

ESPA

Embedded System development Process Reference guide

Written and edited by
Software Engineering Center,
Technology Headquarters,
Information-technology Promotion Agency, Japan

This document has been published as the English edition of ESPR (Embedded System development Process Reference) Version 2.0 published by IPA/SEC* in Japan. ESPR describes the standard types of work and best practices for facilitating the processes in embedded software development.

The purpose of ESPR is to be used as the reference guide for improving the current conditions of high-quality embedded software development and promoting its efficiency by defining and organizing the development processes that best suit the interests and needs of individual organizations, groups and/or projects.

October 2012

Software Engineering Center, Technology Headquarters,
Information-technology Promotion Agency, Japan

Copyright © 2012, IPA/SEC

Permission to copy and distribute this document is hereby granted provided that this notice is retained on all copies, that copies are not altered, and that IPA/SEC is credited when the material is used to form other copyright policies.

* Software Engineering Center, Technology Headquarters, Information-technology Promotion Agency, Japan

Foreword

On the occasion of publication of ESPR Ver.2.0

In fiscal year (FY) 2006, Software Engineering Center (hereafter “SEC”) of Information-Technology Promotion Agency, Japan (IPA) published the first edition of Embdeded Software Development Process Reference (hereafter “ESPR Ver 1.0”), which intended to describe the embedded software development process in an orderly manner. Since its release, SEC has been able to enjoy seeing ESPR Ver 1.0 read and utilized by an unexpectedly large number of embedded software engineers. ESPR Ver 1.0 has been published in advance as the outcome of careful review on Software Engineering Process (SWP) and a part of Support Process (SUP) carried out within SEC in response to the growing needs in the embedded software industry to further smoothen the development process that has lately become increasingly large-scaled and complex.

But when the embedded system is perceived as a product with embedded software, the software development process must take account of the System Engineering Process (SYP), which includes system designing performed prior to software development. For this reason, SEC has, in FY 2006, proceeded with the efforts to review and define SYP by gaining cooperation again from all those who contributed in the making of ESPR Ver.1.0. The result of their efforts has led to the creation of ESPR Ver.2.0, which not only includes the descriptions on SYP, but also provides various sets of information to supplement and update the original contents of ESPR Ver.1.0. In ESPR Ver. 2.0:

- Description on System Engineering Process (SYP) has been newly added;
- Description on Safety Engineering Process (SAP) for securing system safety and reliability has been newly added;
- Description on Support Process (SUP) has been partially supplemented with additional information;
- Typographic errors and misleading information in ESPR Ver.1.0 have been corrected.

In this way, ESPR Ver.2.0 consolidates know-how to achieve smooth development of embedded software based on ESPR Ver.1.0. We hope ESPR Ver.2.0 will also help embedded system engineers as much as ESPR Ver.1.0.

November 2007

Embedded Software Engineering Section, IPA/SEC

Acknowledgment

“What should be done to develop high-quality embedded software efficiently?”

This is probably one of the most frequently asked questions during our two years of research in SEC (headed by Dr. Seishiro Tsuruho). In responding to this inquiry, SEC has been promoting the skill building of human resources allocated in software development and consolidation of software engineering practices specific to embedded software, as a solution aimed at enhancing the overall embedded software development capabilities.

This document has been prepared as one of the tangible solutions to the aforesaid efforts to describe, in an orderly manner, the “development process for embedded software” that has been discussed and reviewed within the Embedded Software Engineering Section of SEC. In Japan, embedded software has grown rapidly both in scope and variety over the last few years. We have been witnessing a number of cases of confusion arising among on-site engineers due to the increasing complexity of development tasks. We have therefore organized this document to explain about the tasks and basic operations required in embedded software development in a systematic, easy-to-understand manner by using technical expressions and terminology that can be well comprehended by the on-site engineers. We hope that this document will be helpful to many readers in re-examining their organizations, projects, and individual tasks, and restructuring their style of development into a more efficient form that better suits their given conditions and work environment.

In preparing for its publication, approximately two years have been taken to carefully review, revise and refine the draft text of this document primarily by the members of Development Process Technical Working Group that constitutes a part of Embedded Software Development Improvement and Promotion Taskforce (Chairperson: Hiroshi Monden; Vice-chairperson: Kiichiro Tamaru) jointly formed by the Ministry of Economy, Trade and Industry (METI) and SEC. The members of this working group come from various business and academic institutions who have long been engaged in the implementation and research of software processes as leading experts in embedded software. Thanks to their cooperation, we are pleased to be able to publicize this document.

There is “no silver bullet” in software development. It has been a while since we first heard this phrase, and we know that it still holds true. There is indeed no single solution that can miraculously fix the various issues existing in different stages of software development all at once. It would therefore be our utmost pleasure to see this document serve as one of the effective solutions that could at least help eradicate the demons hidden deep beneath the embedded software development processes.

October 2006

Masayuki Hirayama, Taro Yamazaki, Shuji Muro
Embedded Software Engineering Section, IPA/SEC

Our lives are surrounded by an overwhelming number and variety of information equipment. Many of them provide their required functionalities by means of the so-called embedded system. In the core of this embedded system is the embedded software. In order to develop the embedded software efficiently and ensure that the developed software is of high quality, there is a need to execute the appropriate tasks in the appropriate sequence in the course of development. “ESPR : Embedded System development Process Reference – Development Process Guide for Embedded Software” has been prepared as a document that provides and describes the standard types of work and best practices to smoothen the processes in embedded software development.

Positioning and Composition of This Document

This document is intended to be used by leaders and managers responsible of embedded software development projects as a reference material to examine the development processes that should be carried out by the organizations and/or in the projects that they are supervising.

This document is comprised of the following three sections:

Part 1 Development Process Guide for Embedded Software: Descriptive Section

Part 2 Development Process Guide for Embedded Software: Technical Section

Part 3 Development Process Guide for Embedded Software: Practical Section

Remarks

This document has been prepared on the basis of careful reviews and repeated discussions carried out within Development Process Technical Working Group of Embedded Software Development Improvement and Promotion Committee under Ministry of Economy, Trade and Industry (METI).

Table of Contents

Foreward.....	iii
Preface.....	v

Part 1	Descriptive Section	1
1.1	What Is Development Process?.....	2
1.2	Purpose and Positioning of Development Process Guide.....	3
1.3	Intended Users, Usages and Benefits	5
1.4	Structure of Development Process Guide	7
1.5	Instructions and Directions for Using This Guidebook	12
1.6	Related Standards	14

Part 2	Technical Section	17
2.1	Overall Structure	18
2.2	Process Definition Documents	21
	System Engineering Process (SYP)	21
	Software Engineering Process (SWP)	62
	Safety Engineering Process (SAP)	131
	Support Process (SUP)	148
2.3	Document Template Samples.....	163

Part 3	Practical Section	217
	3.1 Procedure for Practical Use	218
	3.2 Tailoring the Development Process for the Organization / Department	221
	3.3 Designing the Process Phases of the Development Project	222
	3.4 Planning the Development Project Management (Designing the development process phases in detail)	227
	Appendices	233
	Appendix 1. Terminology	234
	Appendix 2. Standards Correspondence Table (Between This Guidebook and X0160)	238
	Appendix 3. List of Activities / Tasks / Sub-tasks	239
Afterword		243

Part **1**

Descriptive Section

1.1 What Is Development Process?.....	2
1.2 Purpose and Positioning of Development Process Guide.....	3
1.3 Intended Users, Usages and Benefits	5
1.4 Structure of Development Process Guide.....	7
1.5 Instructions and Directions for Using This Guidebook.....	12
1.6 Related Standards	14

1.1 What Is Development Process?

What is a Process?

Generally in any type of work, various tasks need to be performed in the course of meeting the purposes and objectives aimed at achieving through that work. In addressing “what kind of tasks need to be performed”, various inputs and outputs, and the overall contents of these tasks are defined and organized in an orderly manner, which are collectively referred to as the “process”.

What is Software Development Process?

In order to complete the development of software as a product, various types of work need to be performed in layers like in any other businesses. The activities deemed necessary in the making of a ‘software product’ will collectively become the so-called “software development process” when they are organized in an orderly manner.

Software Development Process in General

Various concepts in software development process, including ISO/IEC12207, have been advocated and are being practiced today. For the development of systems comprised of such software, international standards like ISO/IEC15288 have also been proposed and established.

However, the existing standards in software development process, on the overall, have been developed to be generic. Due to this nature, they have been often perceived as difficult to follow in case of embedded software development.

Development Process Guide for Embedded Software

This guidebook has been prepared to organize what needs to be done in embedded software development from the standpoint of development process.

Although needless to say, the development of embedded software must comply with the conventional concepts of software development process to a certain degree, since embedded software, after all, is one type of software. Therefore, this guidebook contains numerous references made to existing standards on software development process.

Difference from Software Development Process Models

Some well-known software development process models include the “waterfall development model” and “spiral development model”. These development process models intend to outline the typical types of work involved in software development and their relationships in chronological order.

This guidebook and various relevant international standards like ISO/IEC12207 and 15288, on the other hand, have been compiled to systematically describe the types of work needed in software development, without defining the individual process and work sequences.

1.2 Purpose and Positioning of Development Process Guide

Development Process in Embedded Software Development

Over the past few years, the functional requirements of systems comprised of software have increased extensively, due to the advancement of various technologies used in devices. This trend has led the development of embedded software to also expand in scale in order to meet the ever-growing functional needs. In earlier days, the developers of embedded software did not have to be that conscious about development process to build the software required to be embedded in the final product, since their scope of development was relatively limited. But due to the recent expansion in scale of software development, numerous problems attributable to embedded software or embedded system have come to surface. As a result, more and more attention is given now to improve the development process to address these emerging problems.

Current Status of Development Process

The following lists the possible causes of various issues arising from the expansion of scale of development and increased complexity of organizations involved in the development that are actually faced in the workplaces where embedded software is developed:

- (1) Development process is not clearly defined in the organization in charge of development.

- (2) The development process adopted by the organization is neither full-fledged (some aspects are missing) nor definitive (some parts remain ambiguous).
- (3) Only a portion of the development process is considered. The activities and tasks necessary to complete the entire development process are not fully covered.

These factors, among others, tend to adversely affect the quality of the embedded system.

Purpose of Development Process Guide for Embedded Software

This guidebook intends to improve the current conditions and promote efficient development of high-quality embedded software by providing the crucial elements of the development process in an orderly manner.

The set of information provided systematically in this guidebook combines the knowledge and expertise accumulated and put into practice by the embedded software developers in Japan with information referenced from relevant international standards (i.e.: ISO/IEC12207, 15288) established under the agreement of global interest groups based on past experiences in software development.

