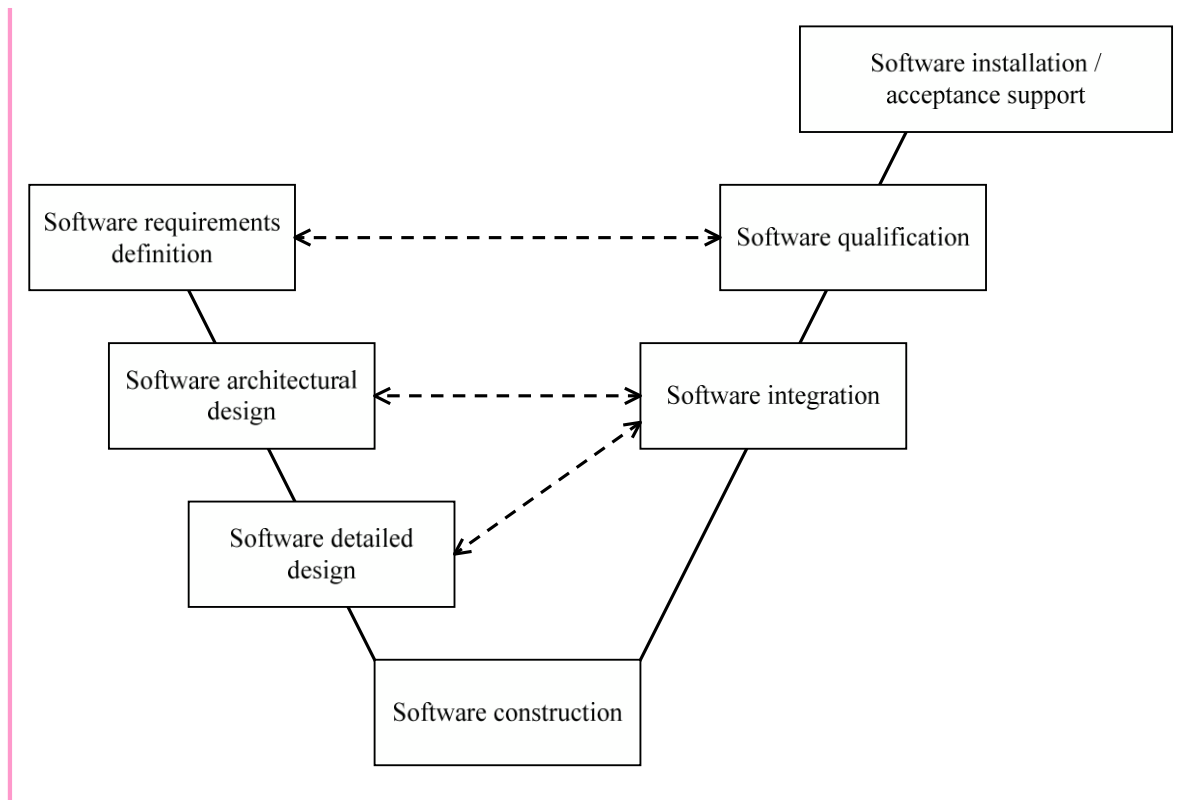
[Objectives of the software implementation process (excerpt)]

The purpose of the software implementation process is to produce a specified system element implemented as a software product or a software service.

The software implementation process transforms specified behavior, interfaces, and implementation constraints are transformed into a system element (i.e., a software configuration item) implemented as a software product or a software service.

[Processes of the software implementation process (excerpt)]

- (i) Software requirements definition process
- (ii) Software architectural design process
- (iii) Software detailed design process
- (iv) Software construction process
- (v) Software integration process
- (vi) Software qualification testing process
- (vii) Software installation process
- (viii) Software acceptance support process



In addition, one of the preparations for starting the software implementation process is preparation of a **development environment**. In preparation of the development environment, the work standards, methods, tools, and programming languages, which are documented and established in the organization, are appropriately selected, and then **tailoring** is performed as necessary, in order to conduct the software implementation process.

1-2-1 Software Requirements Definition Process

In the **software requirements definition** process, the requirements for software configuration items are converted to technical requirements (i.e., **software requirements**) usable in design of the software.

[Activities of software requirements definition process]

- (i) Establishment of software requirements
- (ii) Evaluation and review of software requirements

Note: In the Common Frame 2013 software implementation process, (i) and (ii) are treated as a single activity. In this textbook, they are separated for ease of understanding.

(1) Establishment of software requirements

In the establishment of software requirements, the requirements allocated to software configuration items are analyzed, software requirements are clarified as described below, and these are documented as **software requirements definitions**.

[Software requirements]

- Functional and capability specifications, including performance, physical characteristics, and environmental conditions under which the software item is to perform
- External interfaces of the software items
- Qualification requirements (development of **the software qualification testing specifications**)
- Safety specifications, including those related to methods of operation and maintenance, environmental influences, and personnel injury
- Security specifications, including those related to the leakage of information
- Human-factors engineering specifications, including those related to manual operations, human-equipment interactions, constraints on personnel, and areas needing concentrated human attention, that which sensitive to human errors and training
- Data definition and database requirements
- Installation and acceptance requirements of the delivered software product at the operation and maintenance sites
- User documentation requirements
- User operational and execution requirements
- User maintenance requirements

In the establishment of software requirements, a business model that portrays the business and a logical data model that portrays data to be handled are created. Determination of specifications for functions and capabilities required of the software is performed, along with interface design. At the same time, software qualification requirements, for confirming whether requirements are fulfilled, are defined. With regard to software requirements, it is considered advisable to also determine the order of implementation priority.

It is noted that quality characteristic specifications of the software to be developed are also included in software requirements. The **software quality characteristics** defined in **ISO/IEC 9126 (JIS X 0129-1)** are used as guidelines for specifying the **quality characteristics** of software.

Quality Characteristics	Overview
Functionality	The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions [Sub-characteristics] Suitability, accuracy, interoperability, security
Reliability	The capability of the software product to maintain a specified level of performance when used under specified conditions [Sub-characteristics] Maturity, fault tolerance, recoverability
Usability	The capability of the software product to be understood, learned, used, and attractive to the user, when used under specified conditions [Sub-characteristics] Understandability, learnability, operability, attractiveness
Efficiency	The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions [Sub-characteristics] Time behavior, resource behavior
Maintainability	The capability of the software product to be modified. The capability to adapt the software product to corrections/improvements of functions, changes in environment, and changes in requirements specifications [Sub-characteristics] Analyzability, changeability, stability, testability
Portability	The capability of the software product to be transferred from one environment to another [Sub-characteristics] Adaptability, installability, co-existence, replaceability

In **business operations modeling** to create business models or **data modeling** to create logical data models, business analysis is performed by using methods described below.

- **Hearing**

This is a method of hearing comments from users and others in order to clarify the requested particulars of software or work. Hearings are implemented on the basis of hearing planning that includes the subjects and objectives of the hearings, and the results are documented in hearing minutes.

- **Use case**

This is a method used to define the associations and relationships between systems and actors (e.g., outside persons or machines that boot the system, interact with information).

- **Mock-up and prototype**

This is a method of verifying and evaluating validity of specifications, oversights, feasibility, and others by creating mock-ups or prototypes.

- **Diagramming method**

This is a method of diagramming and verifying business processes and others. Typical diagramming methods include **DFD**, which focuses on the flow of data to represents business processes; **E-R diagrams**, which abstract the information handled in business to represent it as entities and relationships among entities; **UML**, which uses object-oriented concepts in representation; and **decision tables**, which summarize complex conditions and others in table format. (Details of diagramming methods are described in “2-2 Software Design Technique.”)

In addition, **form design** and **slip design**, which define the forms and slips used in business in formats adapted to the system, may be performed in data modeling.

- **Form design**

This designs the items and layout of forms (e.g., reports). In form design, the purpose of forms, the period of use, where forms will be used, and others are considered comprehensively. Then, necessary items and layouts are determined.

- **Slip design**

This designs the items and layout of slips (e.g., sales slips). Since slips are used for data input into the system, and others, items and layouts are to be determined in consideration of ease of use (e.g., ease of understanding input items) for the user.

If necessary, **code design** may be performed at this time. (For example, in slip design, managing customers by customer code and entering the code may be deemed easier than entering a customer's name and address.)

- **Code design**

This selects the target items with a code, and creates a code table for each target item. Since code includes sequence code, block code (or classification code), group classification code, mnemonic code, synthetic code, and many other types, appropriate code is to be selected.

However, data modeling is to be performed for the purpose of clarifying the functions and capabilities of software items, external interfaces, and others. For that reason, form design, slip design, code design, and others do not need to be rigidly performed at this time. Moreover, as database requirements and definition of the data to be clarified on the basis of these results are for the purpose of gaining a conceptual image of data to be used in the system, the file organization method for recording data, normalized database tables, and others do not need to be determined.

(2) Evaluation and review of software requirements

Two tasks are executed in evaluation and review of software requirements.

(i) Evaluation of software requirements

Defined software requirements are evaluated, with consideration given to the criteria of traceability to system requirements and system design, external consistency with system requirements, internal consistency, testability, feasibility of software design, and feasibility of operation and maintenance.

(ii) Implementation of joint review of software requirements

A joint review is conducted, and agreement is obtained concerning software requirements.

1-2-2 Software Architectural Design Process

In the **software architectural design** process, software requirements are implemented, and software design able to verify these is provided.

[Activities of software architectural design process]

(i) Software architectural design

(ii) Evaluation and review of software architectural design

(1) Software architectural design

Five tasks are executed in software architectural design.

(i) Architectural design of software structure and components

Requirements concerning the software configuration items are transformed into an architecture that identifies the **software components** that describe the top-level structure. At this time, it is confirmed that all software requirements are allocated

to a software component and that they are refined to facilitate software detailed design, which is documented in a **software architectural design document**.

(ii) **Architectural design of interfaces**

A top-level design is to be developed and documented for the external interfaces of the software configuration items and for the interfaces between the software components.

(iii) **Top-level design for the database**

A top-level design for the database is to be developed and documented.

(iv) **Development of user documentation (preliminary version)**

A preliminary version of user documentation is to be developed and documented.

(v) **Definition of test requirements for software integration**

Preliminary test requirements and the schedule for software integration are defined and documented. The scope, test plans, test procedures, and other policies of **software integration testing** are to be determined. A **software integration testing specification** is to be developed. It includes test requirements to confirm whether the requirements for the software are fulfilled.

In software architectural design, software is partitioned into functional-level software components (hereinafter called components), and functional specifications of each component, and the processing procedures and relationships among components are made clear.

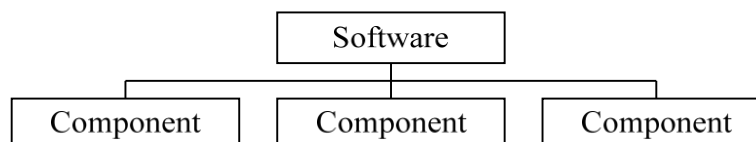


Fig. 4-2 Software and components

In (i) described above, when software is partitioned into components, the concept of **structuring** is often used to design functions in step-by-step detail and create a hierarchical structure. At this time, the criteria used for partitioning components include processing patterns for uniform records, differences in processing timing (i.e., processing cycle), differences in processing efficiency, simultaneously usable resources, characteristics of input devices, and such other factors. (As an example, the update processing for the same file is partitioned as one component.) Moreover, from the perspective of component partitioning, file integration or file partitioning may be used. (For example, when there are different functions for male and female customers, customer files may be partitioned into male customer files and female customer files, and the software is partitioned into components that process each file.) Functional specifications are determined for each partitioned component.

Components become **programs** that provide functions collected into a single function. Since

programs can be reused by other software as components, partitioning is considered on the basis of criteria for ease of understanding, development productivity, operability, processing capability, maintainability, reusability, and such other factor. (This way of thinking is called “**partitioning into components.**”) Furthermore, it is also important to consider the use of existing programs.

In “(ii) Architectural design of interfaces” (**interface design**) described above, top-level interfaces are to be made clear on the basis of the software requirements definition document, in consideration of operability, visibility, software functions, and processing method, according to interface design standards. Since screens, forms/slips, files, and others correspond to external interfaces, GUI-based screen design, form/slip design, and other **input/output design** (i.e., **input/output conceptual design**), along with **logical data design**, are to be performed. In top-level design, logical definitions such as items, layouts and screen transitions are created. Physical definitions that rely on hardware or specific methods are not created. This way of thinking is that same as that in “(iii) Top-level design for the database,” and design that relies on specific DBMS products and others are not performed at the software architectural design stage.

In “(v) Definition of test requirements for software integration,” **checklists** and others are to be used to verify that there are no oversights in test requirements. In addition, test requirements are to be created on the basis of criteria for **black box tests**. These tests verify that the relationships between input and output are correct. (Details of black box tests are described in “1-2-4 Software Construction Process.”)

(2) Evaluation and review of software architectural design

Two tasks are executed in evaluation and review of software architectural design.

(i) **Evaluation of software architectural design**

Software architectural design is to be evaluated, with consideration given to the criteria of traceability to the requirements of the software configuration item, external consistency with the requirements of the software configuration item, internal consistency between the software component, appropriateness of design methods and standards used, feasibility of detailed design, and feasibility of operation and maintenance

(ii) **Implementation of joint review of software architectural design**

A joint review is to be conducted, with agreement obtained concerning software architectural design.