Moreover, this guidebook (ESPR Ver.2.0) introduces two additional elements in embedded software development process that have not been discussed in the previous version (ESPR Ver.1.0), which are as follows:

- (1) Process equivalent to the upstream phase of embedded software (system) development that partly takes into consideration of the system viewed from the standpoint of software to supplement the perception of embedded system as a product;
- (2) Set of requirements to develop a safe system and basic operations performed safely by the developed system that have been defined from the standpoint of developing an “embedded system that can be used safely and without anxiety”, using reliable source information like international standards on system safety (IEC61508) for reference.

This guidebook is intended to serve as a useful reference material for everyone involved in embedded software development.

Features of Development Process Guide for Embedded Software

Outlined below are the main features of “Development Process Guide for Embedded Software”:

- Feature 1: Organized description on more specific low-level development processes, using the international standards on system development process as the reference;
- Feature 2: Organized description on work items that should be carried out in embedded software (system) development and precautions that the developers should watch out for;
- Feature 3: Explicit description of inputs and outputs to accomplish individual tasks and concrete explanation of the work contents using expressions that are easy to understand.

1.3 Intended Users, Usages and Benefits

Intended Users / Usages

The intended users and usages of this guidebook are as described below:

Intended Users

This guidebook is intended to be used by the following users involved in embedded software development:

- (1) Managers and leaders who are responsible of managing the development projects and organizations taking part in the development, as well as examining and deciding on the actual processes and phases to be executed in each development project;
- (2) Members of the organizations taking part in the embedded software development who are in charge of defining the standards and basic concepts of the development process applied by the participating organizations and groups, and supporting the efforts to put these standards and concepts into effect;
- (3) Members of the support groups belonging to the organizations taking part in the embedded software development who are indirectly supporting the software development in areas, such as, quality assurance.

Intended Usages

This guidebook aims at providing the overall picture of the types of work required in software development in a holistic, orderly manner, with main emphasis placed on embedded software development.

This guidebook is intended to be used by those who are interested in organizing the development processes that best suit the interests of individual organizations and groups, or the needs of individual projects.

Outlined below are some of the typical situations when there will be a foreseeable need to organize the processes of the development process:

- (1) When the software development process is not defined, and the standards for development process carried out by the organizations and groups need to be newly established;
- (2) When there is a need to redefine the standard development process already put into practice at the organizational or group level due to the gap and/or conflicts arising between the existing process and the actual operations at the development site;
- (3) When there is a need to define the standard development processes newly because the conventional development processes are inadequate or outdated.

Benefits

The users of this guidebook can expect to gain the following benefits by organizing the development processes to be carried out by the relevant organizations and groups.

Benefits of Organizing the Development Process

By organizing the development processes to be carried out by the relevant organizations and groups involved in embedded software development, the members of these organizations and groups will be able to:

- (1) Check that all the activities (tasks / sub-tasks) that should be performed are carried out without fail, and reexamine the activities (tasks / sub-tasks) that may be found to be unnecessary;
- (2) Establish the framework for performing elaborate work to achieve the functionalities demanded by the market and/or required by the clients, and ensure that they are highly reliable and of high quality;
- (3) Ensure the division of labor and collaboration between the different teams allocated with specific jobs in case of large-scaled development projects that require development members of various disciplines to be involved and work assignments to be sub-divided among them;
- (4) Clearly define the information (e.g.: information regarding the work products, deliverables, contents of activities (tasks / sub-tasks) that are outsourced, names of these activities (tasks / sub-tasks), etc) to be provided to the subcontractors in case of projects that outsource a part of the development work to subcontractors.

Benefits of Using This Document

But using this guidebook as a reference material for organizing the development processes, the users will be able to:

- (1) Sort out the activities (tasks / sub-tasks) that need to be performed respectively by the relevant organizations and groups, and organize them as the development process to be carried out respectively by these organizations and groups, by referring to the relevant international standards and knowledge accumulated over the years by those engaged in software development process;
- (2) Review the outcome of individual activities (tasks / sub-tasks) and the contents of the deliverables, by referring to the document template samples provided in this guidebook;
- (3) Prepare for the embedded software development more carefully by referring to the contents of this guidebook (especially, the information provided under Precaution) while defining the development processes that best suit the interests of the relevant organizations and groups.

1.4 Structure of Development Process Guide

Process Structure

This guidebook classifies the different types of work for embedded software development into four layers, which are: “process”, “activity”, “task” and “sub-task” (See Fig. 1.1).

* From hereafter, “development process” will be shortened to “process”, unless otherwise stated.

Process

The various types of work that need to be performed in order to proceed with the development of embedded software have been largely divided into the following four processes (or work clusters):

(1) System Engineering Process

This process mainly consists of the various types of work for developing the embedded system built on the basis of the software that would be embedded.

(2) Software Engineering Process

This process mainly consists of the various types of work related to the actual software development.

(3) Safety Engineering Process

This process consists of the various types of work that should be carried out to develop an embedded system that can be used safely and without anxiety.

(4) Support Process

This process consists of the various types of work for providing a wide range of support that becomes necessary in the course of the development (e.g.: documentation).

Activity

The various types of work that need to be performed in each of the abovementioned processes have been grouped into a more specific work cluster called “activity”.

For example, Software Engineering Process is composed of the following activities:

- Software Requirements Definition
- Software Architectural Design
- Software Detailed Design
- Implementation & Unit Testing
- Software Integration Testing
- Comprehensive Software Testing

Task

The various types of specific work required to execute the individual activities and achieve the pre-defined objectives of these activities respectively have been grouped into a lower-levelled work cluster called “task”.

For example, the activity named “Software Requirements Definition” is composed of the following two tasks:

- Creating the Software Requirements Specifications;
- Reviewing the Software Requirements Specifications.

Sub-task

The specific portions of any given tasks that need to be performed (often step by step) to complete the individual tasks have been grouped into a work cluster of the lowest level called “sub-task”.

For example, the task named “Creating the Software Requirements Specifications” is composed of the following five sub-tasks:

- Identifying the Constraints
- Clarifying the Functional Software Requirements
- Clarifying the Non-functional Software Requirements
- Prioritizing the Requirements
- Creating the Software Requirements Specifications

Wherever applicable, this guidebook provides the detailed description of what should be done to complete each sub-task and the points to keep in mind from the standpoint of embedded software when engaging in these sub-tasks.

Regarding the Time Concept

The various types of work (processes, activities, tasks, sub-tasks) described in this guidebook are basically not arranged in chronological order.

Therefore, in order to determine how the embedded software development of any given project should actually proceed, there is a need to “select the types of work required for the development” and “design the development process phases” by using this guidebook as a reference (See Part 3 for the details.).

Selecting the Types of Work Required for the Development

In determining what types of work are required for the development, the nature and characteristic features of the system and software that need to be developed and/or the special conditions of the development projects and the organizations involved in the development must be taken into account. The orderly set of

information provided in this guidebook is intended to cover the full range of work deemed necessary in developing embedded software newly. Depending on the characteristics of the software to be developed, some of the work provided in this guidebook may either be omitted because they are not necessary or fine-tuned to best fit the given conditions of the software.

Likewise, in case of a project where the main objective of the development is to remodel or reuse the existing embedded software, the project team would need to select only the specific types of work required for the given project from the full set of information on processes, activities, tasks and sub-tasks provided in this guidebook.

Designing the Development Process Phases

In planning the actual development project, there is a need to predict the time and period taken from the very start of the development until the final product is shipped out from the manufacturing site. The act of In planning the actual development project, there is a need to predict the time and period taken from the very start of the development until the final product is shipped out from the manufacturing site. The act of concurrency of the pre-selected set of work.

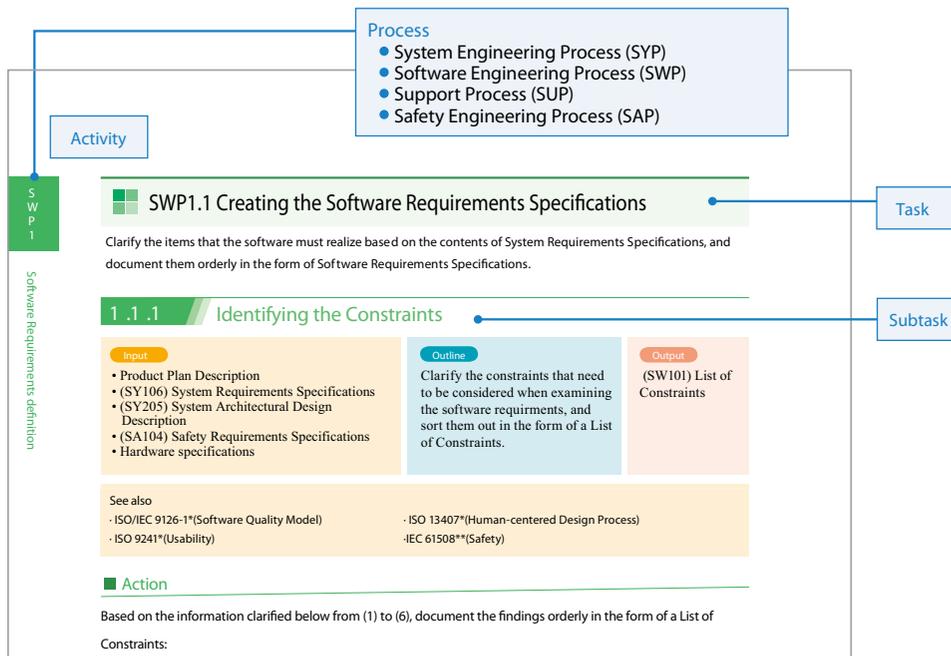


Figure 1.1 Process Structure

Structure of Embedded System

There are many ways of perceiving the term “embedded system”. To some, it may mean a “single equipment”. Some others may say that it is an “aggregate of multiple equipment”. There are even some that support the extended interpretation that the equipment integrated with the server is also an inclusive part of embedded system. In this guidebook, the embedded system is simply defined as a “single equipment”.

At the core of each embedded system are one or more functionalities that each system is expected to provide. In this guidebook, each of these functionalities provided by the system is referred to as the “functional block”, and the embedded software used to make each system function is broken down hierarchically into “functional unit” and “program unit” (See Fig. 1.2.).

Embedded system

Embedded system is a single equipment that has one or more functionalities. Take, for example, a multi-functional system that function as a telephone, facsimile, copier, scanner and printer. If each of these functionalities is provided by a separate equipment, each equipment is considered as an embedded system. On the other hand, if all these functionalities are integrated and inseparable, the entire multi-functional machine itself is considered as one embedded system. Embedded system is composed of one or more embedded software and one or more hardware.

Functional block

Normally, an embedded system provides multiple functionalities. A combination of a specific hardware and specific embedded software, which forms one functional block, is necessary to realize each of these functionalities.

Embedded software

Embedded software is an entity that constitutes a part of the embedded system, and is composed of a specific combination of functional units.

Functional unit

Functional unit is an entity that constitutes a part of the embedded software, and is composed of a specific combination of program units.

Program unit

Program unit is an entity of the lowest level (e.g.: compile or test unit) that constitutes a part of the functional unit.

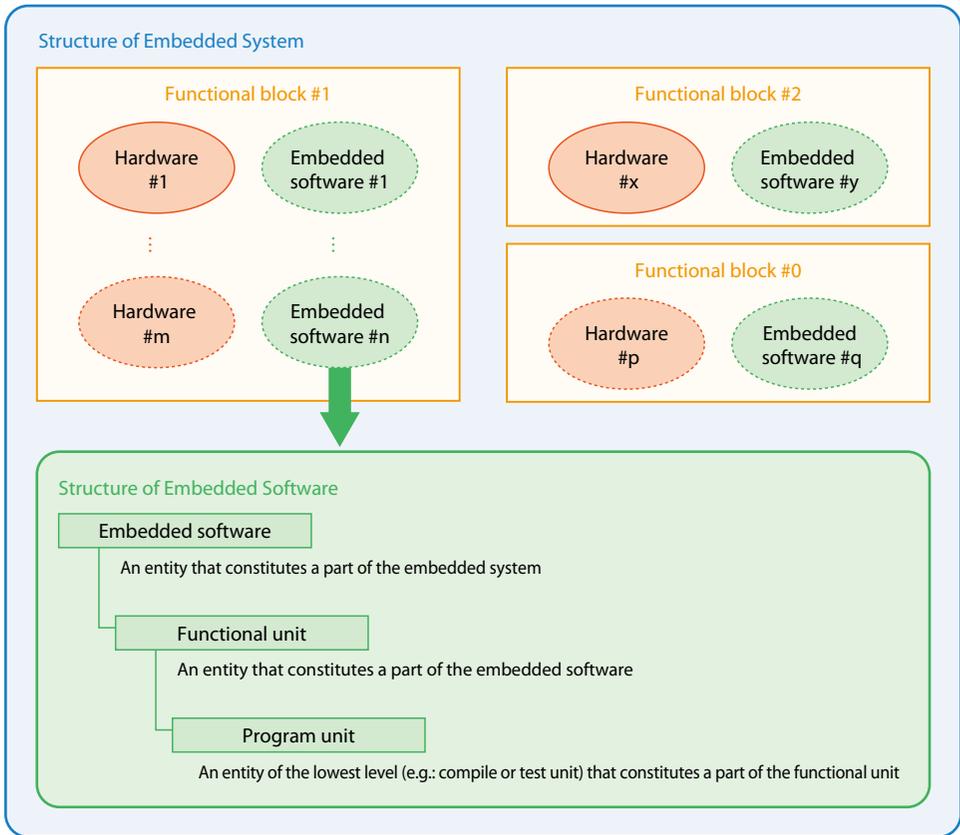


Figure 1.2 Structure of Embedded System

1.5 Instructions and Directions for Using This Guidebook

Treating This Guidebook as Reference Document

This guidebook provides an orderly set of information on the various types of work required for embedded software development.

In general, international standards (like ISO) and local standards (like JIS) are established, by nature, as mandatory conventions and practices intended to be employed and enforced as stated, to ensure that the activities pre-defined within the scope of the objectives of each standard are performed through the prescribed disciplined uniform approach, and normally examined together with the means for checking whether the applicants are fully compliant or not (to grant certification, authentication, etc). In this context, the standards like ISO/IEC12207 and 15288 that are used as the reference when formulating this guidebook are no exceptions.

However, this guidebook itself does not follow the same principle. As one part of the name of this guidebook “ESPR: Embedded System development Process Reference...” suggests, this document is merely intended to provide an orderly set of referential information on the common practices in embedded software development, and is not written with the expectation that the prescribed information, when followed strictly, will meet the compliance requirements for rewarding certification, authentication, or any other forms of official accreditation.

Regarding the Process Definition

In keeping with the aforesaid purpose of what this guidebook intends to achieve, this document expects the readers to view both the general and detailed information on processes, activities, tasks and sub-tasks provided here as merely reference information. Therefore, even when certain usages of the provided information are recommended as preferable practices in this guidebook, they are neither mandatory nor binding.

Regarding the Document Template Samples

This guidebook provides samples of document templates to help the readers sort out the outcome of the tasks and sub-tasks performed in their actual development projects. These document templates are also provided only for reference, and the suggested usages are neither mandatory nor binding.

Important Reminders on Terminology and Conceptual Framework

The editorial policy of this guidebook is “to express the concepts using the terminology that is widely acceptable to the engineers at the development sites”. The slight differences between the terminology used in existing international standards and this guidebook to define the processes and express the concepts of the processes and activities are the result of deliberate efforts to abide by the aforesaid editorial policy as much as possible.

Nevertheless, to avoid unnecessary confusion, a table that maps which portion of the text in this guidebook is corresponding to the existing international standards or other reliable references has been added as one of the appendices of this guidebook.

Important Reminders on Terminology

Some of the existing international standards are already translated and officially published in Japanese (e.g.: ISO/IEC12207 → JIS X0160).

This guidebook have used both the original English texts of international standards and the translated versions published from Japan Industrial Standards Committee (JISC) as references, but some terms and expressions have been deliberately changed to those that the engineers at the actual development sites in Japan feel more familiar, based on their opinions.

Important Reminders on the Conceptual Framework of the Processes Dealt in This Guidebook

This guidebook has used the following four frameworks to group the various types of work deemed necessary in embedded software development:

- System Engineering Process;
- Software Engineering Process;
- Safety Engineering Process; and
- Support Process.

While using the process categories prescribed in existing international standards as references, this guidebook has adopted the above four frameworks to focus on describing the types of work directly related to embedded system building (=System Engineering Process), the types of work directly related to embedded software development (=Software Engineering Process), the types of work to ensure that the system can be used safely and without anxiety (=Safety Engineering Process), and the types of work for supporting the activities, tasks and sub-tasks carried out in these processes (=Support Process) (See Fig. 1.3.). As regards the support process in particular, the scope of work extends diversely from development management to

outsourcing. This guidebook (ESPR Ver.2.0) provides an orderly set of information to describe the activities that should be performed in the support process, as shown in the next diagram (Fig. 1-3).

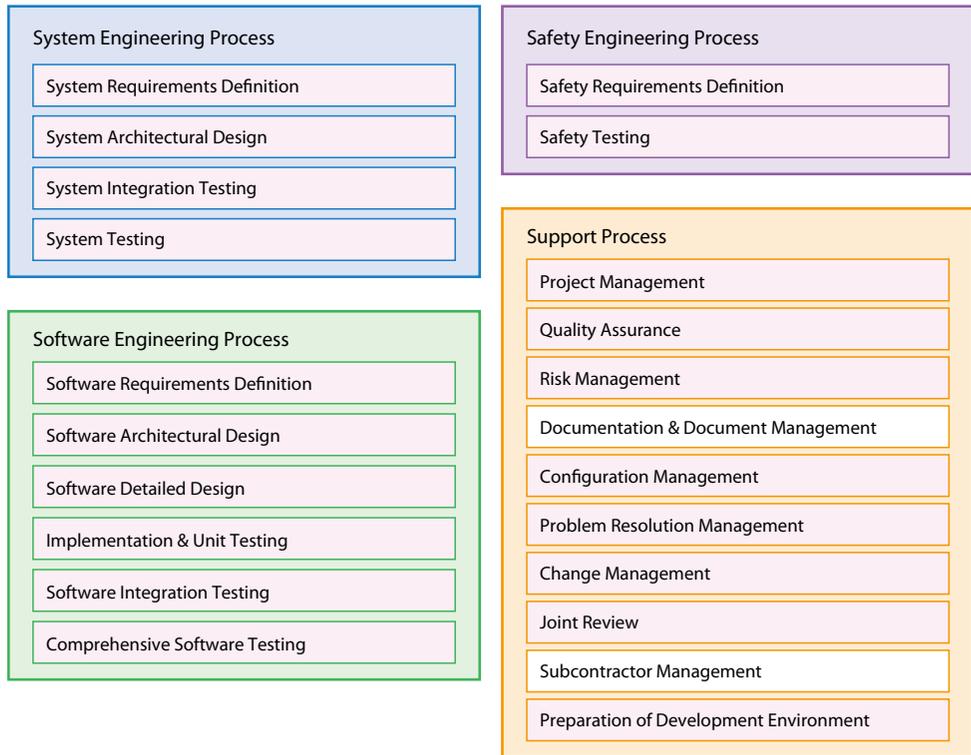


Figure 1.3 Processes and Types of Work Defined in This Guidebook

1.6 Related Standards

Standards on Development Process

As the standards on system and software development processes, various international standards are established, including ISO/IEC12207 and ISO/IEC15288.

The relevant portions of these standards are inserted in this guidebook, wherever applicable, under the heading “Related Standards”.

ISO/IEC12207: 1995 Software Life Cycle Processes

This is an international standard that prescribes the processes in software product development in a systematic manner. In Japan, the translated version of this standard titled “JIS X0160 Software Life Cycle Processes” is published from Japan Industrial Standards Committee (JISC).

This standard has been established for the purpose of sorting out the various types of work pertaining to software development and to standardize their names, among others. Although this standard does not address any specific type of software, one of the key underlying themes studied in the course of establishing this standard was “the development of an enterprise system, especially in the case where a two-party agreement, whether informal or legally binding, exists between the acquirer and the supplier”.

ISO/IEC15288: 2002 System Life Cycle Processes

This is an international standard that addresses the systems that provides services through the application of multiple self-contained software and/or hardware, and prescribes the various types of work required in the life cycle of such systems from the standpoint of development process. In Japan, the translated version of this standard titled “JIS X0170 System Life Cycle Processes” is published from Japan Industrial Standards Committee (JISC).

Conceptually, this standard can be perceived as the extended version of ISO/IEC12207 to include the descriptions on system life cycle processes in its coverage. However, the descriptions on the systems it is addressing are not particularly focused on embedded system.

Standards on Safety and Security

Various standards pertaining to system and product safety and security are established (international standards, national standards, laws and regulations). Among them, international standards like IEC61508 that have been established to promote the functional safety of electric and electronic systems that use built-in computers are recently drawing attention.

IEC61508: Functional safety of electrical / electronic /programmable electronic safety-related systems

This is the international standard that prescribes the various types of work that should be carried out in the course of development of highly safe systems, including those that contain computing devices (e.g. plant control systems, automotive systems, etc), to ensure that the required high level of safety of such systems is achieved. This standard does not address any specific product domain and is strictly limited to providing the general requirements that need to be met to achieve the required level of safety. Various industries have started establishing their own domain-specific safety standards, using this standard as the key source of

reference.