1-2-3 Software Detailed Design Process

The **software detailed design** process provides a design for the software that implements and can be verified against the software requirements and software architecture and is sufficiently detailed to permit coding and testing.

[Activities of software detailed design process]

- (i) Software detailed design
- (ii) Evaluation and review of software detailed design

(1) Software detailed design

Six tasks are executed in software detailed design.

(i) Detailed design of software components

Software components are refined into lower levels containing **software units** (i.e., a single unit, classes, and modules) that can be coded, compiled, and tested. It is confirmed that all software requirements are allocated from the software components to software units. This is documented in a **software detailed design document** (i.e., component detailed design document).

(ii) Detailed design of software interfaces

A detailed design is to be developed and documented for the external interfaces of the software configuration items, the interfaces between the software components, and the interfaces between the software units.

(iii) Detailed design for the databases

A detailed design for the database is to be developed and documented.

(iv) Updating user documentation

User documentation is to be updated as necessary.

(v) Definition of test requirements for software units

Test requirements and the schedule for testing software units are to be defined and documented. The scope, test plans, test procedures, and others of **software unit test** are to be defined, and a **software unit test specifications document** is to be developed. It is desirable that the test requirements include the stress test of the software unit at the limits of its requirements.

(vi) Updating of test requirements for software integration

The test requirements and the schedule for software integration are to be updated.

In software detailed design, software components (i.e., programs) are partitioned into the units of coding called software units.

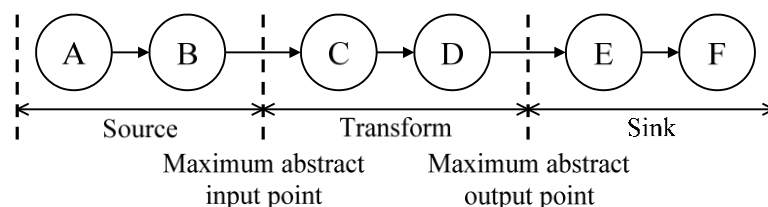
Task (i) described above is also called **module design**, because software component is partitioned into modules, which are the units of development (i.e., coding). In the module partitioning that is conducted during module design, **structured design** using the concept of structuring is used. For the module partitioning that is used in structured design, there are methods that focus on the flow of processing (or data). In general procedures, higher-level module partitioning is first performed by using **STS partitioning**, after which lower-level modules are partitioned by using **transaction partitioning**. After this, the modules are verified, and **common functional partitioning** is performed.

[Module partitioning techniques focused on the flow of processing (or data)]

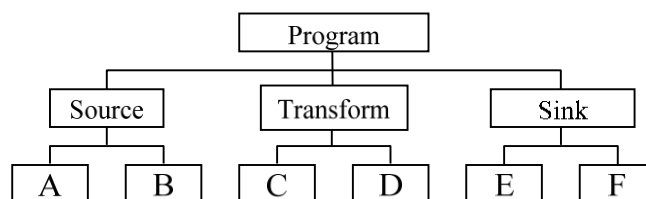
- **STS partitioning**

This is a technique that partitions program flow into three parts: input (i.e., **Source**), processing (i.e., **Transform**), and output (i.e., **Sink**). Each part becomes a module. Partitioning in this case is carried out at the maximum abstract input point where input data has been abstracted, and the maximum abstract output point where the models for output data have been collected.

[Program flow (bubble chart)]

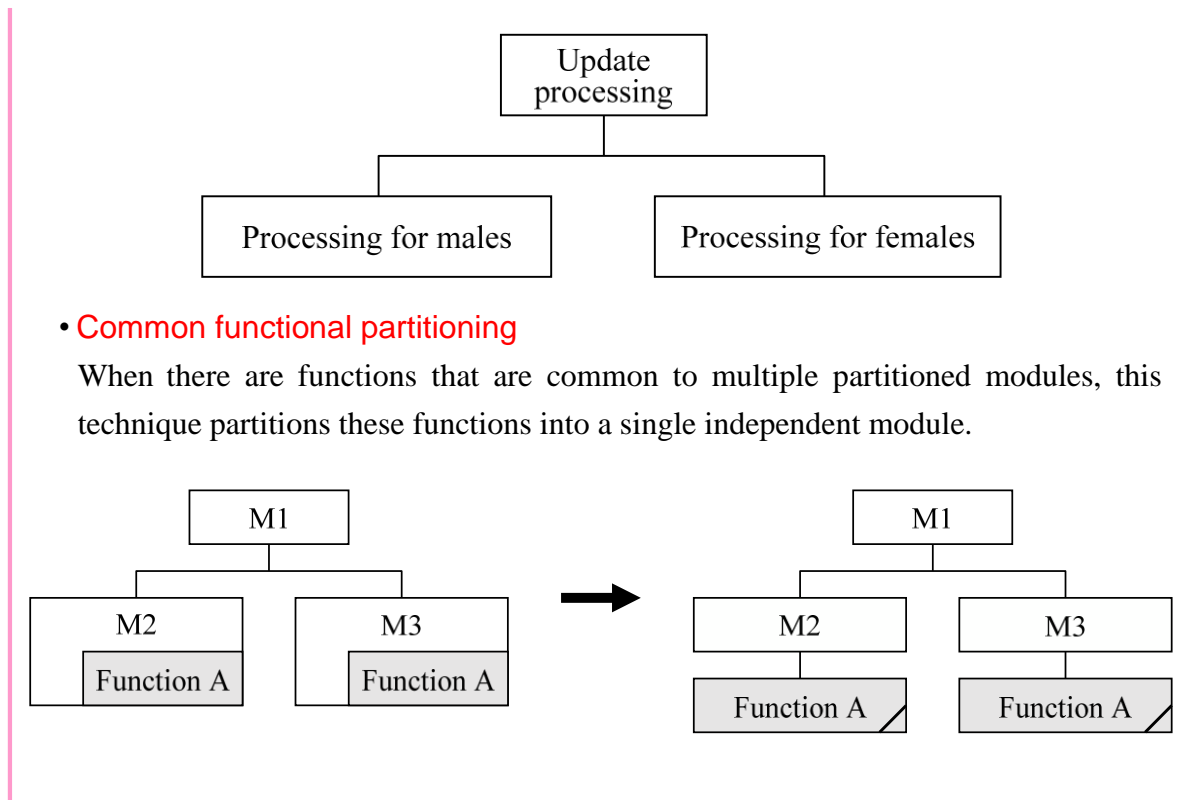


[Module structural diagram] (functional hierarchy diagram)



- **TR partitioning (Transaction partitioning)**

This is a technique that partitions each **process as a module** when processing differs according to type of data. As an example, when processing procedures differ for males and females in file updating, partitioning can be performed as follows:



In addition to methods focused on the flow of processing (or data), there are also methods of module partitioning that focus on data structure.

[Module partitioning techniques focused on data structure]

- **Jackson method**

This is a technique that analyzes input data and output data, and partitions modules according to the output data structure. In the Jackson method, both data structures and program structure are represented as three basic structures (sequence, selection, and iteration).

- **Warnier method**

This is a technique that uses the set theory to analyze input data structures from the perspective of when and how many times to perform processing.

Modules partitioned in this way must be evaluated for independence. Module strength and module coupling are used as metrics for evaluation.

- **Module strength**

This metric represents the associations of functions within the module. A better module has a higher strength. When modules are arranged in order of higher module strength, the result is as follows:

Functional strength	A collection of only functions that execute one certain process
Informational strength	A collection of multiple independent functions that process specific data
Communicational strength	A collection of multiple sequential functions, with internal passing of data
Procedural strength	A collection of multiple sequential functions, without internal passing of data
Temporal strength	A collection of multiple functions executed at a specific timing
Logical strength	A collection of multiple functions selected by parameters
Coincidental strength	A collection of meaningless functions partitioned by size and others

- **Module coupling**

This metric represents associations with other modules. A better module has a weaker coupling. When modules are arranged in order of weaker module coupling, the result is as follows:

Data coupling	Passes only parameters that do not affect control
Stamp coupling	Passes the data structure itself
Control coupling	Passes parameters that affect control
External coupling	Externally declares and shares only necessary data
Common coupling	Externally declares and shares data including unnecessary data
Content coupling	Directly looks up data in other modules

Partitioned modules are to be evaluated according to evaluation criteria, such as scope of control and scope of effect of the module, amount of partitioning of the module (the number of steps), partitioning into components, and reuse. After that, they are to be repartitioned as necessary.

[Guidelines for amount of module partitioning]

- Partition a single component into about 10-200 modules.
- Partition a component so that partitioned modules have a hierarchical depth of up to 4 levels.
- Partition a component so that the number of modules at the same depth (i.e., level) is 10 or less.

Module specification design (or **program design**) is to be performed to clarify the functions of the partitioned modules. At this time, methods for representing module specifications include **flowcharts**, **decision tables**, and **NS (Nassi-Shneiderman)** charts. (Details are described in “2-2-1 Structured Design.”)

In “(ii) Detailed design of software interfaces” (**interface design**), physical design is to be performed, which includes responsiveness, hardware functions, and others along with what have been considered in architectural design. In **input/output detailed design**, as input media, input design to determine input data checking methods (e.g., numeric checks, limit checks, format checks, duplication checks, matching checks, balance checks, logical checks (validity checks)), output objectives, and others are considered, output media (e.g., display screens, printer forms) is determined. In addition, output design is performed to design layouts based on the limitations of paper (specialized paper, general paper), displays, printers, and others. In **physical data design** for files and others, data property analysis, file organization methods (e.g., sequential organization, relative organization, indexed sequential organization, partitioned organization, direct organization), recording media, the layout of records (generally with key items positioned to the front, and reserved items added with expansion taken into account), and others are determined. This way of thinking is that same as that in “(iii) Detailed design for the databases,” and a physical data model is created with an awareness of the DBMS product to be implemented.

With regard to interfaces between software components and interfaces between software units, appropriate interfaces (e.g., parameters and return values) are to be defined on the basis of the relationships between modules (i.e., main routines) that perform calls and modules (i.e., subroutines) that are called. At this time, **call by value** and **call by reference** are to be used appropriately as methods for calling subroutines (i.e., methods for passing parameters).

In “(v) Definition of test requirements for software units,” in the same manner as test requirements for software integration, **checklists**, and others are to be used. In addition, criteria are to be created for test requirements for not only black box tests but also **white box tests** that verify whether the internal structure (i.e., algorithm) of the software is correct. (Details of black box tests and white box tests are described in “1-2-4 Software Construction Process.”)

(2) Evaluation and review of software detailed design

The following tasks are to be executed in evaluation and review of software detailed design.

(i) Evaluation of software detailed design

Software detailed design is to be evaluated, with consideration given to the criteria of traceability to the requirements of the software configuration item, external consistency with architectural design for the software configuration item, internal consistency between the software components and software units, appropriateness of design methods and standards used, feasibility of testing, and feasibility of operation and maintenance.

(ii) Implementation of joint review in software detailed design

A joint review is to be conducted, with agreement obtained concerning software architectural design.

1-2-4 Software Construction Process

The **software construction** process produces executable software units that properly reflect the software design. (In general, this is called **programming**.)

[Activities of software construction process]

(i) Software construction

(ii) Evaluation of software code and test results

(1) Software construction

Five tasks are executed in software construction.

(i) Development of software units and databases

Each software unit and database are to be developed and documented.

(ii) Development of test procedures and test data

Test procedures and test data for testing each software unit and database are to be developed and documented.

(iii) Implementation of software unit and database testing

Each software unit and database is to be tested to confirm that it satisfies software requirements, and the results are to be documented in a **software unit test report document**.

(iv) Updating user documentation

User documentation is to be updated as necessary.

(v) **Updating of test requirements for software integration**

The test requirements and the schedule for software integration are to be updated.

In “(i) Development of software units and databases,” programming is performed according to the specifications for **coding conventions** and **programming languages**, on the basis of the software detailed design. (The work of creating (or writing) a program on the basis of **algorithms** (i.e., procedures) defined in the software detailed design and others is also called **coding**.)

Coding conventions are rules for performing coding. If notational conventions for coding are not set, programs will be inconsistent and will have poor readability, which will reduce maintainability. For that reason, coding conventions (i.e., programming style) should be set, notational conventions should be unified, and program intelligibility (i.e., ease of understanding), efficiency (i.e., ease of creation), maintainability (i.e., ease of change), functionality, usability, and such other factor should be improved. In general coding conventions, indentation, depth of nesting, naming conventions (e.g., how variables are named), prohibited instructions, and others are to be set, in consideration of the standards (i.e., paradigm) of the programming languages used.

In programming, it is also necessary to determine the structure of the data (**data structure**) to be processed. Since appropriate data structures differ by programming language, the used data structures must be decided on the basis of the data processing to be achieved, the ease of implementation, and so on.

Basic data structure		
	Simple-type	Integer type, real type, character type, Boolean type, enumerated type, pointer type
	Structure type	Array, record type
	Abstract data type	Object
Problem-oriented data structure		List, stack, queue, tree, hash

One concept (i.e., technique) for creating easy-to-understand programs with high maintainability is **structured programming**. In structured programming, programs are created with processing procedures considered according to **structure theorem**, which can represent a program with one entrance and one exit as a combination of three basic structural units (i.e., sequence, selection, and iteration).

Furthermore, in order to perform coding efficiently, coding support tools described below

should also be considered.

- **Code auditor**

This is a tool to verify whether a program follows coding conventions.