Standards for Improving the Usability

One of the noteworthy features of embedded systems developed recently is their functional diversity. Numerous standards are established with the aim to enhance the usability of such multi-functional systems, including ISO13407 and ISO9241. This guidebook makes references to ISO13407, as the standard that is also related to development process.

ISO 13407:1999 Human-centered design processes for interactive systems

This is an international standard that prescribes the disciplined uniform approach to incorporate the views of the so-called “human-centered design”, which is the design concept that takes into consideration of the human elements of the system users. In this standard, human-centered design is primarily described in the context of the following three aspects: (1) principles of human-centered design; (2) planning of human-centered design process; and (3) human-centered design activities.

Standards on Software Quality

ISO 9126s: Software Engineering □ Product Quality

This is an international standard that provides conceptual descriptions to define what the quality of software product actually means. It has identified six representative quality characteristics, which are: functionality; reliability; usability, efficiency; maintainability; and portability, and provides detailed definition of each characteristic.

Moreover, this standard also provides information on how these quality characteristics can be measured and visualized.

Relationship Between International Standards and This Guidebook

In formulating this guidebook, the international standards mentioned above that provide orderly set of information on development process, safety, usability and quality, including their basic concepts, have been used as reference sources. In general, many international standards do not give specific binding details on how they must be applied. Likewise, the contents of the particular international standards referenced by this guidebook are also limited to relatively abstract descriptions and do not get into deep details. For this reason, this guidebook has attempted to interpret these international standards to provide more concrete information about embedded software development.

Part 2

Technical Section

2.1 Overall Structure	18
2.2 Process Definition Documents	21
2.3 Document Template Samples	163

2.1 Overall Structure

The development process defined in this guidebook is largely divided into the following four processes. The overall structure of these processes is as shown on the next page.

- System Engineering Process (SYP)

Consists of the orderly description of the activities (and tasks) to meet the system requirements particularly for the embedded system that operates by the workings of the software that is embedded, as well as to verify the system behavior, among others

- Software Engineering Process (SWP)

Consists of the orderly description of the activities (and tasks) directly engaged in software development that ranges from software requirements definition to comprehensive software testing

- Safety Engineering Process (SAP)

Consists of the orderly description of the activities (and tasks) to build an embedded system that can be used safely and without anxiety

- Support Process (SUP)

Consists of the orderly description of mainly the activities (and tasks) to support and indirectly engage in the facilitation of software development

Among the above four processes, descriptions on SYP and SAP have been newly introduced in this guidebook, “ESPR Ver.2.0: Development Process Guide for Embedded Software”, in addition to SWP and SUP that are covered in the previous version, ESPR Ver.1. The relationship between the V-model and the processes/ activities is shown in Fig. 2.1.

- Terminology

Safety, reliability, risk

Safety : The expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered. (JIS X 0134: Information technology – System and software integrity levels)

Reliability : The ability of a functional unit to perform a required function under stated conditions for a stated period of time. (JIS X 0014: Information technology – Vocabulary – Reliability, maintainability and availability)

Risk : Possibility of suffering a loss as a result of partially or entirely losing something that has value, like assets, infrastructure and privacy.

SYP :
System Engineering Process

SYP1 System Requirements Definition	22
1.1 Creating the System Requirements Specifications	
1.2 Reviewing the System Requirements Specifications	
SYP2 System Architectural Design	32
2.1 Creating the System Architectural Design Description	
2.2 Reviewing the System Architectural Design	
2.3 Jointly Reviewing the System Architectural Design	
SYP3 System Integration Testing	43
3.1 Preparing for System Integration Test	
3.2 Conducting the System Integration Test	
3.3 Reviewing the System Integration Test Results	
SYP4 System Testing	51
4.1 Preparing for System Test	
4.2 Conducting the System Test	
4.3 Reviewing the System Test Results	
4.4 Confirming the Completion of System Development	

SAP :
Safety Engineering Process

SAP1 Safety Requirements Definition	132
1.1 Creating the Safety Requirements Specifications	
1.2 Reviewing the Safety Requirements Specifications	
SAP2 Safety Testing	141
2.1 Preparing for Safety Test	
2.2 Conducting the Safety Test	
2.3 Reviewing the Safety Test Results	

SWP :
Software Engineering Process

SWP1 Software Requirements Definition.....	63
1.1 Creating the Software Requirements Specifications	
1.2 Reviewing the Software Requirements Specifications	
SWP2 Software Architectural Design.....	76
2.1 Creating the Software Architectural Design Description	
2.2 Reviewing the Software Architectural Design	
2.3 Jointly Reviewing the Software Architectural Design	
SWP3 Software Detailed Design	89
3.1 Creating the Functional Unit Detailed Design Description	
3.2 Reviewing the Software Detailed Design	
3.3 Checking the Consistency with Hardware Specifications	
SWP4 Implementation & Unit Testing.....	99
4.1 Preparing for Implementation and Unit Test	
4.2 Conducting the Implementation and Unit Test	
4.3 Reviewing the Implementation and Unit Test Results	
SWP5 Software Integration Testing	109
5.1 Preparing for Software Integration Test	
5.2 Conducting the Software Integration Test	
5.3 Reviewing the Software Integration Test Results	
SWP6 Comprehensive Software Testing	120
6.1 Preparing for Comprehensive Software Test	
6.2 Conducting the Comprehensive Software Test	
6.3 Reviewing the Comprehensive Software Test Results	
6.4 Confirming the Completion of Software Development	

SUP :
Support Process

SUP1 Project Management	149
1.1 Creating the Project Plan Description	
1.2 Understanding the Project Execution Status	
1.3 Controlling the Project	
1.4 Creating the Project Completion Report	
SUP2 Quality Assurance	152
2.1 Defining the Quality Objectives	
2.2 Establishing the Quality Assurance Method	
2.3 Controlling the Quality Based on Quality Visualization	
SUP3 Risk Management.....	154
3.1 Identifying and Understanding the Risks	
3.2 Monitoring the Risks	
3.3 Determining and Executing the Risk Treatments	
SUP4 Documentation & Document Management	
SUP5 Configuration management	156
5.1 Understanding the Objects of Configuration Management	
5.2 Managing the Configuration Management / Change Management History	
SUP6 Problem Resolution Management	158
6.1 Recording the Problems and Analyzing the Causes	
6.2 Analyzing the Impact and Devising the Acceptable Solution	
6.3 Implementing the Acceptable Solution	
6.4 Tracking the Implemented Solution	
SUP7 Change Management	160
7.1 Recording the Information on Change Requests	
7.2 Analyzing the Impact of Changes	
7.3 Devising and Executing the Change Plan	
7.4 Reviewing the Outcome of the Changes Made	
SUP8 Joint Review	161
8.1 Preparing for the Review	
8.2 Carrying Out the Review	
8.3 Acknowledging and Following Up on Matters That Have Been Reviewed	
SUP9 Subcontractor Management	
SUP10 Preparation of Development Environment ...	162
10.1 Devising the Development Environment Preparation Plan	
10.2 Building the Development Environment	
10.3 Maintaining the Development Environment	

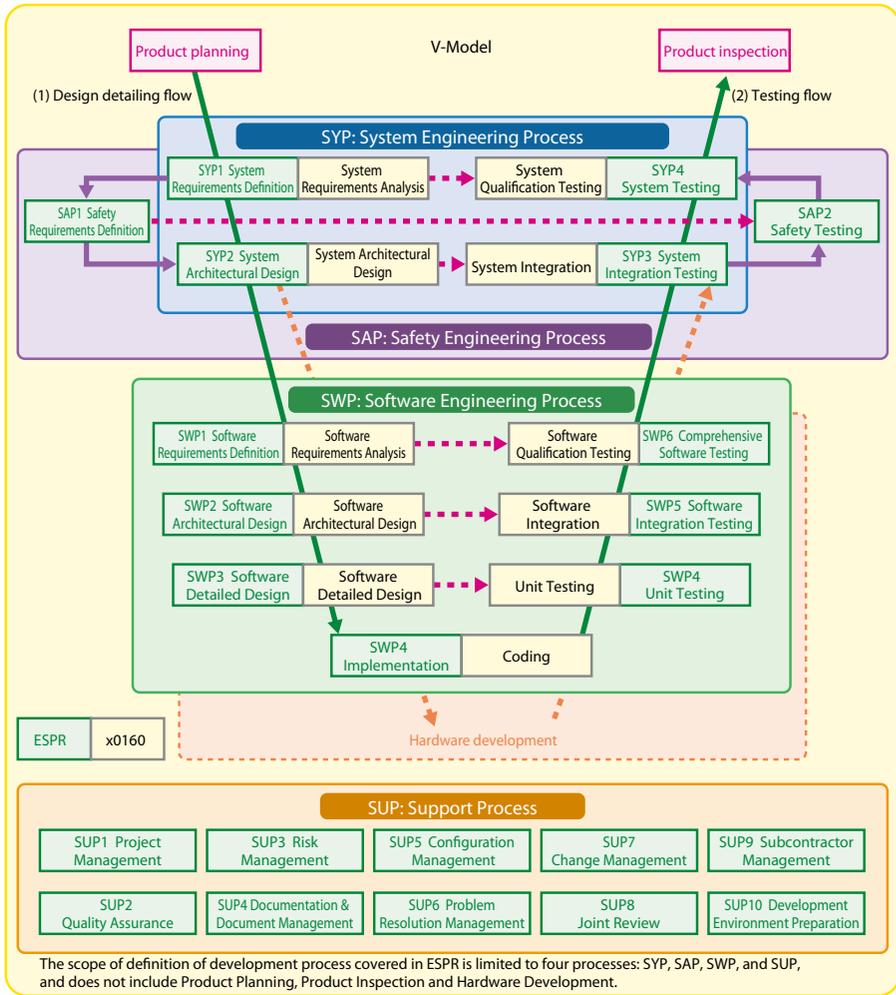


Figure 2.1 V-Model and Development Process

2.2 Process Definition Documents

SYP : System Engineering Process

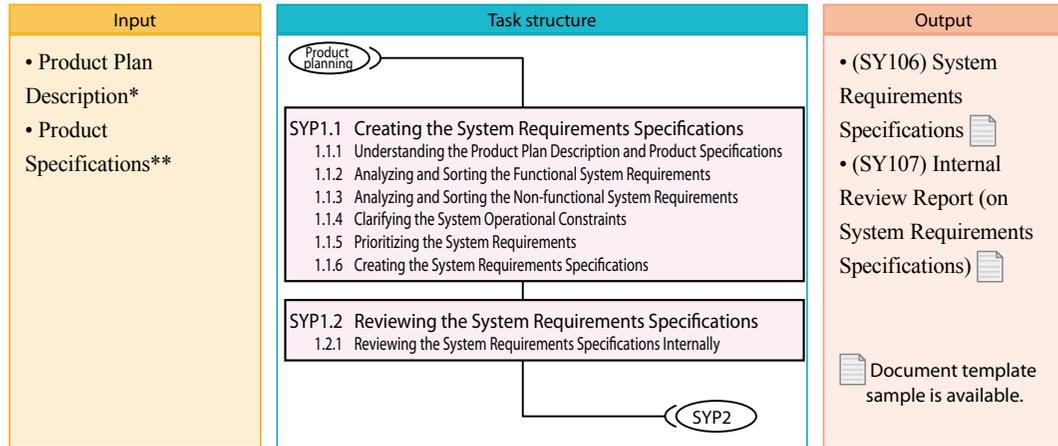
Among the various types of work pertaining to embedded system development, the activities and tasks ranging from the System Requirements Definition for the product system that includes both the hardware and software to System Testing are defined in this process.