- **Source code editor**

This is dedicated software for creating programs (i.e., the source code). It offers functions, such as **code completion** which displays a menu when a portion of a command or a variable name is entered, so that an input item can be selected from the input candidates in the menu, and **syntax highlighting** which displays an instruction in a code in color according to the type or classification of the instruction.

In “(ii) Development of test procedures and test data,” the test system (e.g., the scope and schedule of the test, the test implementer), the test tools to be used, and others are considered by using appropriate test methodology according to the software unit test specifications document, and **test plans** are drafted. After this, creation of test data, preparation of a test environment, and other test readiness are performed in order to execute test plans.

The objective of all testing, including the software unit test performed in (iii), is the discovery of errors (i.e., bugs). Errors (i.e., bugs) are faults or defects in software, or portions of a program that do not fulfill specifications or requirements defined in design documents and specifications documents. The following is a list of testing objectives to be implemented in the software implementation process.

Type of test	Objective of test
Software unit test	This verifies whether software unit functions and others fulfill the specifications of the software detailed design document .
Software integration test	This verifies whether the software unit’s combined portion (e.g., interface) fulfills the specifications of the software architectural design document (and the software detailed design document).
Software qualification test	This verifies whether the software fulfills the requirements of the software requirements definition document .

In order to achieve such objectives, appropriate and efficient testing methods (i.e., techniques) must be used in order to reliably find errors ((i.e., bugs).

Testing methods include **white box tests** and **black box tests**.

- **White box test**

This is a test implemented with a focus on the internal specifications (e.g., algorithms, logic paths) of software units (i.e., modules). It is generally used only in the software unit test that is performed by developers.

- **Statement coverage**

This designs test cases that execute every statement in a unit (i.e., module) at least once.

- **Decision condition coverage (branch coverage)**

This designs test cases that execute TRUE/FALSE at least once for every “decision condition” (i.e., branch) in a unit (i.e., module).

- **Condition coverage**

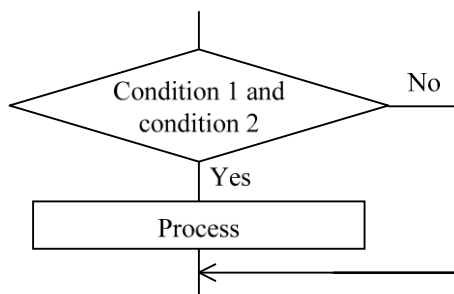
This designs test cases that execute TRUE/FALSE at least once for every “condition” used as a decision condition.

- **Decision condition / condition coverage**

This designs test cases that fulfill both decision condition coverage (i.e., branch coverage) and condition coverage.

- **Multiple-condition coverage**

This designs test cases that include every combination of TRUE/FALSE for every condition.



(Truth value chart)

	Condition 1	Condition 2	Decision condition
I	TRUE	TRUE	TRUE
II	TRUE	FALSE	FALSE
III	FALSE	TRUE	FALSE
IV	FALSE	FALSE	FALSE

Coverage criteria	Test cases that fulfill coverage criteria
Statement coverage	(I)
Decision condition coverage (branch coverage)	(I and II) or (I and III) or (I and IV)
Condition coverage	(I and IV) or (II and III)
Decision condition /condition coverage	(I and II and III) Note: Avoid (I and IV).
Multiple-condition coverage	(I and II and III and IV)

Note: Verification of all test cases may be difficult when processing is complex. In such a case, **coverage rate** (i.e., **test coverage**), a metric that represents coverage rate of test cases (or of paths), is used. The idea is to set a test coverage target in consideration of the balance between productivity and reliability from perspectives of cost, delivery, and others; in other words, the test is completed when this target is cleared.

- **Black box test**

This is a test performed with a focus on the external specifications (i.e., functional specifications and input/output (interface) specifications) of software units (i.e., modules). It is used in all software testing, including software unit test.

- **Equivalence partitioning / boundary value analysis**

This is a method that partitions input data into several groups (i.e., classes), and test data is selected from each group. In general, input data is partitioned into a valid equivalence class (i.e., the group processed normally) and an invalid equivalence class (i.e., the group handled as an error), and then the valid equivalence class is further staged if necessary. From among these partitioned groups, representative values are used as test cases in the case of equivalence partitioning, and values at both ends of each class are used in the case of boundary value analysis.

Example: For a unit processed normally when input data is between 10 and 30

...	7	8	9	10	11	...	29	30	31	32	33	...
Invalid equivalence class				Valid equivalence class					Invalid equivalence class			

Equivalence partitioning: {7, 21, 39} ... One representative value from each class

Boundary value analysis: {9, 10, 30, 31} ... All boundary values from each class

- **Cause-effect graph method**

This method represents the cause-effect relationship of input (i.e., cause) and output (i.e., effect) as a graph (i.e., cause-effect graph) when class sorting of input data is difficult. It is a method that then creates a decision table based on the graph to sort out test data.

In addition to white box tests and black box tests, testing methods and test implementation methods include the following.

- **Error embedding method**

This is a method that embeds the known errors in software, and the number of inherent potential errors is estimated from the ratio of the number of discovered known errors to the number of inherent errors (i.e., number of errors not already known).

- **Experimental design**

This is a method that selects test data efficiently through statistical analysis, when tests that use large volumes of data are required.

- **Metrics measurement**

This is a method that collects and analyzes metrics for the size, complexity, and others of software on the basis of development information and others. Then, the possibility that errors or defects are present is investigated.

When test data is created under these methods, tools such as **test data generators** are used. The implementation of tests is performed according to drafted test plans. At this time, when a module (i.e., software unit) that is the target of testing is called by a higher-level module, a dummy module called a **driver** is used in place of the calling module (i.e., main routine). Conversely, when a module that is the target of testing calls a lower-level module, a dummy module called a **stub** is used in place of the called module (i.e., subroutine).

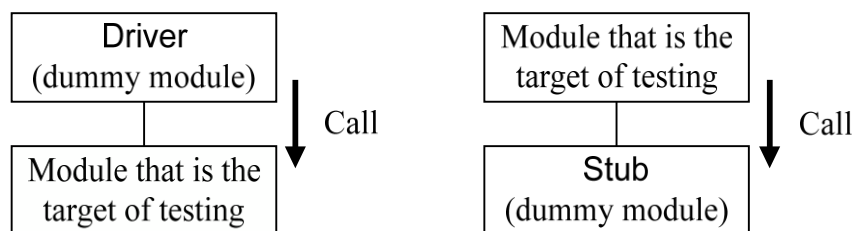


Fig. 4-3 Drivers and stubs

Any errors (i.e., bugs) detected by testing are to be eliminated or corrected. This series of tasks, from detection of such errors (i.e., bugs) to their elimination or correction, is called **debugging**. For efficiency, a debugging environment that offers tools (i.e., **debuggers**) to support debugging and tools (i.e., **test support tools**) to support testing is used.

[Examples of test support tools]

- **Program static analysis tools**

These are tools that support the analysis of static testing (i.e., testing that does not execute the program), such as **desktop debugging** for confirming on the desktop that values have not been substituted in variables that are no longer used.

- **Program dynamic analysis tools**

These are tools that support analysis using dynamic tests (i.e., tests that execute the program), such as **assertion checking**, which embed a logical expression that describes the conditions or the relationships among variables at a specific point in time and verify the validity of the program.

(2) Evaluation of software code and test results

The following tasks are to be executed in evaluation of software code and test results.

(i) **Evaluation of software code and test results**

Software code and test results are to be evaluated, with consideration given to the criteria of traceability to the requirements and design of the software configuration item, external consistency with the requirements and design of the software configuration item, internal consistency between requirements for software units, test coverage of software units, appropriateness of coding methods and standards used, feasibility of software integration and testing, and feasibility of operation and maintenance.

In “evaluation of software code,” **code review** is performed. In code review, whether coding conventions are maintained and are based upon the software detailed design document, whether the efficiency and maintainability of code are appropriate, and others are verified along with the results of metrics measurement.

[**Types of code review**]

- **Peer code review**

This is a review performed by members of the development team, colleagues, and other people primarily for the purpose of eliminating defects in deliverables (i.e., code).

- **Code inspection**

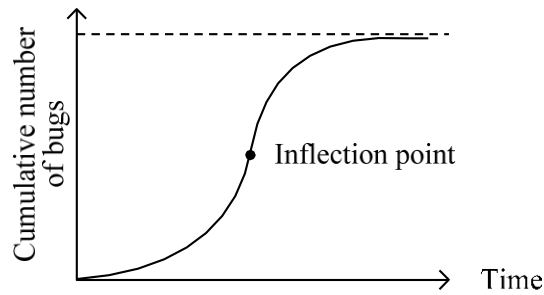
This is a formal review performed by determining a **moderator** (i.e., the person responsible for implementation), primarily for the purpose of improving the quality of deliverables (i.e., code).

In “evaluation of test result,” test results are to be comprehensively evaluated by using methods as described below. Through the evaluation, errors (i.e., bugs) in the program are to be corrected as necessary.

- **Bug curve (reliability growth curve)**

This is a graph that plots test results with the cumulative number of bugs or errors on the vertical axis and the elapsed time on the horizontal axis. It is used to quantitatively predict reliability. It can also be used for test progress management by additionally entering the number of finished test items (or the number of unfinished test items), the number of unresolved bugs, and others.

When the bug curve has an inflection point and eventually converges to a **logistic curve**, the program can be assessed as having high reliability (i.e., high quality).



For using in progress management, the progress status and quality of the program are to be comprehensively assessed, including not only the cumulative number of bugs but also the number of finished test items, number of unresolved bugs, and others. As an example, even if the cumulative number of bugs exceeds expectations, the progress of testing can be considered fast if the number of finished test items also exceeds expectations. However, if the number of finished test items is below expectations, then the quality of the program should be assessed as low.

- **Bug control chart**

This is used to understand test status, product quality, and others on the basis of bug curves. The actual bug curve is recorded onto a graph that already depicts the lower control limit line and the upper control limit line, which are predicted from past performance and such other factor, and is compared with these lines.

1-2-5 Software Integration Process

In the **software integration** process, software units and configuration components are integrated to create software configuration items that have consistency with the software design and that fulfill the software requirements (i.e., functional requirements and non-functional requirements) within the operating environment (i.e., the production environment or equivalent environment).

[Activities of software integration process]

- (i) Software integration
- (ii) Evaluation and review of test readiness and software integration

(1) Software integration

Three tasks are executed in software integration.

(i) **Development of a software integration plan**

An **integration plan** to integrate the software units and the software components into the software configuration item is to be developed. In the integration plan, test requirements, procedures, data, responsibilities, schedules, and others are to be documented.

(ii) **Implementation of software integration test**

Software units and software components are to be integrated in accordance with the integration plan. At this time, performing testing for each created collective will ensure that each aggregation satisfies the software requirements and that the software item is integrated at the conclusion of the integration activity. The results of integration and testing are to be documented as a **software integration test report**.

- A test environment and test data are to be prepared and implemented under test plans in accordance with a **software integration test specifications document** defined in software architectural design and updated in later processes.

(iii) **Updating user documentation**

User documentation is updated as necessary, with the inclusion of corrections of operation manuals.

In software integration, the software units developed in the software construction process, the software components (i.e., programs) and software units to be reused, and other configuration components are assembled, and software configuration items are created.

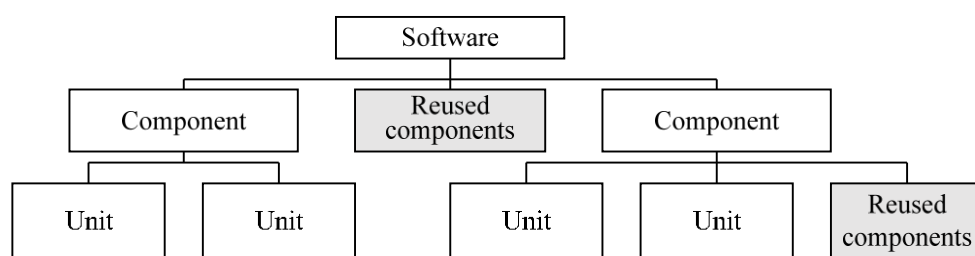


Fig. 4-4 Depiction of software integration

The **software integration test** is a test that assembles partitioned and developed deliverables to verify correct operation (i.e., correct interfaces).

Typical methods of an integration test include **incremental testing** and **non-incremental testing**.

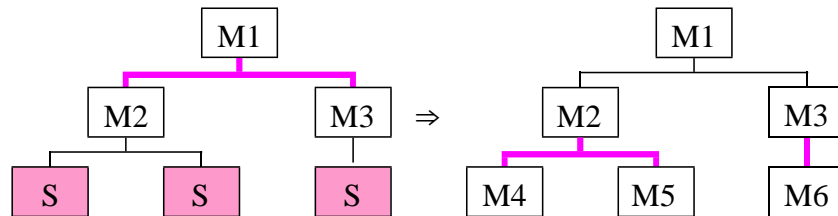
- **Incremental testing**

This is a test in which tested module groups are incrementally integrated with new

modules. It is a test suited to large-scale systems, and requires dummy modules for integration testing.

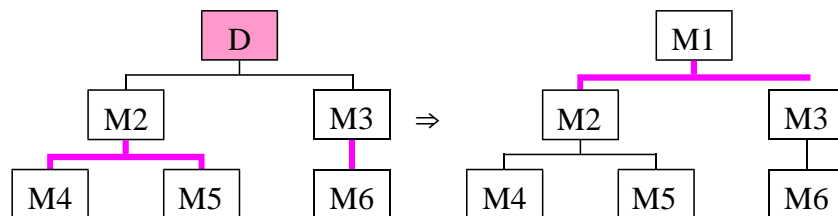
- **Top-down testing**

This is a test that performs integration from top-level module to lower-level modules. It requires **stubs** that work as lower-level modules.