The following activities are included in this process:

ID	Activity	Outline of the activity	Comprising tasks
SYP1	System Requirements Definition	Clarify the requirements that the system must fulfill to realize the targeted product.	SYP1.1 Creating the System Requirements Specifications SYP1.2 Reviewing the System Requirements Specifications
SYP2	System Architectural Design	Examine how to actualize the embedded system to be developed, including the division of roles of hardware and software.	SYP2.1 Creating the System Architectural Design Description SYP2.2 Reviewing the System Architectural Design SYP2.3 Jointly Reviewing the System Architectural Design
SYP3	System Integration Testing	Confirm that the functional blocks operate when the hardware and software that structure the system are integrated.	SYP3.1 Preparing for System Integration Test SYP3.2 Conducting the System Integration Test SYP3.3 Reviewing the System Integration Test Results
SYP4	System Testing	Confirm that the system requirements are fully met.	SYP4.1 Preparing for System Test SYP4.2 Conducting the System Test SYP4.3 Reviewing the System Test Results SYP4.4 Confirming the Completion of System Development

SYP1 System Requirements Definition

Clarify the requirements that the system must fulfill to realize the targeted product.



Description

In this activity:

- (1.1.1) Grasp the contents of product requirement specifications, based on the contents of Product Plan Description and Product Specifications;
- (1.1.2) Analyze and clarify the functional requirements of the system;
- (1.1.3) Also clarify the non-functional requirements of the system;
- (1.1.4) Clarify the operational conditions and constraints of the product system by taking the actual operating environment and other relevant factors into consideration;
- (1.1.5) Prioritize the individual functional and non-functional requirements by taking account of the constraints in actual system development, including the development period, resources and operating environment of the product;
- (1.1.6) Create the System Requirements Specifications by organizing the information gained from the above tasks in an orderly manner;
- (1.2.1) Review the created System Requirements Specifications, based on the pre-defined check points, and document the outcome of this review orderly in the form of an Internal Review Report.

Consideration

- ▶ Keep in mind the following points as the prerequisites for commencing the activity to define the system requirements:
 - Product planning: Product strategies (such as, the

• Terminology

* Product Plan Description : A document that contains catalog-level information on the product, and description about product strategies (such as, the end users' needs) that are clearly defined.

** Product Specifications : A document that contains user guide-level information on the product, and description about the services provided by the product that are clearly defined.

- end users' needs) are clearly defined;
- Schedule: The overall schedule is already fixed (product release/launch date, milestones shared with external stakeholders, etc).
- ▶ In case of embedded system, place particular importance on analyzing the external environment (such as, the system's operating environment) and the functional analysis on how the system is capable of responding to abnormalities;
- ▶ Place importance on analyzing not only the functional aspects but also the non-functional

aspects of the system (such as, performance and maintainability);

- ▶ Examine the requirements also from the standpoint of the external system that operates closely or in conjunction with the system that is going to be developed;
- ▶ Decide on the requirements specifications by also taking account of the functionalities that the product is expected to continue providing in long term.

<Reference> Techniques and Tools

- ▶ Scenario analysis (use case diagrams, activity diagrams, etc)

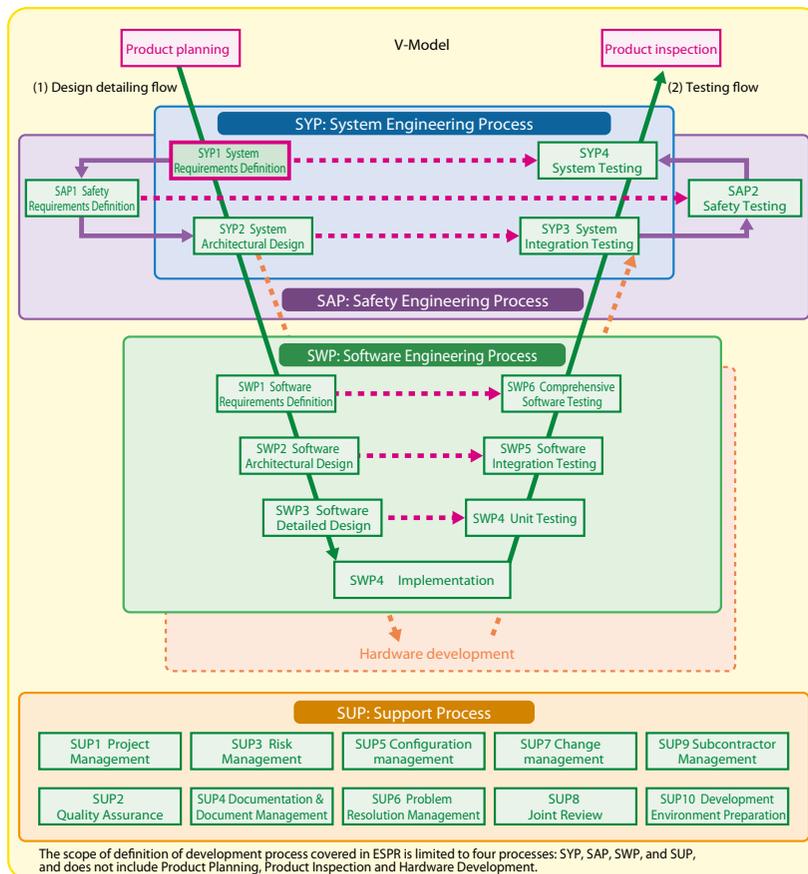


Figure 2.2 V-Model and Development Process (SYP1 System Requirements Definition)



SYP 1.1 Creating the System Requirements Specifications

Based on the Product Plan Description and Product Specifications, clarify the requirements that the system must fulfill to realize the targeted product, and create a document called System Requirements Specifications.

1.1.1 Understanding the Product Plan Description and Product Specifications

Input

- Product Plan Description
- Product Specifications

Outline

Grasp and confirm the contents stated in the Product Plan Description and Product Specifications prepared by Product Planning Department or other relevant organizations.

Output

(Confirmation Note for Product Plan Description and Product Specifications)

Action

- (1) Confirm the following points stated in the Product Plan Description:
- ▶ Product outline and characteristics, functional differences from existing products, etc;
 - ▶ Product market, time to market (launch schedule);
 - ▶ Positioning and level of importance of the product based on medium- to long-term product strategies;
 - ▶ Standards, conventions, laws and regulations

related to the product.

- (2) Confirm the following points stated in the Product Specifications:
- ▶ Product vision and concept;
 - ▶ Intended users and system requirements;
 - ▶ Situations and context in which the product is used;
 - ▶ Constraints in realizing the product.

Precaution

- Listed below are matters that need to be careful of when confirming the contents of the Product Plan Description:
 - ▶ Product Plan Description is a document normally prepared by the product planning or sales department/division to provide the overview of the product to be developed. Keep in mind that it does not necessarily state all the information pointed above.
 - ▶ The information stated in the Product Plan

Description may be revised due to various factors, including the changes in the intended users and market trends. Give attention to when the Product Plan Description was created.

- Listed below are matters that need to be careful of when confirming the contents of the Product Specifications:
 - ▶ Product vision and concept:
 - Strategic positioning of the product;
 - Superiority over similar products developed by

• Terminology

* Context : Context is something written or spoken that immediately precedes or follows a word or passage, background information, or an interrelated condition, situation or circumstance that helps clarify an idea or its meaning. In this guidebook, this term is used, for example in particular section, to mean the purpose, background or strategy of the embedded system to be developed.

- competitors;
- Concept of not only the functionalities and services provided by the product, but also the concept in terms of quality, cost and delivery (QCD) perceived from the standpoint of stakeholders engaged in product development;
- ▶ Intended users and system requirement:
 - Age, gender, nationality, occupation and place of residence of the intended users, among others
 - Their expected frequency of use of the targeted product, and whether they have past experience using the existing product or not;
 - Requirements from the manufacturer (vendor);
 - Typical user requirements.
- ▶ Situations and context in which the product is used:
 - Situation or condition when the intended users use the product in:
 - 1) Normal state;
 - 2) Unexpected state.
 - Elements related to product behavior and

the users' operation of the product in typical situations and context

- Scenarios for using the product, including contextual information and users' operation (use case diagram, activity diagrams, etc)
- ▶ Constraints in realizing the product:
 - Length of time (development period) and budget (development costs) that can be allocated for product development;
 - Serviceable life of the product (durable years), and product life cycle;
 - Expected product quality (reliability, safety, usability, etc);
 - Peripheral systems and hardware used as the prerequisites for using the product;
 - Constraints on the continuity from the existing product specifications;
 - Constraints on reusing the existing system;
 - Constraints due to security and environmental issues;
 - Product (sales) price

1.1.2 Analyzing and Sorting the Functional System Requirements

Input

- Product Plan Description
- Product Specifications

Outline

Based on the information described in the Product Specifications, analyze and sort out the functional requirements of the system from the standpoint of those developing a system that enables the product to function as required.

Output

- (SY101) List of Functional System Requirements
- (SY102) System Functionality-Operation Matrix

Action

- (1) Analyze the requirements stated in the Product Specifications, and identify the functional requirements that the system must meet to realize the product.
- (2) Perform use case analysis that takes account of the system users and when the system is going to be used, and include the analytical results in the examination process to identify the functional requirements.
- (3) After identifying the functional requirements, do the following:
 - ▶ Analyze the relationship between functionalities (such as, the sequential relationship and concurrency of system operations) and use the findings to create an orderly matrix of system functionalities and operations;
 - ▶ Examine whether any of the functionalities of the existing systems can be carried over or reused in the current system, by considering the likelihood of current and future reuse of existing systems;
 - ▶ Clarify which data will be associated to functionalities that involve data processing.
- (4) In addition, gain a good knowledge about how the systems (and/or sub-systems) are linked via the network or bus, and the external interfaces for the linkage.
 - ▶ Also have a clear understanding of the communication protocol, configuration and contents of data exchanged via the network, among others.
- (5) In case the system comes together with user interface, the display and operation items, as well as the devices for controlling the user interface should also be clearly defined.

Precaution

- ▶ Be able to easily map which functional system requirements correspond with which requirements stated in the Product Specifications and/or Product Plan Description.
- ▶ During the use case analysis, create use case scenarios and diagrams, as well as an activity diagram, among others.
- ▶ The functional system requirements stated in the Product Specifications may not be sufficient to fulfill all the functionalities the system is required to provide for the product. Should this be the case, perform additional analysis consisting of the following items to define the full-fledged functional system requirements in a systematic order:
 - (1) Sort and analyze the necessary functionalities from the standpoint of service;
 - (2) Based on the perceived granularity of the

respective functionalities, sub-divide the functionalities and sort them out in hierarchical order;(3) Map the linkage and/or co-relation between the respective functional items

- ▶ To analyze and sort the functionalities, also look at the relationship between the data used for the individual functionality, the product and the externals, and examine the co-relation between

the functionalities and the chronological state transition of the system and the functionalities.

- ▶ If the system to be developed is a multi-CPU type where multiple CPUs are provided to achieve the required functionalities, the specifications on the internal linkage operation should also be examined.

1.1.3 Analyzing and Sorting the Non-functional System Requirements

Input

- Product Plan Description
- Product Specifications

Outline

Based on the information described in the Product Specifications, analyze and sort out the non-functional requirements of the system from the standpoint of those developing a system that enables the product to function as required.

Output

- (SY103) List of Non-functional System Requirements

Action

- (1) Analyze the requirements stated in the Product Specifications, and identify the non-functional requirements that the system must meet to realize the product.
 - Processing time and response time of individual functionality
 - Easiness to operate the user interface
 - Method of performing corrective actions to defects that occur after the product is launched in the market (such as, remote maintenance)
- (2) Note that non-functional system requirements include the "reliability", "efficiency", "maintainability", "portability" and "usability" of the system.

(Example)

 - Tolerable frequency and severity of system errors
- (3) If the system is assumed to be used by connecting it to the network, also clarify the security-related requirements.

Precaution

- ▶ For non-functional requirements that can be specified with concrete numerical targets, explicitly state the target values.

(Example) Response time, etc
- ▶ For items related to system safety, also examine the impact and frequency of the problematic state caused by the system's functional defects.
- ▶ Non-functional requirements are often not stated explicitly in Product Plan Description and Product Specifications. Therefore, they frequently need to be identified by referring to the explicit descriptions and contextual information on when the system is intended to be used.

Related processes: SAP1 Safety Requirements Definition

1.1.4

Clarifying the System Operational Constraints

Input

- Product Plan Description
- Product Specifications

Outline

Clarify the operational conditions and constraints of the system used to realize the product by taking the actual operating environment and other relevant factors into consideration.

Output

- (SY104) List of System Operational Constraintst

■ Action

- (1) Grasp the system's operating environment.
 - ▶ Clarify the operational conditions that are attributable to, such as, the location where the system is installed.
 - ▶ Clarify the range of data and values inputted into the system via a sensor or other means.

(Example)

“Specifications for withstanding cold environments” defined for automobiles
- (2) Clarify the size and properties of data processed by the system and the timing of data input/output.
- (3) Clarify the legal restrictions incidental to system use and other constraints related to social conventions and business practices.
- (4) Clarify the dimensions and layout of the product, and any other information related to system mounting. **Safety**
- (5) Clarify how and to what extent the system is related to intellectual properties and proprietary technologies of other companies.
- (6) Clarify also the constraints regarding the hardware platform (MPU, LSI, etc) that uses the system.

■ Precaution

- ▶ Normally, a system is developed, based on the assumption of the operating environment in which it is going to be used. A system developed in this way is often prone to system failures when it is actually used in unexpected environmental conditions that have not been assumed during the development. Therefore, the possible usages of the system should be assumed as broadly as possible when analyzing the system's operating conditions.

Safety : Work related to safety

1.1.5 Prioritizing the System Requirements

Input

- Product Plan Description
- Product Specifications
- (SY101) List of Functional System Requirements
- (SY103) List of Non-functional System Requirements
- (SY104) List of System Operational Constraints

Outline

Prioritize the system requirements by taking account of the critical factors like constraints that affect the efforts to develop a system that fully meets the defined requirements.

Output

- (SY105) Prioritized List of System Requirements

Action

- (1) Prioritize the system requirements by taking account of the items listed in (SY101), (SY103), and (SY104).
- (2) In prioritizing the items that need to be addressed to develop a system that fully meets the defined requirements, weigh the importance of each of these items by referring to the estimated cost and development period that would be necessary to fulfill the requirements (both functional and non-functional), product evaluation (on the value of the product when the system requirements are fully met, including the value perceived by the users), and feasibility study.
- (3) After examining all the conceivable elements that should be considered for prioritization, the system requirements should be classified preferably into three priority levels: High / Mid / Low.

Precaution

- ▶ In some cases, the following analytical methodologies may prove to be effective for system requirements prioritization:
 - Quality Function Deployment (QFD)
 - Analytical Hierarchy Process (AHP) for relative comparison of the requirements.
- ▶ Moreover, there are cases when the system requirements may conflict with each other. In such cases, additional examinations would become necessary, including the trade-off analysis.
- ▶ There are also cases when some system requirements may not be feasible. In such cases, careful examination to distinguish the feasible ones from others would also be an important step in narrowing down the system requirements.
- ▶ The prioritization of system requirements may have to be repeated in some cases when coordination with the customers and other stakeholders become necessary.
- ▶ The prioritization of system requirements should also be examined with a medium- to long-term view that takes account of the possibility of launching variants or new versions of the product in the future.

1.1.6

Creating the System Requirements Specifications

Input

- Product Plan Description
- Product Specifications
- (SY101) List of Functional System Requirements
- (SY102) System Functionality-Operation Matrix
- (SY103) List of Non-functional System Requirements
- (SY104) List of System Operational Constraints
- (SY105) Prioritized List of System Requirements

Outline

Sort out the system requirements and describe them orderly in the form of a document called System Requirements Specifications.

Output

- (SY106) System Requirements Specifications 

 Document template sample is available.

Action

- (1) Create the System Requirements Specifications by taking account of the items listed in (SY101) through (SY105).
- (2) If multiple alternative draft documents of System Requirements Specifications have been examined, the most desirable document must be selected and finalized in this sub-task.
- (3) In conjunction with the above, organize the system design guidelines as well.

Precaution

- ▶ The requirements pertaining to the following points should be clearly specified in the System Requirements Specifications:
 - Peripheral systems and hardware that are related to the system that is going to be developed;
 - Input/output information or data exchanged between the system and external sources/destinations;
 - Functional and non-functional system requirements (that are prioritized);
 - Operational constraints of the system as a whole and individual functionalities (sequence, priority, concurrency).
 - ▶ Add the outcome of use case analysis (use case diagram, use case scenario, etc) on as-needed basis.
 - ▶ The following set of information should also be included to describe the system design guidelines:
 - Design-related constraints;
 - Applicable techniques for optimal designing;
 - System platform suitable for implementation;
 - Possibility of reusing the existing system.
 - ▶ When there are still uncertainty factors, they should also be stated explicitly in the System Requirements Specifications.
 - ▶ Points to keep in mind in documentation:
 - Attach the revision history and indicate clearly where have been revised;
 - Clearly indicate who or which organization is responsible of the created document;
 - Ensure that the created document is managed properly by performing configuration management and change management.
- Related processes: SUP5 Configuration Management; SUP7 Change Management

■ SYP1.2 Reviewing the System Requirements Specifications

Confirm that the defined system requirements meet the product requirements.

1.2.1 Reviewing the System Requirements Specifications Internally

Input

(SY106) System Requirements Specifications

Outline

Check whether or not the contents of System Requirements Specifications cover all what is required for the system, and document the findings orderly in the form of an Internal Review Report.

Output

(SY107) Internal Review Report (on System Requirements Specifications) 

 Document template sample is available.

■ Action

(1) Review the System Requirements Specifications (SY106) internally, based on the following perspectives:

- ▶ Whether the prerequisite environment and conditions for system operation, and relevant contextual information are clearly described or not;
- ▶ Whether the intended users and the scenarios of their operation are clearly described or not;
- ▶ Whether the functional and non-functional requirements of the system are clearly described or not, based on the findings of the above check points;
- ▶ Whether the functional and non-functional system requirements checked above are clearly prioritized or not.

- ▶ Whether open issues that are still undecided (“TBD (To Be Determined) matters”) are clearly indicated or not;
- ▶ Whether the requirements described in System Requirements Specifications are appropriate or not from the standpoint of business (in terms of business viability, novelty, etc).

Related processes: SUP3 Risk Management

(2) Document the findings of the above check points orderly in the form of an Internal Review Report (SY107) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

■ Precaution

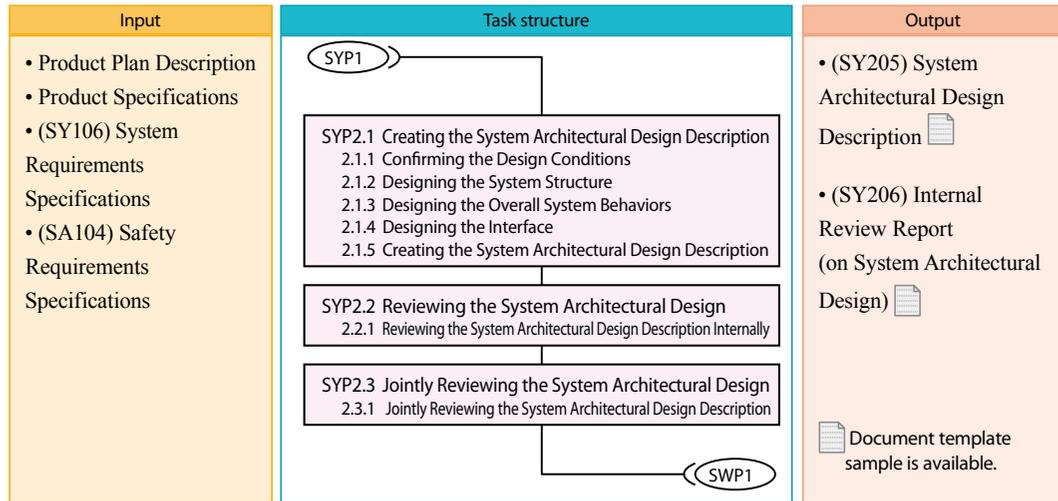
- ▶ In reviewing the System Requirements Specifications, it is desirable to invite the following members as reviewers:
 - Developers and engineers engaged in the current system development;
 - Members who participated in the study group to define the Product Specifications and Product Plan Description;
 - Engineers who have been involved in similar system development projects in the past.
- ▶ The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably

together with possible solutions and/or actions that address these issues.