- **Bottom-up testing**

This is a test in which integration is performed from bottom-level modules to higher-level modules. It requires **drivers** that work as higher-level modules.



- **Sandwich testing**

In sandwich testing, a sandwich line is determined, after which top-down testing is performed on modules above the line and bottom-up testing is conducted on modules below the line. At the end, these are integrated at the sandwich line. This allows testing in a short period of time, but both stubs and drivers are necessary for use as dummy modules.

- **Non-incremental testing (Big bang testing)**

In this test, all modules are integrated and tested all together after each unit test is completed. It is suited to small-scale systems, and does not require dummy modules for integration testing. However, it has demerits, such as disallowing implementation in parallel with development activities, and difficulty in identifying the locations (i.e., interfaces) of the causes of detected errors.

In “(i) Development of a software integration plan,” the integration sequence of units and components, and others, are created in the form of an **integration plan**, in consideration of the software integration test methods. The integration plan is to also include **test planning**, covering the testing system (scope of testing, schedule, and test implementer), the test tools to

be used, and others, in accordance with the software integration test specifications document. In (ii), test preparation (e.g., preparation of test data and test environment) is performed in order to execute the integration plan.

(2) Evaluation and review of test readiness and software integration

Three tasks are executed in evaluation and review of test readiness and software integration.

(i) Preparation for software qualification testing

A set of tests, test cases (e.g., inputs, outputs, test criteria), and test procedures are developed and documented for conducting the software qualification testing.

(ii) Evaluation of software integration

The results of software integration are to be evaluated, in consideration of standards including traceability to the system requirements (i.e., software requirements), external consistency with system requirements (i.e., software requirements), internal consistency, test coverage of the requirements of the software items, appropriateness of the test methods and test standards used, conformance to expected results, feasibility of the software qualification testing, and feasibility of operation and maintenance. The software is then to be modified and documentation updated as required.

(iii) Implementation of joint review of software integration

A joint review is conducted, and agreement is obtained concerning software integration.

1-2-6 Software Qualification Testing Process

In the **software qualification testing** process, whether the integrated software configuration items (i.e., software products) fulfill the defined requirements is to be confirmed.

[Activities of software qualification testing process]

(i) Software qualification testing

(1) Software qualification testing

Six tasks are performed in the software qualification testing.

(i) Implementation of software qualification testing

Software qualification testing is conducted in accordance with the qualification

requirements specified for the software in the software requirements definition. It is ensured that the implementation of each software requirement is tested for compliance. The qualification testing results are documented in a **software qualification testing report**.

- Whether the software products have been implemented in line with the software requirements is to be confirmed through functional testing, non-functional testing, performance testing, load testing, security testing, exception testing, regression testing, and other testing.

(ii) **Updating user documentation**

User documentation is to be updated as necessary.

(iii) **Evaluation of software qualification testing**

The results of the software qualification testing are to be evaluated, in consideration of test coverage of software requirements, conformance to expected results, feasibility of system integration and testing, and feasibility of operation and maintenance.

(iv) **Implementation of joint review of software qualification testing**

A joint review is to be conducted, with agreement obtained concerning qualification test.

(v) **Support for audits**

Audits are to be supported, and the audit results are documented.

(vi) **Preparation of deliverable software products**

The deliverable software product is to be updated and prepared for system integration, system qualification testing, software installation, and software acceptance support.

1-2-7 Software Installation / Acceptance Support Process

The **software installation** process and **software acceptance support** process are processes used when the acquirer directly installs the software developed in the software implementation process, in the form of a software product. Thus, these processes are not necessary when software configuration items are developed as system elements. The activities and tasks of the software installation process and software acceptance support process are composed of nearly the same concepts as in the system installation process and the system acceptance support process.

In the software installation process, software products fulfilling the agreed-upon software requirements are installed in the production environment (i.e., the actual operational environment).

[Activities/tasks of software installation process]

- (i) Software installation
 - Development of the software installation plan
 - Implementation of software installation

In the software acceptance support process, the acquirer is supported in being certain that the software product fulfills software requirements.

[Activities/tasks of software acceptance support process]

- (i) Software acceptance support
 - Support for the acquirer's acceptance review and acceptance testing
 - Delivery of software products
 - Education/training and support for the acquirer

1 - 3 Maintenance and Disposal Process

1-3-1 Maintenance Process

The **maintenance** process provides high-cost-performance support for delivered systems and software products. In maintenance, modifications and improvements are carried out while the safety of systems (or software products) is maintained, in response to the occurrence of problems, requests for functional expansion, improvements, and others.

In implementing maintenance, **maintenance requirements** and **maintenance frameworks** are to be defined, and **maintenance agreements** are to be concluded, in consideration of both the requests on the part of the party receiving maintenance and the costs and feasibility on the part of the party providing the maintenance.

[Activities of maintenance process]

- (i) Preparations for initiating the maintenance process
- (ii) Assessment of issues, and analysis of changes
- (iii) Implementation of changes
- (iv) Maintenance review and/or acceptance
- (v) Support for operational testing and migration

(1) Preparations for initiating the maintenance process

Five tasks are executed in preparations for initiating the maintenance process.

(i) **Transfer of deliverables necessary for maintenance**

Deliverables necessary for maintenance are to be transferred, including code and databases, design documents, qualification test plans / test results, the acquirer's acceptance confirmation document, documents concerning current business operations and system operations, and the system migration procedures / migration framework for the new operational environment.

(ii) **Development of a plan and procedures**

A maintenance plan and procedures are to be developed, documented, and executed.

(iii) **Establishment of problem management procedures**

Procedures are to be established for receiving, recording, and tracking problem reports and modification requests from users, and for providing feedback to users.

(iv) **Management of change activities**

Configuration management (i.e., defining configuration items and managing information) is to be implemented for managing modifications to the systems (or software products).

(v) **Development of documents for maintenance**

Transferred deliverables are to be identified, and documents for maintenance are to be created as necessary.

In “(ii) Development of a plan and procedures,” a **maintenance plan** is established in consideration of how maintenance is to be implemented, the type and form of maintenance, implementation methods, points for caution, and other factors.

Typical types and forms of maintenance include the following.

• **Preventive maintenance**

This is implemented with a maintenance plan prepared in advance, to prevent the occurrence of faults and others. Since it allows planned securing of maintenance staff members, it enables efficient maintenance.

• **Daily maintenance**

This is maintenance in which the equipment that composes systems is monitored on a daily basis.

• **Scheduled maintenance**

This is performed periodically.

• **Corrective maintenance**

This is implemented when faults or abnormal situations occur. Unlike preventive maintenance, it is not scheduled in advance. Because of this, it is difficult to secure maintenance staff, and therefore, it is recommended to put emergency maintenance

staff in regular posts.

- **Non-scheduled maintenance**

This is performed when conditions appear that differ from normal operation.

- **Emergency maintenance**

This is performed for the purpose of restoration after the occurrence of a fault.

- **Software maintenance**

This is maintenance for the purpose of improving the software configuration items that compose software products or systems (e.g., changes and improvements in programs, changes in documents). By contrast, maintenance of the whole system, including software maintenance and **hardware maintenance** for the purpose of improving the hardware configuration item (e.g., increasing the hard disk information storage capacity), is called **system maintenance**.

- **Corrective maintenance**

This is maintenance (e.g., changes in programs) that is implemented to resolve problems (i.e., items that do not fulfill system requirements) that are discovered after system delivery.

- **Adaptive maintenance**

This is maintenance (e.g., improvement of programs) that is implemented to resolve issues that occur because of the changes in the environment after system delivery.

- **Perfective maintenance**

This is maintenance (e.g., improvement of programs) that is performed after system delivery to improve performance and others of the delivered systems (or software products).

- **On-site maintenance**

This is implemented through on-site visits by maintenance staff members. On the other hand, **remote maintenance** is performed from a remote location without on-site visits by using **WOL (Wake On LAN)** and others to boot computers over a LAN.

Since continually securing dedicated maintenance staff members is costly, external contracting (i.e., **outsourcing**) of maintenance work and others are also to be considered.

(2) Assessment of issues and analysis of modifications

Five tasks are executed in assessment of issues and analysis of modifications.

- (i) **Analysis of problem reports and modification requests**

The effects of problems reports and modification requests on the system and on

related systems are analyzed, in terms of type (e.g., correction, improvements, prevention), scope (e.g., size of modification, cost involved, period to modify), and severity (e.g., performance, safety).

(ii) **Reproduction or verification of problems**

Problems are to be confirmed through reproduction or verification.

(iii) **Preparation of options for implementation of modifications**

On the basis of the analysis, options are to be developed for the implementation of modification.

(iv) **Documentation**

Problems, modification requests, analysis results, and options for implementation are to be documented.

(v) **Approval of modification proposals**

Approval is to be obtained for the selected modification options, as specified in the contract.

(3) Implementation of modifications

Two tasks are executed in implementation of modifications.

(i) **Analysis and determination of modification portions**

Analysis of modifications is performed, and the system (or software) and related documents to be modified, and the content of modifications (e.g., functional additions, performance improvements, correction of problems), are determined and documented.

(ii) **Implementation of modifications**

Modifications are implemented. Modification tasks are implemented according to the system development process and the software implementation process. Depending on the object and content of modifications, the planning process and the requirements definition process may also be used.

- The requirements and evaluation criteria of **maintenance testing** to verify modifications are to be determined and documented. In addition to testing to confirm whether the modified portions accurately fulfill modification requirements, maintenance testing includes **regression testing** to verify that there are no effects on unmodified portions.
- **Reverse engineering** to analyze and derive the specifications of in-service systems (or software products) is to be performed as required.

When modifications consist of the correction of problems, implementation of preventive

measures against recurrence of problems is also to be considered. In order to prevent recurrence of problems, the root causes of the problems are to be identified through specific cause analysis, and others, and the possibility of the occurrence of similar problems is to be considered. Depending on the results of consideration, improvement of the systems (or software products), revision of manuals, and others are to be implemented. When prevention of the recurrence of problems is deemed difficult, life cycle evaluation must be performed, with disposal also taken into consideration.

(4) Maintenance review and/or acceptance

Three tasks are executed in maintenance review and/or acceptance.

(i) **Review of the modified system**

A review is to be conducted in order to confirm the integrity of the modified system.

(ii) **Approval of completion**

Approval is to be obtained regarding the satisfactory completion of modifications.

(iii) **Updating of documents for maintenance**

Documentation for maintenance is to be updated as required.

(5) Support for operational testing and migration

Two tasks are executed in support for operational testing and migration.

(i) **Support for implementation of operational testing**

Prior to the release of a system (or software product), the operator is to take the lead in supporting the implementation of operational testing to verify whether there are any problems in the production environment.

(ii) **Support for migration**

The migration of a system (or software product) to the production environment is to be supported. The general migration procedures for the system (or software product) are as follows:

- 1) Documentation and verification of the migration plan
- 2) Notification of migration plans and others to all concerned parties
- 3) Parallel operation of the old and new environments, and disposal of the old system
- 4) Notification of migration to all concerned parties
- 5) Migration evaluation (i.e., verification and review of migration results)

- 6) Maintenance of old environment-related data, and assurance of security

1-3-2 Disposal Process

The **disposal** process ends the existence of system or software products.

[Disposal process tasks]

- (i) Drafting of a disposal plan
- (ii) Execution of the disposal plan
- (iii) Notification of the disposal plan and others to users
- (iv) Parallel operation of the new and old system, and education/training of users
- (v) Notification of disposal to all concerned parties
- (vi) Maintenance of old disposal-related data, and assurance of security / assurance of accessibility

2 Software Development Technology

The software implementation process defines the activities to be provided to the acquirer and provider, as a “shared yardstick” for software development. What is actually used to perform these activities is software development technology.

2 - 1 Software Development Method

2-1-1 Software Development Model

A **software development model** is a standard model that represents the procedures for developing software. This is also called a system development model for system development.

[Merits of software development models]

- Since development activities are routine, even developers with little experience can participate in work.
- Since deliverables are easily routinized, maintenance work can be mitigated.

(1) Waterfall model

The **waterfall model** is a technique in which large-scale and complex development processes are partitioned into multiple phases, and development is conducted for each phase. Development proceeds in the technique from upstream processes to downstream processes, without redoing the previous phases, as indicated by the name “waterfall.”

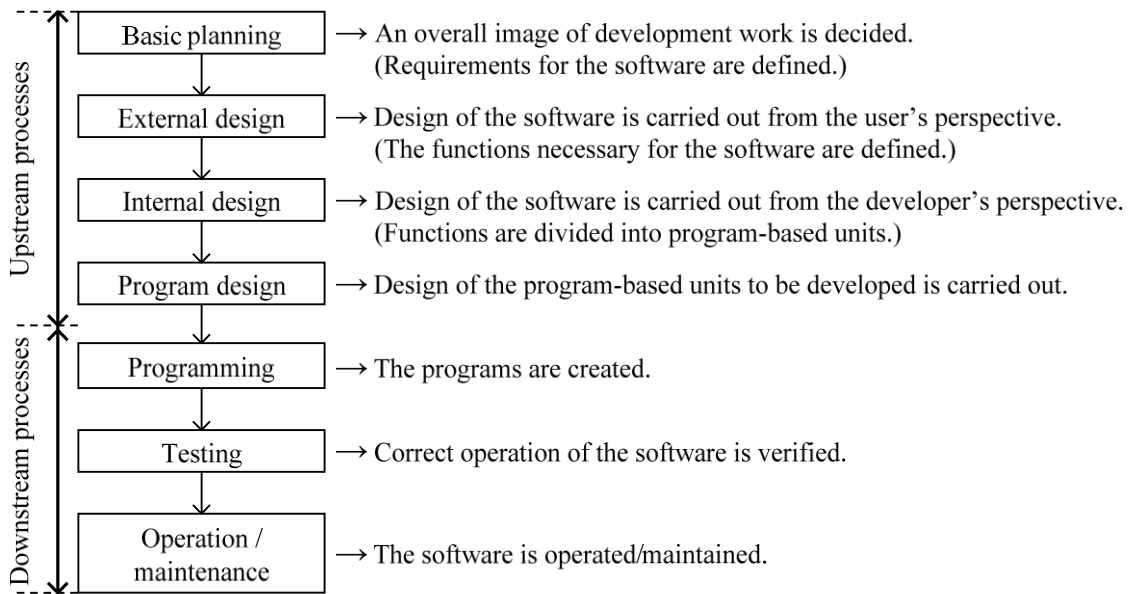


Fig. 4-5 Waterfall model

The main activities to be implemented in the waterfall model software process, and overall corresponding SLCP, are as follows:

Development process	Main activities	Corresponding SLCP
Basic planning	Computerization planning, requirements definition	Software requirements definition process
External design	Definition of subsystems, input/output outline design, code design, logical data design	Software architectural design process
Internal design	Functional partitioning /refinement, input/output detailed design, physical data design	
Program design	Module partitioning	
Programming	Coding	Software construction process

In development under the waterfall model, a **top-down approach** is used so that development can proceed from the overall image of the software to increasingly detailed design. Conversely, in testing processes for developed software, a **bottom-up approach** is used so that development can proceed from the detailed level to the software overall. Thus, illustrating the relationship between design phases and tests reveals a V shape as shown below.

(This is known as a **V-shaped model**.)

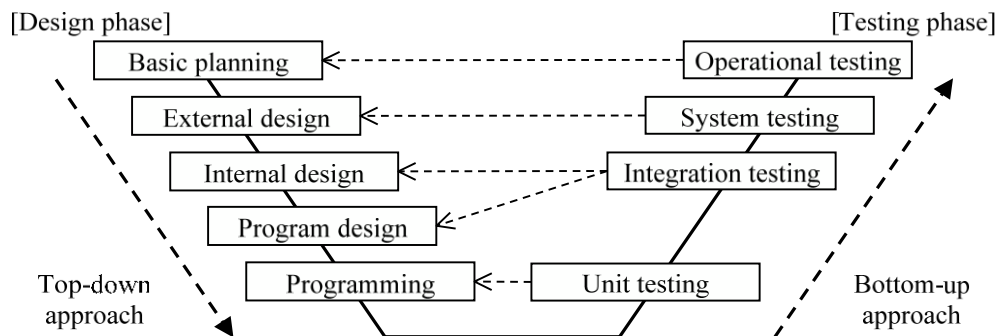


Fig. 4-6 Overall image of the waterfall model (V-shaped model)

Name of test	Content of testing	Corresponding SLCP
Unit test	Confirm that each individual module operates correctly.	Software unit test
Integration test	Modules are combined to confirm correct operation.	Software integration tests
System test	For the software overall, whether required functions are fulfilled and whether there are problems in operability or performance are to be confirmed.	Software qualification test
Operational test (Acceptance test, approval test)	The user department operates the software under conditions of actual operation, to confirm whether it fulfills requirements.	Acceptance test or operational test

(2) Prototype model (prototyping)

The **prototype model** is a development technique that creates a provisional prototype in a short time and is tested and evaluated by users, with specifications finalized while changes are repeated.

Since it enables users to confirm at an early stage that user requirements are met, it is able to reduce reworking. However, as adjustments to the schedule are difficult and prototypes must be created even if incomplete, it is suited to development of small-scale systems.

In the end, it follows two methods: disposing the prototype after confirmation of specifications and then creating a new system, or improving the created prototype (through functional addition) to turn it into the production system.

(3) Spiral model

The **spiral model** is a development technique that divides a large-scale system into independent partial units, and processes including design/development/testing are repeated for each part to achieve a high degree of completion of a system. This technique combines the waterfall model and prototyping, and is suited to development of large-scale systems in which system development personnel are limited.

(4) RAD (Rapid Application Development)

RAD refers to rapid application development that is conducted by a small group of people through the use of development support tools. Users can be brought in to participate in the development work at an early phase. The technique was originally aimed at achieving development by a small number of people and facilitating communication.

(5) Software product line

Software product line is a method that analyzes the group of software to be developed and decomposes it into small parts that can and cannot be shared, which are then developed. By developing the sharable parts as core assets, software development can be performed efficiently.

(6) Iterative model

The **iterative model** is a technique in which software development processes (i.e., design/development/testing) are repeated many times. (The spiral model is a type of iterative model.)

- **Incremental model**

This is a method that partitions software into multiple independent functions, which are developed and released incrementally in units of functions.

- **Evolution model**

This is a technique that develops software of limited functional scope, and adds improvements repeatedly.

2-1-2 Agile

Agile is a general name for techniques that develop high-quality software with rapid and appropriate action. The Manifesto for Agile Software Development released in 2001 declares

the values of agile software development to be as follows:

[The Values of Agile Software Development]

- Individuals and interactions are valued over processes and tools
- Working software is valued over comprehensive documentation
- Customer collaboration is valued over contract negotiation
- Responding to change is valued over following a plan

Note: With recognition of value in the compared items, the Manifesto declares the stated items to be of higher value.

Rather than spending time on design documents under the conventional idea that creating quality design documents equates to creating quality products, agile software development places importance on promoting smooth communication to develop software that actually works.

[Agile-related methods/technologies]

- **XP (eXtreme Programming)**

This is a typical agile software development technique that simplifies the design phase and enhances programming and testing. It is a relatively new technique that emphasizes communication, and places importance on constant feedback, changes, and redesign rather than on setting up and proceeding processes in order. XP development practices include test driven development, pair programming, and refactoring.

- **Test driven development**

This is a method that creates testing before programming (i.e., “test first”), and then creates programs which pass the testing. By performing the testing first, the required functions are made clear and simple design is possible.

- **Pair programming**

This is a method in which programming is performed by a pair of persons. One person creates a program and the other person gives instruction while checking the program. The pair proceeds by switching each role, and thus it is always possible to perform code review. The technique is also expected to aid in maintaining concentration.

- **Refactoring**

This is a technique that improves the finished code without changing its action (i.e., behavior) that is seen from the outside. It reworks code that is difficult to understand, because of bug fixes or adaption to requirements changes, into easily understood code with high maintainability.

2-1-3 Software Reuse

Software reuse is the concept of using developed software, commercially available software packages, and others to efficiently develop new software. In addition, the reuse of software packages may require **customization**.

- **Partitioning into components**

This is the concept of developing modules (i.e., **componentware**) on the precondition of reuse. Development on the precondition of reuse requires standardization in the development phase, improvement of module independence, and management of module-based units, and therefore, man-hours and cost are increased in comparison with development of the same scale. However, modules created as components have high reliability and can be expected to improve development productivity (i.e., shortening of the development period) and quality. In general, using a large component has a greater effect on reducing development man-hours, but caution must be taken in that the component may be difficult to reuse.

- **Re-engineering**

This is to acquire technology for the creation of new software from software that is already running, and to perform some customization. Within re-engineering, reverse engineering and forward engineering are performed.

- **Reverse engineering**

This is technology for analyzing existing programs and for creating **call graphs** (i.e., directed graphs indicating the relationships among procedures) and specifications.

- **Forward engineering**

This is technology for creating new programs by changing specifications taken from existing programs.

- **Mashups**

This is technology for constructing new services by combining APIs from multiple providers. It is used as technology for **Web2.0**, a newer manner of using the Web.

- **Architecture pattern**

This is a pattern of software construction that supports structures, such as packages, subsystems, and layers, and their connection and interaction. It uses existing architecture patterns and can improve software development efficiency.

- **MVC (Model-View-Controller) model**

This is a design model that implements software by combining “models” that form the core of processing, “views” for display or output, and “controllers” that control the views and models according to input content.

- **Design pattern**

This is a design pattern that is used in object-oriented design, and is composed of three types of patterns: creational pattern, structural pattern, and behavioral pattern.

2 - 2 Software Design Technique

2-2-1 Structured Design

Structured design is a technique (i.e., structured technique) that analyzes and designs the functions required for software and the data flows used by each function. The technique in which development is focused on functions (i.e., processing) as in structured design is called the **POA (Process-Oriented Approach)** or **process-oriented design**. Structured design offers merits that include improvement of processing efficiency, ease of maintenance, and partitioning of modules into components (for reuse).

In structured design, functions are incrementally refined according to the following procedures, and are placed into a hierarchical structure.

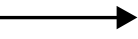


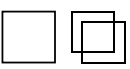
[Structured design procedures]

- 1) Identification of functions
- 2) Clarification of data flow
- 3) Grouping of functions
- 4) Hierarchical structuring
- 5) Determination of program functions
- 6) Documentation of functional specifications

“1) Identification of functions” and “2) Clarification of data flow” are processes that correspond to the software requirements definition process, and are also called **structured analysis**. In structured analysis, work processes, and others are analyzed by using diagramming methods as shown below, and the functions necessary for the software are clarified.

- **DFD (Data Flow Diagram)**

This is a diagramming method in which the targeted workflows are represented as data flows passed among processes. It is used in business operations modeling, which represents processing/functions and flows of data used in work.

Symbol	Name	Meaning
	Data flow	Represents flows of data.
	Process	Represents activities and processes including data processing/conversion.
	Data store	Represents stored data (e.g., ledgers, files, databases).
	Data source (External)	Represents the originating origin (i.e., source) or destination (i.e., sink) of data.

- **State transition diagram**

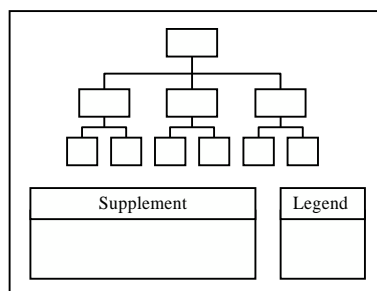
This is a diagramming method that represents the state that changes depending on the course of time, actions, and others. It is used when workflows (i.e., processing/functions) are visually analyzed or for screen transitions.

“3) Grouping of functions” through “6) Documentation of functional specifications” are processes that correspond to the software architectural design process and the software detailed design process. In these processes, functions required for software are organized (i.e., classified) and analyzed by using diagramming methods as shown below, and functional hierarchies are made clear through stepwise refinement.

- **HIPO (Hierarchy plus Input Process Output)**

This is a diagramming method that represents the functions and processing of software by using a hierarchical structure.

<Visual table of contents>



<IPO Diagram>

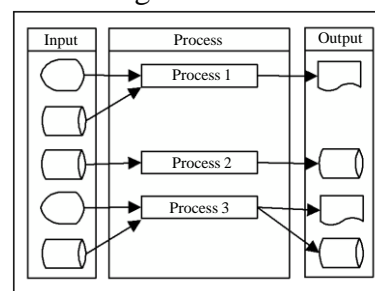
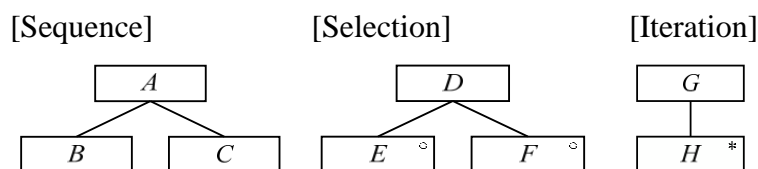


Diagram name	Role
Visual table of contents (hierarchical structure diagram)	A figure representing the functions of software (or a system) as a hierarchy
Summary diagram (IPO diagram)	A figure representing the input, process, and output of software (or a system)
Detailed diagram (IPO diagram)	A figure representing the input, process, and output of the components (or functions) composing the whole

- **Jackson method / Warnier method**

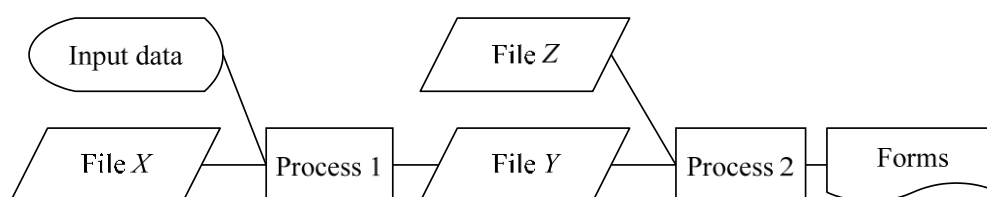
This is a module partitioning technique that is focused on data structures. Each technique defines functional structure diagrams that correspond to data structures.