- ▶ Issues found in the early stage of development when the system requirements are defined should be addressed as soon as possible to prevent them from growing or leading into bigger problems in the latter half of the development process. Therefore, the information mentioned in the Internal Review Report should be shared with the stakeholders that include the project manager, development team leader and personnel in charge of product planning, and their consensus should also be built this early stage.

SYP2 System Architectural Design

Examine how to actualize the embedded system to be developed, including the division of roles of hardware and software.



Description

In this activity:

- (2.1.1) (2.1.2) Confirm the functional and non-functional system requirements that must be achieved, based on the System Requirements Specifications. Moreover, confirm the constraints in achieving the system requirements from the standpoint of how to realize the specifications;
- (2.1.3) Sort out the functionalities that constitute the system from the standpoint of both the hardware and software, and examine the behavior and structure of the system by taking account of the division of roles played respectively by the hardware and software;
- (2.1.4) Design the interface between the functional blocks that constitute the system as well as the interface between the system and external entities;
- (2.1.5) Create the System Architectural Design Description by arranging the results of the above sub-tasks in an orderly format;
- (2.2.1) Review the created System Architectural Design Description internally, based on the pre-defined check points, and document the outcome of the internal review orderly in the form of an Internal Review Report;
- (2.3.1) Also hold joint review meetings with stakeholders, and document the outcome of the joint review orderly in the form of a Joint Review Report.

Consideration

- ▶ Examine the architecture to be designed for embedded system from multiple standpoints, including the estimate total cost to realize the system,

requirements on system performance, as well as the expected level of reliability, safety, reusability, and scalability. Also examine whether it is better

to achieve the required functionalities through software or through hardware.

- ▶ In addition, explore the possibility of whether some functionalities constituting the existing sys-

tems are going to be reused or not.

- ▶ Preferably, make use of drawings that will help grasp the overview of the entire system and visualize the outcome of the architectural design.

<Reference> Techniques and Tools

- ▶ Design modeling methodology
 - UML (Undefined Modeling Language)
- ▶ Scenario analysis (use case diagrams, activity diagrams, etc)

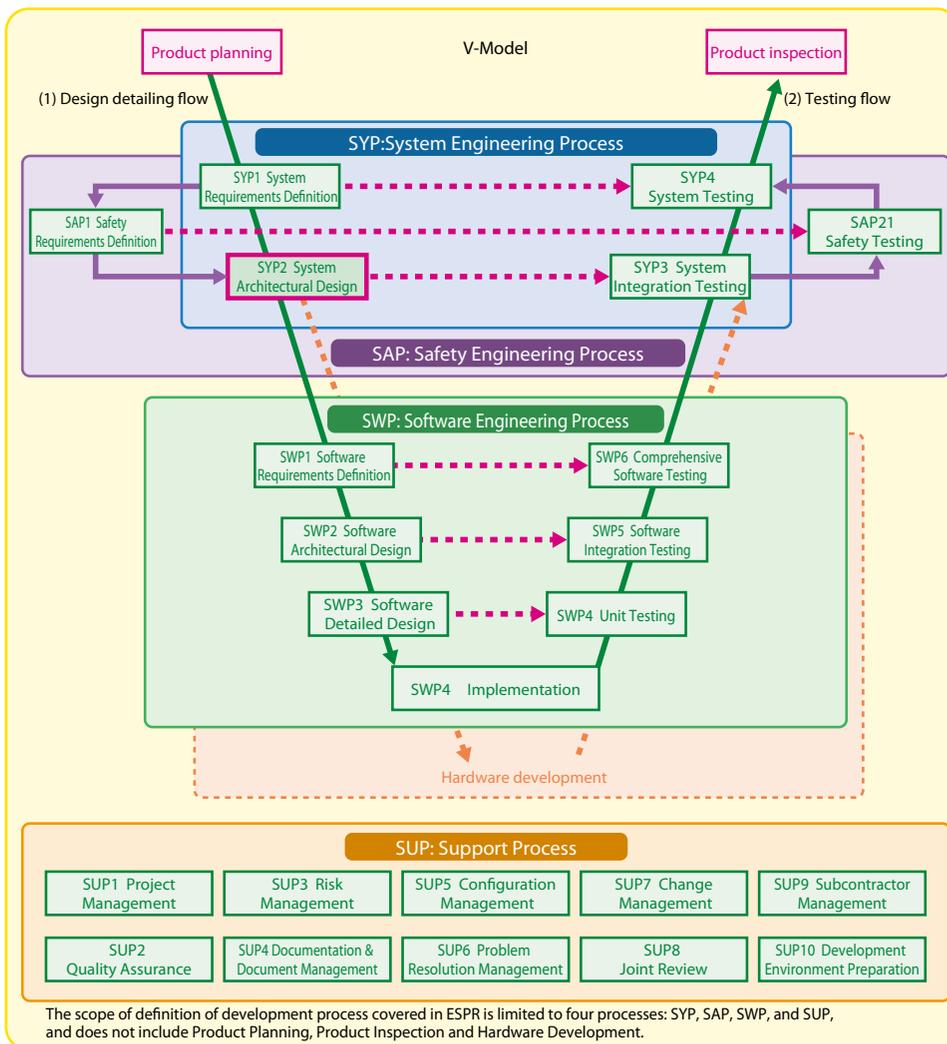


Figure 2.3 V-Model and Development Process (SYP2 System Architectural Design)



SYP2.1 Creating the System Architectural Design Description

Examine how to achieve the requirements (on system architecture) defined in the System Requirements Specifications, including the division of roles played by the hardware and software, and document the findings orderly in the form of System Architectural Design Description.

2.1.1 Confirming the Design Conditions

Input

- Product Plan Description
- Product Specifications
- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications

Outline

Confirm what the requirements and conditions are in designing the system architecture.

Output

(Design Conditions Confirmation Note)

Action

- (1) Confirm the contents of the following items described in the System Requirements Specifications:
 - ▶ System constraints (including the operational constraints);
 - ▶ Functional and non-functional system requirements.
- (2) Reconfirm the system's operating environment.
- (3) Confirm the conditions of using the existing system in case the existing system is going to be partially used or extended.
- (4) Also confirm the future possibility of expanding the functionality of the current system.
- (5) Confirm the scope of current system development.

Precaution

- ▶ Design conditions that can be defined numerically should be stated explicitly with the respective numerical targets.
- ▶ Bear in mind the necessity of cross-development

and have the differences between development environment and execution environment clarified beforehand.

2.1.2 Designing the System Structure

Input

- Product Plan Description
- Product Specifications
- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications

Outline

Design the system structure by sorting out the system functionalities from the standpoint of both the hardware and software, and extracting the functional blocks that structure the system.

Output

- (SY201) System Structural Diagram (Functional Block Diagram)
- (SY202) List of Common Functionalities and Data

Action

- (1) Identify the high-level functionalities that structure the system, and extract them as functional blocks.
- (2) Examine the division of roles played by the hardware and software in each functional block.
- (3) Examine the relationship between respective functional blocks.
- (4) Examine the general contents of each functional block, and divide the functionalities specified in the System Requirements Specifications respectively into functional blocks.
- (5) Identify the functionalities and data used commonly by the entire system.
 - Based on the outcome gained from the above, create a Functional Block Diagram that shows the functional structure of the system schematically. Depending on the type of processing or service that the system is expected to perform, there may be cases when it is better to extract and break down the system into components based on the data handled by the system.

Precaution

- ▶ Examine whether it is better to achieve the required functionalities by means of software or hardware.
 - Examine the division of roles played by the hardware and software from multiple standpoints, including the estimate total cost to realize the system, requirements on system performance, expected level of reliability, safety, reusability, and scalability, as well as the appropriate method to realize the system (whether through hardware or software).
- ▶ Also explore the possibility of reusing a part of the functionalities provided by the existing systems to develop the required system, based on the following needs:
 - Develop the product in series in the future with new versions and variants;
 - Improve the foreseeability of the entire system (structure, behaviors, etc)
 - Develop a system that can easily be debugged and maintained;
 - Ensure high reliability and safety.
- ▶ Explore, in addition, the possibility of purchasing or implementing external component blocks to support any of the functionalities that structure the system;
- ▶ Organize the functional blocks in levels that will make them easily reusable later on, by bearing in mind the future possibility of developing new versions and variants of the product in series;
- ▶ Use descriptive diagrams that will help improve the development foreseeability of the entire system.
- ▶ Also examine which functional block will handle the initialization of the system;
- ▶ Preferably, the functionalities for diagnosing the hardware and software, debugging and testing should also be examined;
- ▶ Also consider designing the redundancy of functionalities that need to be highly reliable and safe;
- ▶ Clarify also the MPU, ROM/RAM capacity, LSI, related control devices and sensors used for the system, among others.

2.1.3 Designing the Overall System Behaviors

Input

- Product Plan Description
- Product Specifications
- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications

Outline

Examine and sort out the overall system behaviors by bearing in mind the functional blocks that structure the system.

Output

- (SY203) System Behavioral Design Description

Action

- (1) Gain an organized view on how the functional blocks are co-related to enable the system to provide the required functional services.
- (2) Sort out the operational scenarios (sequences) by considering the possibility of concurrent operation of multiple functional blocks and examining the operational timing of the functional blocks. Also list the possible interrupt actions.
- (3) Clarify the operational context of the functional blocks that structure the system.
- (4) Associate the functional blocks respectively with the use cases that have been examined in system requirements analysis.
- (5) Roughly estimate the time-related performance of software and of hardware by bearing in mind the non-functional requirements of the system, and considering the time constraints (like response time) that affect the system operation.
- (6) Gain a clear concept on how the system handles errors.
- (7) Examine the framework on how to ensure the level of safety required by the system by considering the conditions in which the system is intended to be used. 

Precaution

- ▶ Examine not only the normal behaviors expected from the system, but also its abnormal behaviors caused by abnormal processing and other reasons.
- ▶ Clarify the state transitions of the system by taking account of the internal states of the system (consider the state transition of the system as a whole that includes both the hardware and software).

 : Work related to safety

2.1.4 Designing the Interface

Input

- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications
- (SY201) System Structural Diagram (Functional Block Diagram)
- (SY202) List of Common Functionalities and Data

Outline

Clarify the external and internal interfaces used for the system.

Output

- (SY204) System Interface Design Description

See also

Hardware specifications

Action

(1) Clarify the external interface.

- ▶ Clarify the interface between the system to be developed and the external systems.
- ▶ Clarify the types of data and information exchanged via the external interface, as well as the method or means applied for communication as well as for data transmission and reception to make the interface work.