[Diagram structure of Jackson method]



- **Process flow**

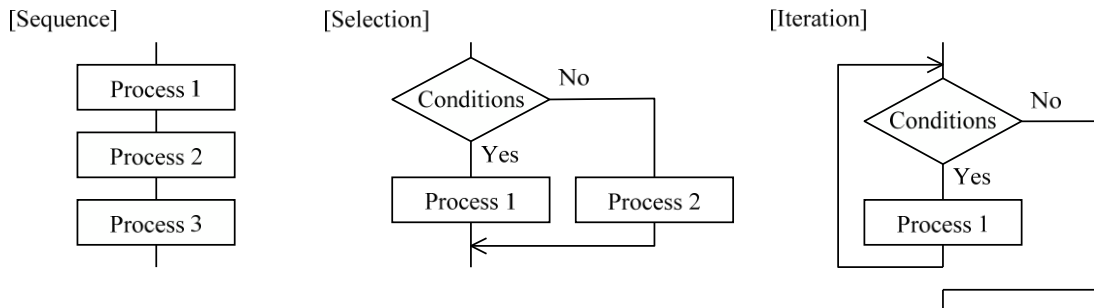
This is a diagramming method that represents the execution order (i.e., flow) of processing (i.e., functions), along with interfaces (e.g., input/output).



Structured charts as shown below are used to represent the processing procedures of each function (i.e., program) in “6) Documentation of functional specifications.” A structured chart is a chart suited to **structured programming**, in which processing procedures are considered according to the **structured theorem** that states that a program with one entrance and one exit can be represented as a combination of three basic structural units (sequence, selection, and iteration).

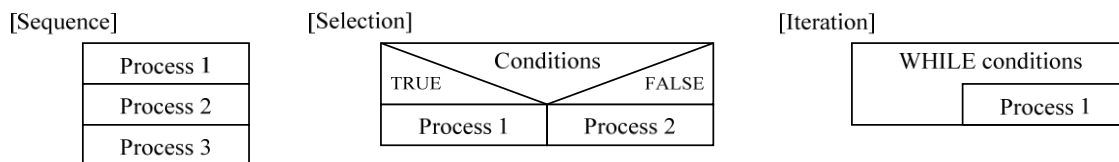
- **Flowchart**

This is a commonly used method of algorithm notation. It represents processing procedures by using combinations of symbols, and has the merit of being easy to understand visually.



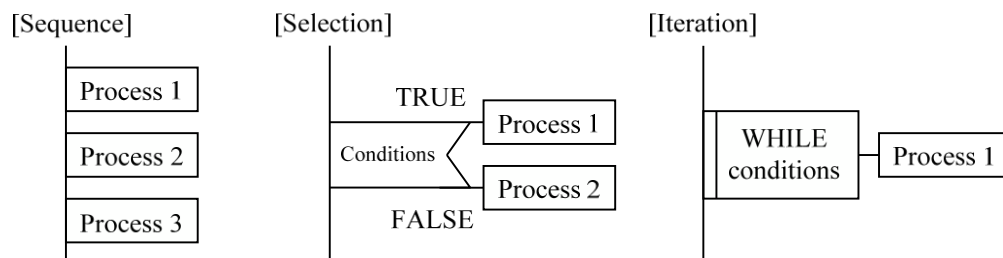
- **NS (Nassi-Shneiderman) chart**

This represents the basic control structure (i.e., sequence, selection, iteration) by using quadrilaterals.



- **PAD (Problem Analysis Diagram)**

This represents the logical structure of algorithms by using a tree structure.



By performing hierarchical structuring and stepwise refinement of functions through structured design, the software structure is refined in the following order: software configuration items → software components → software units (i.e., modules).

In stepwise refinement, the fulfillment of **software requirements** and **software quality characteristics** (i.e., functionality, reliability, usability, efficiency, maintainability, portability) must not be forgotten. In addition, in partitioning into software units (module partitioning), care should be taken to perform optimal module partitioning by evaluating module independence through module strength and module coupling.

2-2-2 Object-Oriented Design

Object-oriented design is an analysis/design technique based on the concepts of **object orientation**. Techniques for development with a focus on data, like object-oriented design, are called **DOA (Data Oriented Approach)** or **data-oriented design**. In object-oriented design, software is partitioned into components called objects, which are combined during development. This enables higher efficiency and quality in software development.

(1) Object orientation

In **object orientation**, objects with shared properties are collected as a **class**. (This process is called **abstraction**.) An embodiment of this class is called an **instance**.

Example: Class "Animals"

Name	Number of legs	Character	Cry
------	----------------	-----------	-----

Instance

Cat	4	Self-centered	Meow
-----	---	---------------	------

Dog	4	Obedient	Woof
-----	---	----------	------

Fig. 4-7 Class and instances

Classes are defined by data and also by the procedures (i.e., processes) for that data. This is called **encapsulation**, while the data is called **attributes** and the procedures (i.e., processes) are called **methods**. Encapsulation can be depicted graphically as follows:

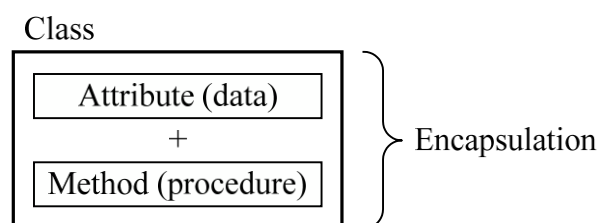


Fig. 4-8 Image of encapsulation

Through encapsulation, users need only send processing instructions (i.e., **messages**) to process data, without being concerned about the data. This is called **information hiding**. Classes can be reused in different software as components. At this time, classes can be

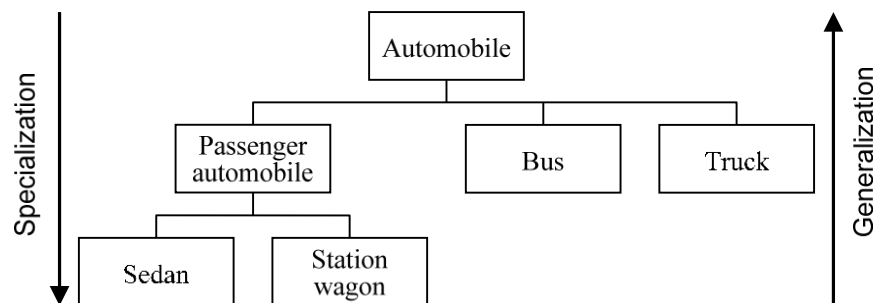
grouped and managed as **packages** to facilitate reuse. Also, typical object-oriented languages, such as **Java**, offer APIs, which are packages that group reusable classes.

While classes can be used alone, associating multiple classes further enhances usability and others.

Class relationships (i.e., associations) in object orientation include the following:

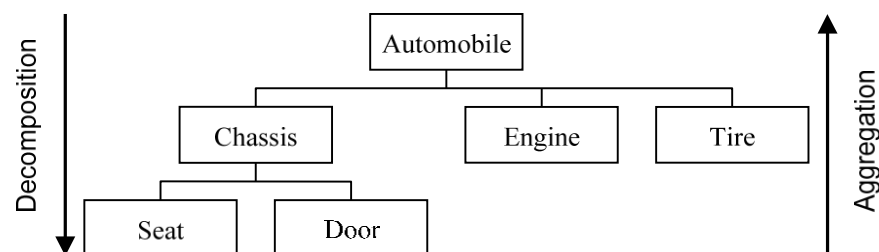
- **“is-a” relationship** (specialization/generalization relationship)

This is a relationship in which sections common to several classes are extracted to define a higher-level class. The higher-level class is called a **superclass** and the lower-level classes are called **subclasses**. Extracting the common sections from multiple subclasses is called **generalization**; conversely, refining a superclass is called **specialization**.



- **“part-of” relationship** (aggregation/decomposition relationship)

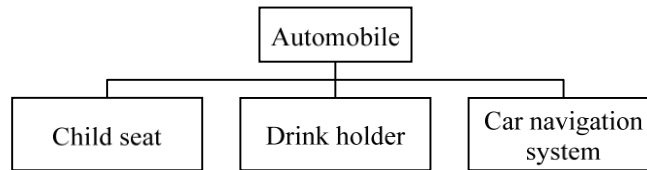
This is a relationship in which multiple lower-level classes are aggregated to compose a single higher-level class. The development of higher-level classes into lower-level classes is called **decomposition**, and the grouping of lower-level classes into higher-level classes is called **aggregation**.



- **“has-a” relationship**

This relationship means that a higher-level class holds a lower-level class. This relationship has almost the same meaning as the “part-of” relationship. However, they differ in that whereas the higher-level class is not possible in the “part-of” relationship

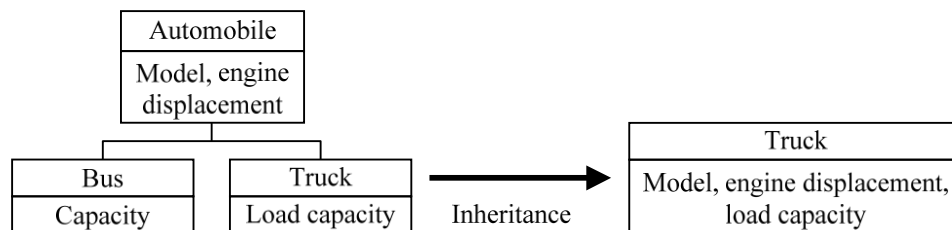
if even one lower-level class is missing, it is possible in the “has-a” relationship even if multiple lower-level classes are missing.



Characteristics of object orientation include the following:

- **Inheritance**

Attributes defined in an upper-level class can be passed (or inherited) to lower-level classes. Inheritance includes single inheritance from one superclass, and multiple inheritance from multiple superclasses. Inheritance does not exist in “part-of” relationships. The rewriting, by an inheriting lower-level class, of methods that are defined in a superclass is called **override** (**redefinition**).

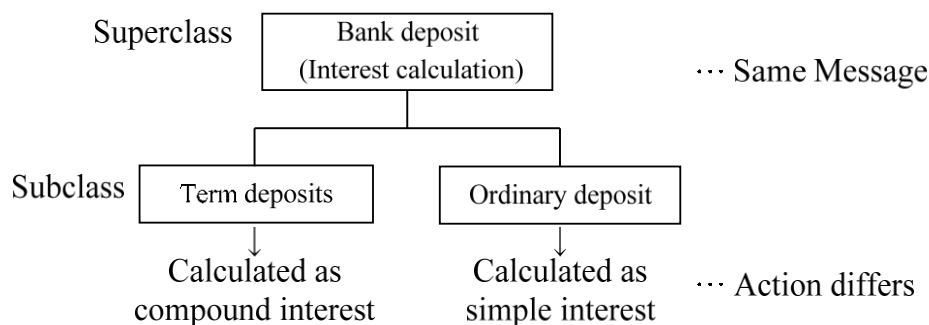


- **Differential programming**

This is a programming technique in which a superclass is inherited, and only the portions differing from the superclass are defined in the subclasses.

- **Polymorphism**

This is the ability of actions (or behaviors) for a given message to differ according to the class (i.e., object). Polymorphism is implemented by **override** (**redefinition**) etc.



- **Delegation**

This is the ability to delegate processing to other classes (i.e., objects).

- **Propagation**

This is the ability, when operations are applied to a given class (i.e., object), to automatically apply the operations to other related classes (i.e., objects).

- **Role**

It is possible to group multiple instances and define roles. Actions will change when roles differ, even for instances of the same class.

Class relationships are extremely important in object orientation. Since many attributes in object orientation are also determined by class relationships, classes and the relationships among classes are summarized as a **class diagram**. Class diagrams are diagrams included in the UML (Unified Modeling Language) described later.

(2) Object-oriented development model

Object-oriented development model is a spiral model-type development technique that begins with analysis/design/development of important classes, and proceeds in stages to analysis/design/development of detailed classes.

[Development phases in the object-orientated development model]

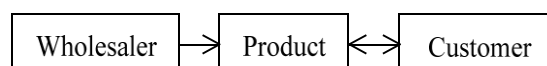
Object-oriented analysis	Object-oriented design		Object-oriented programming
	Architectural design	Class design	

(i) **OOA (Object-Oriented Analysis)**

This models the real world that is the target of development through object orientation (i.e., the integration of data and procedures). E-R diagrams and state transition diagrams are used in the modeling.

- **E-R diagram / ER model**

These are data models that represents the targeted work as entities and relationships among entities. They are used in data modeling, and others which represents the data structures used in work.



- **One fact in one place**

This is a concept in data analysis that manages one fact (i.e., information) in one place. It checks whether the same data, used in multiple tasks, is managed

under different names. In a database, one fact in one place is achieved by performing normalization.

(ii) **OOD (Object-Oriented Design)**

The architectural design (i.e., global design) that equates to external design in the waterfall model and the class design (i.e., local design) that equates to internal design are carried out.

(iii) **OOP (Object Oriented Programming)**

Object orientation is more the concept of clearly defining methods of using classes than the concept of creating programs. In this phase, class libraries (i.e., libraries for preserving created classes) and packages are referenced in order to reuse existing classes. Programming languages used include Java and C++.

- **CORBA (COMmon Request Broker Architecture)**

This is a standard specification that enables message exchange among objects developed by using different programming languages in a distributed environment.

- **IDL (Interface Definition Language)**

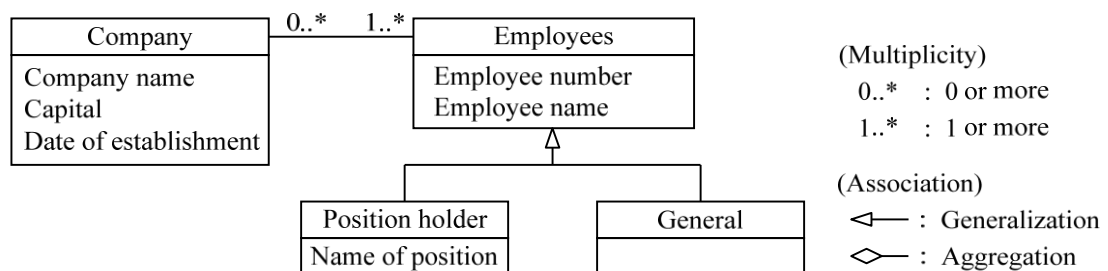
This is a set of rules concerning interfaces for reusing created objects from other programming languages.

(3) UML (Unified Modeling Language)

UML is a standard unified modeling language approved by the **OMG (Object Management Group)** (a standardization body for object-oriented technologies). It is used in the notation of deliverables (e.g., specification documents) in object-oriented development, from analysis to design, implementation, and testing.

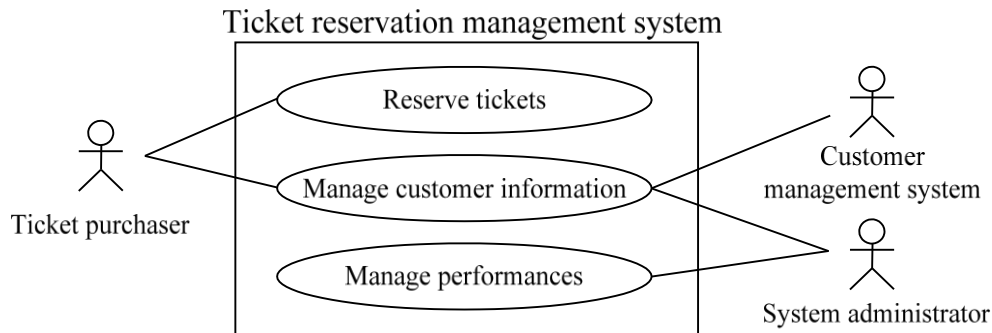
(i) **Class diagram**

This is a diagram that represents the relationships among classes. It describes class names, operations, attributes, multiplicity, role names, and others. It is also used as data models.

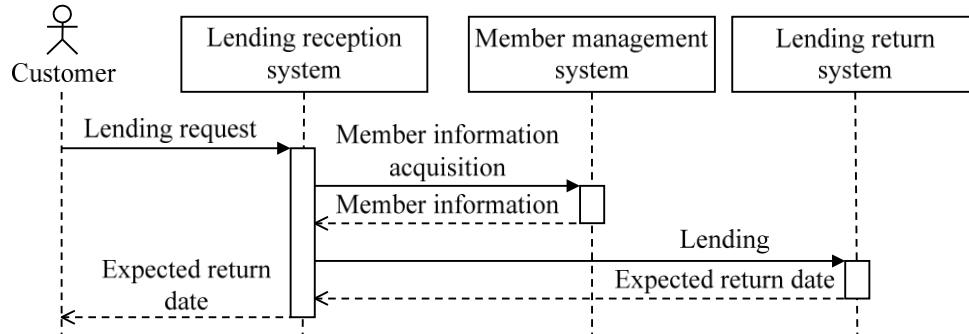


(ii) **Use case diagram**

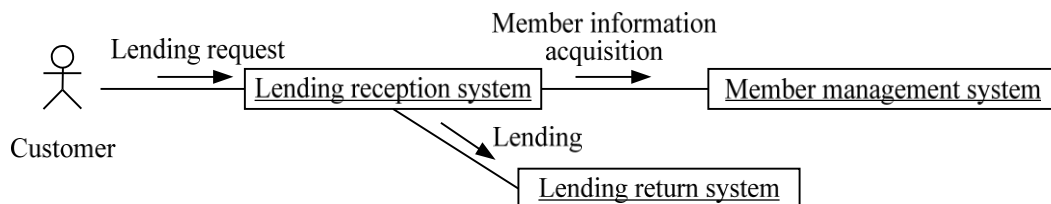
This is a diagram that represents scenarios for what the system will do, from the point of view of the actors (i.e., external users or machines that will boot or exchange information with the system).

(iii) **Sequence diagram**

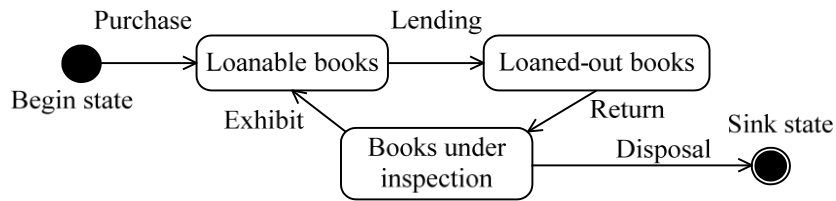
This is a type of interaction diagram (i.e., a diagram that depicts exchange of messages among objects), and represents message transmission and object lifelines in a time series.

(iv) **Communication diagram (collaboration diagram)**

This is a type of interaction diagram that represents exchange (i.e., message flow) of messages, with a focus on objects and such others.

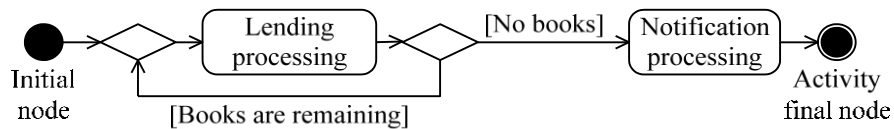
(v) **State machine diagram (statechart diagram)**

This is a diagram that represents the state transitions which occurs during object life cycles.



(vi) **Activity diagram**

This is a diagram that represents the flow of systems and work.



(vii) **Component diagram**

This is a diagram that represents the configuration of systems and software. In general this is a diagram that depicts software component configuration, and on the other hand, a diagram that depicts system hardware configuration is called a deployment diagram.

(viii) **Object diagram**

This is a diagram that represents the relationships among objects (i.e., instances).

(ix) **Package diagram**

This is a diagram that represents the associations among packages of grouped classes.

(x) **Timing diagram**

This is a diagram that represents message exchanges and the status of object lifelines that change over time.

[Other languages]

- **ADL (Architecture Description Language)**

This is a language that describes the architecture (i.e., structure) of systems (i.e., software). UML can be considered a type of ADL.

- **DDL (Data Definition Language)**

This is a language that defines the logical structure and names of databases.

2 - 3 Development Process

(1) SLCP-JCF (Software Life Cycle Process-Japan Common Frame)

SLCP-JCF is the Japanese version of SLCP, which is standardized by ISO/IEC as a common frame that clarifies the life cycle (e.g., development, operation, maintenance), with the aim of optimization of software development and its transactions. At present, Common Frame 2013 (SLCP-JCF2013), jointly drawn up by users, vendors, academic experts, and others in Japan, is in use.

The content of Common Frame 2013 contains ISO/IEC 12207 (JIS X 0160), which was standardized as a “common yardstick” for development work overall, in order to clarify dealings between software acquirers and providers. **JIS X 0160** is a JIS standard that provides collection of defined processes necessary for smooth communication among the acquirer, provider, and other concerned parties in the life cycle of a software product. It also defines the responsibilities of the acquiring party, such as the clarification of acceptance criteria and procedures.

(2) CMMI (Capability Maturity Model-Integrated)

CMMI (Capability Maturity Model-Integrated) is a process improvement model developed at the SEI (Software Engineering Institute) at U.S. Carnegie Mellon University. It evaluates the maturity level of development processes through quality management standards that objectively indicate software development capabilities.

It evaluates capabilities as a software development organization on a scale of 1 to 5 as shown in the table below, on the basis of the results of project management practice.

Maturity level	Overview
Level 5 Optimizing	This is the level at which all persons participate in process improvements, and improvement activities become routine. Continuous improvements in processes are implemented through feedback and an innovative technology orientation.
Level 4 Quantitatively managed	This is the level at which quantitative management of processes and products is performed. Detailed quality data on processes/products is collected, and understanding and control of processes and products can be performed on the basis of data.
Level 3 Defined	This is the level at which process management is conducted organizationally. Definition and integration of management processes and development processes are conducted, and all

	projects observe documented processes.
Level 2 Managed	This is the level at which basic project management is performed. Processes for rudimentary management of schedules, expenses, and functionality are established, and process discipline exists which can repeat success experiences in the same area.
Level 1 Initial	This is the level of an organization that has not achieved Level 2. Work is performed on an ad hoc basis and is chaotic at times, with almost all processes undefined.

3 System Development Environment

A system development environment is an environment used to develop systems (or software). An appropriate system environment is necessary to perform development efficiently.

3 - 1 Intellectual Property Application Management

Intellectual property application management is the management of intellectual property rights related to system development. Intellectual property application management aims to protect a company's own intellectual property rights, and to avoid infringing on the intellectual property rights of other parties.

(1) Copyright management

In **copyright management**, the rights (**copyrights**) of the authors of works are managed. In system development, software (i.e., program) copyrights are mainly the target of management. In the case of **employee works** in particular, copyrights are managed so that the corporation or such other organization becomes the author as far as there are no agreements or work rules. (Refer to p.96 for more about copyrights.)

In order to protect its own copyrights, a company uses technologies that may include **copy guards** to prevent the duplication of works, **DRM (Digital Rights Management)** to protect digital data works, or **activation** to require license registration before use.

(2) Patent management

In **patent management**, applications (to the Patent Office) for **patent rights** to inventions newly conceived in the development process, and licensing of the use of patents held by others, are managed. Forms of licensing, such as **cross-licensing** and **patent pools**, should be considered in addition to fee-based license agreements. Furthermore, patent rights include **non-exclusive licenses** and **exclusive licenses**, and it is impossible to use a patent when an exclusive license is registered for the patent. (Refer to p.97 for more about patent rights.)

(3) License management

In **license management**, a company manages license agreements when it uses software for which it does not own copyright. The company must also perform management so that the

number of users of the software does not violate licensing agreements. (Refer to p.113 for more about licensing agreements.) The party conferring a license is known as the **licenser**, and the party receiving the license is the **licensee**.

3 - 2 Development Environment Management

In **development environment management**, the hardware, software, network, development tools, and such others to be used in development are prepared (or constructed) in line with development requirements, and are appropriately managed.

(1) Development environment operating status management

In **development environment operating status management**, the operating status of the development environment (e.g., computer resources, development tools) is observed, and is appropriately managed for conducting efficient development.

- **Resource management**

Computer resources, development tools, and other configuration management items are managed, and are prepared so that necessary resources are available when needed.

- **Operations management**

The development environment is managed so as to be operated appropriately, and operating status is observed.

- **Maintenance management**

The development environment is always maintained in an appropriate state. Maintenance of appropriate status does not always update the environment to the latest state, but rather periodic confirmation of operation to assure the ability to use the environment appropriately in line with objectives.

(2) Design data management

In **design data management**, version management, sharing management, safety management, and such others are performed for the various kinds of data (e.g., specifications documents) involved in design.

- **Change history control**

The update history (i.e., versions) of the objects to be controlled is managed. In the version management system (i.e., database) for unified management of important information, various types of definition information, design information, program

information, test information, and such others are managed in a **repository**. In order to use the repository effectively, the format and such other of documents to be recorded is important. Recently, there is increased emphasis on design (i.e., **document design**) of physical/logical structure of design documents, reports, and other documents coupled with process automation and management methods. For example, there is a method of designing all documents in XML format and managing these in an XML-compatible repository by effectively using text information, diagrams, images, and such others.

- **Access rights control**

Appropriate access rights are to be set and managed as required to prevent unauthorized search/removal/falsification of a company's confidential information, such as trade secrets and personal information.

(3) Tool management

In **tool management**, the types and versions of development tools to be used are managed for the effective development of software. Developed software carries the risk of compatibility problems, because of differences in types and versions of development tools. Moreover, the occurrence of problems, such as bugs or security holes originating in development tools, may have adverse effects on the developed software. For those reasons, it is necessary to manage the development tools used.

[Typical development tools]

- **IDE (Integrated Development Environment)**

This is a general name for tools that provide integrated support for the overall development process. Typical IDEs include the OSS **Eclipse** and such other tool.

- **Design tools** (design support tools)

These are tools to support design processes, and include support tools for various designs (e.g., screen design), database design support tools, and library management tools.

- **Building tools** (programming support tools)

These are tools to support programming, and include language processors (e.g., compilers, interpreters), editors, and such others. In the way that programs created with editors are translated by a compiler, the linking of development tools (in which the output of one tool is entered into the next tool) is called a **tool chain**.

- **Program test support tools**

Program inspection test tools (i.e., test support tools) include **program static analysis tools** to support static testing and **program dynamic analysis tools** to support dynamic testing.

- **Test execution support tools**

These are test tools (i.e., test support tools) to support test implementation.

- **Stub / driver tool**

This automatically generates stubs or drivers.

- **Test data generator**

This automatically generates test data.

- **Dump**

The content of main memory and registers is written to a dump file. This type of tools includes **memory dump** that writes the status of main memory, and **snapshot dumps** that writes the status at a given time.

- **Tracer**

This traces the running state of a program. This is often used together with an **inspector** that writes the values of the variables used in a running program.

- **Assertion checker**

This tests the validity of programs. (This test is called assertion check.)

- **Emulator**

This is a tool (i.e., a microprogram-based tool) that creates a virtual environment including a different OS or such other software, in which programs made for other computer types can be executed.

- **ICE (In-Circuit Emulator)**

In microprogram development, this emulates microprocessor functions through hardware.

- **Simulator**

This is a tool that simulates the actions of programs.

- **CASE (Computer Aided Software Engineering) Tools**

CASE refers to a group of software to aid the processes of system development.

Upstream CASE tool	This primarily supports design.
Downstream CASE tool	This primarily supports implementation (building) and testing.
Maintenance CASE tool	This primarily supports operation/maintenance.
Integrated CASE tool	This provides integrated support for all development processes.

- **VDM (Virtual DOS Machine) Tool**

This is a development support tool for VDM, one of the **formal methods** (i.e., methods that rigidly describe according to formal specification language to enhance

software quality).

Use of these development tools can promote **EUC (End User Computing)**, by which users actively engage in system development, and **EUD (End User Development)**, by which users themselves perform system development. For example, integrated CASE tools or such others are used in RAD (Rapid Application Development), by which users participate in development work from an early stage. EUC is a general name for the use of spreadsheet software, database software, and such other application software to execute work, and the technologies or methods to achieve this.

(4) License management

In **license management**, the number of target software installations and licenses held are regularly checked and verified with understanding of the content of the licenses, and management is performed to avoid violation of licensing conditions (e.g., unauthorized copying of software and other unauthorized usage).

[Points of caution for license management]

- Software installation information, hardware configuration information, network configuration information, and such others are collected from clients and are recorded in databases by using **inventory collection** functions and such other functions to accurately assess usage status.
- The number of licenses is accurately assessed by performing inventory taking and such other control.
- Since using different versions of software can violate licensing conditions, software version management is also to be conducted.

3 - 3 Configuration Management and Change Control

3-3-1 Configuration Management

The purpose of the **configuration management** process is to establish and maintain the integrity of all identified outputs of a project or process and make them available to concerned parties.

[Activities/tasks of configuration management process (Common Frame 2013)]

- (i) Configuration management planning
 - Define a configuration management strategy
 - Identify items that are subject to configuration control

- (ii) Configuration management execution
 - Maintain information on configurations
 - Secure configuration baselines

SCM (Software Configuration Management) is specialized as a software configuration management process. In this process, configuration identification system, or what combination of **SCI (Software Configuration Items)** composes the software as a whole, is established, and management methods for the configuration identification system are set.

3-3-2 Change Management

Change management refers to the software configuration management process activities/tasks, in which changes to software configuration items are centrally managed and reflected in configuration management.

[Activities and tasks related to change control]

- (i) Configuration control
Identification and recording of change requests, analysis and evaluation of changes, approval or rejection of change requests, and implementation, verification, and release of modified software items are performed.
- (ii) Recording of configuration status
The number of times changes are made in a project, the latest version, migration (or release) status, and such others are documented as management records and status reports.
- (iii) Evaluation and assessment of integrity of configuration items
The functional completeness (e.g., consistency, accuracy) and the physical completeness (e.g., whether the latest technical descriptions are reflected) of software items are determined and guaranteed.
- (iv) Release management and delivery (or shipment)
The release and delivery of software products and documentation are formally controlled. Master copies of the delivered items are maintained, with version management, for a period of retention equal to the lifespan of the software product, which is based on SLCP (Software Life Cycle Process).

4 Web Application Development

Web application development is the development of web applications used in the web services on the Internet. Since these presume use of the Internet, they may require technology differing from that of general software development.

4 - 1 Web Applications

Web application is a general name for software used in web services. In general, this often means software that runs on the **web servers** that compose **web systems** and provide services to **web clients**.

Web systems are **client/server systems** in which web clients make a processing request to web servers that provide web services. However, in the case of web systems that use databases, **3-tier client/server systems** are often used. The system configuration of 3-tier client/server systems uses web browsers at the presentation layer, **application servers** (web servers) at the function layer, and database servers at the database access layer. Using an application server as the web server facilitates adaption to system changes or enhancements.

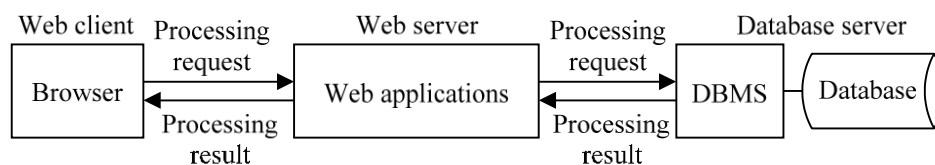


Fig. 4-9 Web system configuration

A web server launches and executes web applications in response to processing requests from web clients (i.e., browsers). This mechanism is known as **CGI (Common Gateway Interface)**, and the web applications that run on web servers are sometimes called CGI programs.

Web applications receiving processing requests use database servers as required. At this time, a processing request from a web application is passed as SQL statements to a database server, which in turn returns the execution result of the SQL statements to the web application. Then, the web application returns HTML document, dynamically generated from the received result of executing the SQL statements, to the browser, which displays the HTML document.

Web applications executed on the web server side, as in this example, are called **servlets**. Conversely, web applications downloaded to and executed by the web client are called **applets**, and the feature by which such applets are downloaded to and executed by web clients as necessary is called **rich clients**.

4 - 2 Web Application Development

In **web application development**, depending on usage conditions, web design and security technologies may be necessary in addition to general software development technologies.

(1) Server-side programming

Server-side programming is the development (i.e., programming) of web applications that run on web servers. Programming languages, such as Java, PHP, and Perl, are often used in server-side programming.

In addition to general programming points of caution, security measures are emphasized in server-side programming. In particular, in the case of a website or other use by a large number of anonymous users, **secure programming** is required to create programs that implement various security measures and have no security holes (i.e., software vulnerabilities).

(2) Web design

Web design is the logical design of a website overall. Web design strives for not only technical design but also “user-friendly” design, in consideration of “ease of use (**web usability**)” and “ease of access (**web accessibility**).”

The validity, efficiency, and satisfaction demanded by web usability is evaluated by experts through **heuristic evaluation**, by test subjects (i.e., users) through **usability testing**, and other means.

(3) Other related technologies

- **SOAP (Simple Object Access Protocol)**

This is a protocol in which web applications cooperate by exchanging messages described in XML. Web applications that can be used from outside via SOAP may be called web services.

- **Ajax (Asynchronous JavaScript + XML)**

This is a mechanism by which browsers and web servers transmit and receive XML-format data without synchronizing, to dynamically redraw screens. This mechanism enables redrawing of only necessary portions of a web page, without screen transitions.

- **RSS (RDF Site Summary)**

This is an XML-based document format that enables structuring and writing of metadata, such as page headlines, summaries, and update time. It enables efficient information collection and communication on web sites.

Chapter 4 Exercises

Q1

In system development, which of the following is the process that clarifies configuration items at the top level of a system?

- a) System integration process
- b) System installation process
- c) System architectural design process
- d) System requirements definition process

Q2

Among the software quality characteristics defined by ISO/IEC 9126 (JIS X 0129-1), which of the following is an explanation of reliability?

- a) The capability of the software product to maintain a specified level of performance when used under specified conditions
- b) The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions
- c) The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions
- d) The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions

Q3

Which of the following is the deliverable that can be obtained as the result of software architectural design?

- a) A specification document for system architectural design
- b) A specification document for software integration test
- c) A specification document for software qualification test
- d) A document for software requirements definition

Q4

Among module partitioning techniques, which of the following is a technique that partitions input functions, conversion functions, and output functions according to data flow, and makes each into a module?

- a) STS partitioning
- b) Common functional partitioning
- c) Jackson method
- d) Transaction partitioning

Q5

As a measure of module independence, it can be said that the weaker the module coupling, the higher the level of module independence becomes. Which of the following is the coupling with the highest module independence?

- a) Common coupling
- b) Stamp coupling
- c) Data coupling
- d) Content coupling

Q6

Which of the following is an appropriate explanation of a code auditor?

- a) It verifies whether the created source code fulfills the criteria that are set in advance.
- b) It displays a menu of candidate names for a variable when the name of the variable is being entered.
- c) It creates programs by using a combination of three basic structures.
- d) It displays the input command in color according to the type of the command.

Q7

A program has the complex conditions below.

Condition 1 OR (Condition 2 AND Condition 3)

When testing is performed on the basis of the decision condition coverage (i.e., branch coverage), test coverage is not fulfilled by executing only (1) and (2) of [Completed test items] shown below. Which of the following is the most appropriate test item to be added?

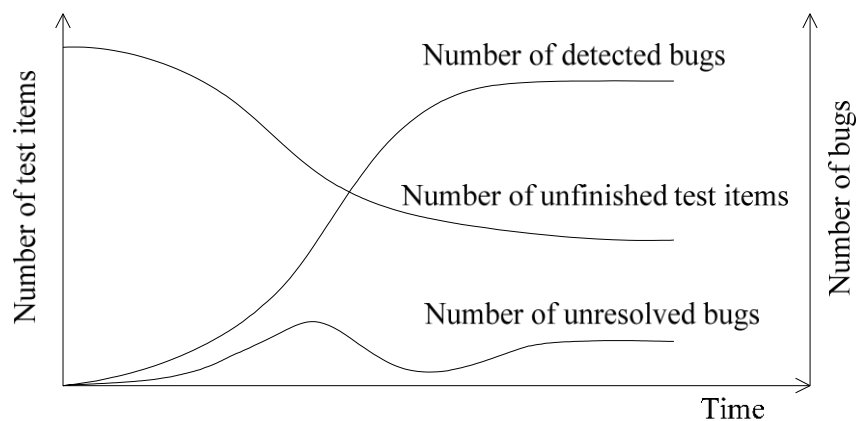
[Completed test items]

- (1) Condition 1 is TRUE, Condition 2 is FALSE, and Condition 3 is FALSE
- (2) Condition 1 is FALSE, Condition 2 is TRUE, and Condition 3 is TRUE

	Condition 1	Condition 2	Condition 3
a)	FALSE	FALSE	TRUE
b)	TRUE	FALSE	TRUE
c)	TRUE	TRUE	FALSE
d)	TRUE	TRUE	TRUE

Q8

In program testing, all lines become flat, as shown in the graph below. Which of the following is the testing status that can be surmised from this graph?



- a) The project faces bugs that are difficult to resolve, and subsequent testing is not progressing.
- b) The number of finished test items is increasing, and bugs no longer occur.
- c) There are many bugs, and the number of finished test items is no longer increasing.
- d) The rate of occurrence of bugs matches the rate of the finished test items, and there are no more unresolved bugs.

Q9

Which of the following is an appropriate description of top-down testing?

- a) It is required to create drivers that function as lower-level modules.
- b) Since higher-level modules are used repeatedly in testing, the reliability of higher-level modules is enhanced.
- c) Since testing is conducted from both higher-level modules and lower-level modules, programming and testing can be conducted in parallel from the initial stage of development.
- d) When a problem occurs in the interface among modules, it is difficult to identify the locations of the causes of the problem.

Q10

Since the sales tax was changed after the delivery of an invoice issuing system, maintenance was implemented to revise the sections concerned with invoice amount calculation. What is this sort of maintenance called?

- a) Remote maintenance
- b) Corrective maintenance
- c) Adaptive maintenance
- d) Daily maintenance

Q11

Which of the following is an appropriate explanation of the software development model known as the waterfall model?

- a) Design, development, and testing of applications are carried out and repeated on a part-by-part basis.
- b) Since software development progresses in order of phases, software development efficiency considerably declines at the occurrence of rework.
- c) An executable trial model is created, and requirements specifications are confirmed and evaluated at an early development stage.
- d) Software is developed in a short period of time by a small group of people through the active participation of users and the use of development tools.

Q12

Which of the following is a method that improves software quality by starting programming at an early stage to enhance programming and testing, instead of spending time to create a high-quality design document?

- a) XP (eXtreme Programming)
- b) Test-driven development
- c) Pair programming
- d) Refactoring

Q13

Which of the following is an appropriate combination of the components of the diagramming method known as HIPO?

- a) Model, View, Controller
- b) Sequence, Selection, Iteration
- c) Visual table of contents, summary diagram, detailed diagram
- d) Data flow, process, data store

Q14

In object orientation, which of the following is the characteristic by which attributes defined in a superclass are passed to subclasses?

- a) Inheritance
- b) Override
- c) Delegation
- d) Polymorphism

Q15

Which of the following is an appropriate explanation of a UML sequence diagram?

- a) This is a diagram that represents the relationships among objects and is used for the purpose of organizing message interactions in a time series.
- b) This is a diagram that represents state transitions in objects and is used for the purpose of organizing object life cycles.
- c) This is a diagram that represents system functions as viewed externally and is used for the purpose of organizing functional requirements through requirements definition,

and so on.

- d) This is a diagram that represents the relationships among classes and is used for the purpose of organizing data relationships.

Q16

Which of the following is a development tool that writes the values of the variables used in a running program, for the purpose of tracing program execution?

- a) Assertion checker
- b) Inspector
- c) Emulator
- d) Memory dump

Q17

In software development, which of the following is a problem that originates in configuration management?

- a) Multiple versions of the same program exist, and the latest version that should be modified cannot be identified.
- b) Test data (i.e., data that is used in the real working environment) that is used in development is discovered through search by a party that is not meant to see the data, and is leaked to the outside.
- c) There is a request for improvement of a program delivered several years earlier, but the development environment from that time is not maintained, and the request cannot be responded.
- d) In software qualification testing, there are many bugs at the software unit testing level, and development is not proceeding on schedule.