(2) Clarify the internal interface.

- ▶ Clarify the interface between the functional blocks that structure the system.
- ▶ Clarify the types of data and information exchanged

via the internal interface, as well as the method or means applied to make the interface work between system units (such as, bus and shared memory).

(3) Also clarify the interface between software elements and hardware elements.

(4) As regards the interface between the system to be developed and external system, also consider the specifications of intermediate devices like sensors and actuators, and the data related to them, and create a table that lists the related data and/or sensor data on as needed basis.

Precaution

- ▶ As regards the interface between the software and hardware, consider the specifications and characteristic properties of the hardware device, and confirm that the designed interface conforms to the specifications of the interfaced hardware device on as needed basis.
- ▶ When data exists in between as the interface, clarify its type and size, as well as the timing or interval to receive the data.
- ▶ In examining the external system interface,

include the user interface (audio, visual, etc) in the examination if they exist. In such cases, also consider the non-functional system requirements that pertain to usability.

- ▶ There is a need to clarify and sort out all the interfaces between the hardware and software including the following:
(Example) Definition of interrupt;
Input/output register read and written by the software.

2.1.5 Creating the System Architectural Design Description

Input

- (SY201) System Structural Diagram (Functional Block Diagram)
- (SY202) List of Common Functionalities and Data
- (SY203) System Behavioral Design Description
- (SY204) System Interface Design Description

Outline

Sort out all the matters pertaining to system architectural design, and document them orderly in the form of System Architectural Design Description.

Output

- (SY205) System Architectural Design Description 

 Document template sample is available.

Action

Sort out the information outputted in (SY201) through (SY204), and document them orderly in the form of System Architectural Design Description.

- ▶ Sort out the information outputted in various materials in the course of designing the system architecture, and document them systematically.
- ▶ Confirm that the created document adequately reflects all the points indicated in internal and joint reviews.

Precaution

- ▶ In creating the System Architectural Design Description, information on static and dynamic (behavioral) structures of the system as a whole should be included to make the system to be developed easily foreseeable.
- ▶ The prerequisites for system architecture should be stated explicitly in the System Architectural Design Description. Moreover, the reasons why the architecture was defined to be so should also be explained in this document.
- ▶ It is desirable to perform version management that will enable the readers to know which part of the document has been revised or added, and attach the list of past versions.
- ▶ Points to keep in mind in documentation:
 - Attach the revision history and indicate clearly where have been revised;
 - Clearly indicate who or which organization is responsible of the created document;
 - Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management; SUP7 Change Management



SYP2.2 Reviewing the System Architectural Design

Confirm that the system architecture that has been designed satisfies the system and safety requirements.

2.2.1 Reviewing the System Architectural Design Description Internally

Input

- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications
- (SY205) System Architectural Design Description
- (SY201) System Structural Diagram (Functional Block Diagram)
- (SY203) System Behavioral Design Description
- (SY204) System Interface Design Description

Outline

Confirm that the contents of the created System Architectural Design Description satisfy and are appropriate for the specifications of the system requirements, and document the findings orderly in the form of an Internal Review Report.

Output

- (SY206) Internal Review Report (on System Architectural Design) 

 Document template sample is available.

See also

Hardware specifications, etc

Action

(1) Check whether the contents of System Architectural Design Description are appropriate or not.

- ▶ Check whether the functional blocks that structure the system are divided appropriately or not, and whether they can enable the system to achieve fully what it is required to provide, as described in the System Requirements Specifications (confirm the traceability).
- ▶ Check whether the division of roles played respectively by the software and hardware is appropriate or not.
- ▶ Check whether the prerequisites for dividing the functional blocks and the roles played respectively by the software and hardware are correct or not.
- ▶ Check whether the non-functional system requirements are all achievable or not.
 - Check whether the system can achieve the expected level of performance (on efficiency,

etc) or not.

- Check whether the maintainability and portability of the system can be appropriately achieved, when considering the potential expansion of the system in the future.
- ▶ Check whether or not the usability and safety of the system are taken into account, based on the targeted system users and the context of their use.
- ▶ Check whether or not the system architecture is designed to enable the system to function systematically as a whole.
- ▶ Check whether or not the design of the system architecture is traceable from the system requirements, and whether the designed system architecture can be traced to the test specifications.
- ▶ Confirm the feasibility of the finalized system architecture design.

(2) Document the findings of the above check points orderly in the form of an Internal Review Report (SY206) where the issues raised in the internal review and the personnel in charge of handling

these issues are stated explicitly, and distribute this report to the relevant members of the development project.

■ Precaution

- ▶ One of the effective ways of checking the contents of the System Architectural Design Description is to hold review meetings. As regards the points of concern on holding review meetings, refer to the description under SWP2.2.
- ▶ Conduct the review that take the system design conditions into account, as well as the background and reasons for reaching the final system architecture design.
- ▶ When multiple members are assigned to examine the different aspects of system architecture, make sure that the views on system architecture held by the members do not conflict drastically.
- ▶ The review manager should make sure that the issues raised in the internal review meetings are

included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues.

- ▶ Issues found in the early stage of development when the system architecture is designed should be addressed as soon as possible to prevent them from growing or leading into bigger problems in the latter half of the development process. Therefore, the information mentioned in the Internal Review Report should be shared with the project manager, development team leader, personnel in charge of product planning, and other relevant stakeholders, and their consensus should also be built at this early stage.



SYP2.3 Jointly Reviewing the System Architectural Design

Evaluate the designed system architecture among the stakeholders from the standpoint of how much it meets the product plan and system requirements.

2.3.1 Jointly Reviewing the System Architectural Design Description

Input

- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications
- (SY205) System Architectural Design Description
- (SY206) Internal Review Report (on System Architectural Design)

Outline

Hold joint meetings among stakeholders to review the contents of the System Architectural Design Description and confirm the validity of the proposed methods to fulfil the system requirements.

Output

- (SU801) Joint Review Records (System Architectural Design Description) 
- (SY208) System Architectural Design Joint Review Report

 Document template sample is available.

Action

- (1) Hold joint meetings among stakeholders of system development (personnel in charge of product planning, software developers, hardware developers, system evaluators, personnel in charge of manufacturing, etc) to review the validity of the proposed system architectural design. In these joint review meetings, focus on examining in particular the following points dealt in the System Architectural Design Description:
- ▶ Is the “how” (system architectural design that has been studied) meeting the “what” (system requirements)?
 - ▶ Is the feasibility of the system architectural design ensured?
 - ▶ Are the requirements for software development clearly described?
 - ▶ Are the requirements for hardware development clearly described?
 - ▶ Are the requirements for integrating and evaluating the system clearly described?
 - ▶ Are security-related aspects (such as, system vulnerability) taken into consideration?
 - ▶ Are system safety-related aspects taken into consideration?
 - ▶ Are the system development environment, system’s operating environment (OS, Lib) and middleware used in the system clearly described?
 - ▶ Are techniques and approaches for operation evaluation during the manufacturing phase taken into consideration?
 - ▶ Are the laws and regulations applied to the domain related to the system clearly described and are they being observed?
 - ▶ Are system constraints clearly described?

• Terminology

* Stakeholders: Individuals belonging to corporate entities to end users who have interest in the product

(2) Create a Joint Review Report, based on the review records.

- ▶ Explicitly state in the Joint Review Report all

the issues and possible solutions raised during the review, and distribute this document to the relevant stakeholders.

■ Precaution

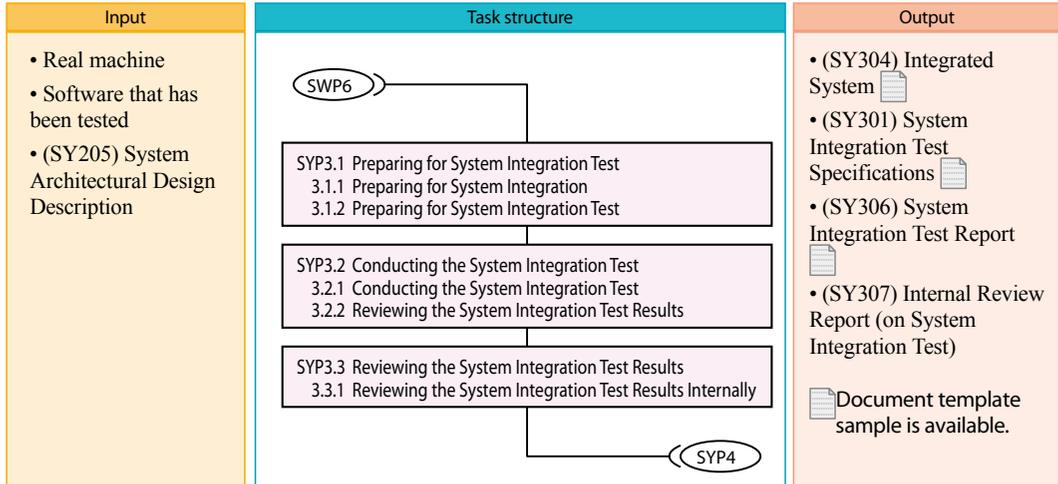
- ▶ Holding formal-styled review meetings where the participating members have clearly assigned roles (as chairperson, secretary, reviewers, etc) would be effective in balancing the interests of the stakeholders smoothly, should they conflict.

- ▶ Joint reviews are also desirable to evaluate the functional blocks, because they will be examined from various angles by multiple stakeholders with varying interests and viewpoints.

Related processes: SUP8 Joint review

SYP3 System Integration Testing

Confirm that the functional blocks operate when the hardware and software that structure the system are integrated.



Description

In this activity:

- (3.1.1) As one of the tasks to prepare for the finalization of system development, prepare for system integration by making the software that has gone through the Comprehensive Software Test phase in Software Engineering Process (SWP) and the real machine (hardware) used for the product available;
- (3.1.2) Prepare for the System Integration Test;
- (3.2.1) Integrate the real machine and the software that is built to be mountable in a real machine, conduct the System Integration Test, and review the test results;
- (3.2.2) (3.3.1) Review the results of the System Integration Test, based on the pre-defined check points, and document the outcome of this review orderly in the form of an Internal Review Report.

In the context of embedded system development, System Integration Test can be positioned as the opportunity to confirm that the integration of software and hardware has been achieved in accordance with the system requirements and system architectural design.

Consideration

- ▶ The integration of the embedded system is achieved by integrating the software with the real machine (hardware). Before integrating the system, there is a need to examine in which order the software and hardware should be integrated, and how to confirm that the integrated system works properly.
- ▶ Arrange the integration schedule in such a way so that the engineers in charge of hardware and the engineers in charge of software can collaborate smoothly.

<Reference> Techniques and Tools

- ▶ Bug management tool
- ▶ Reliability growth curve

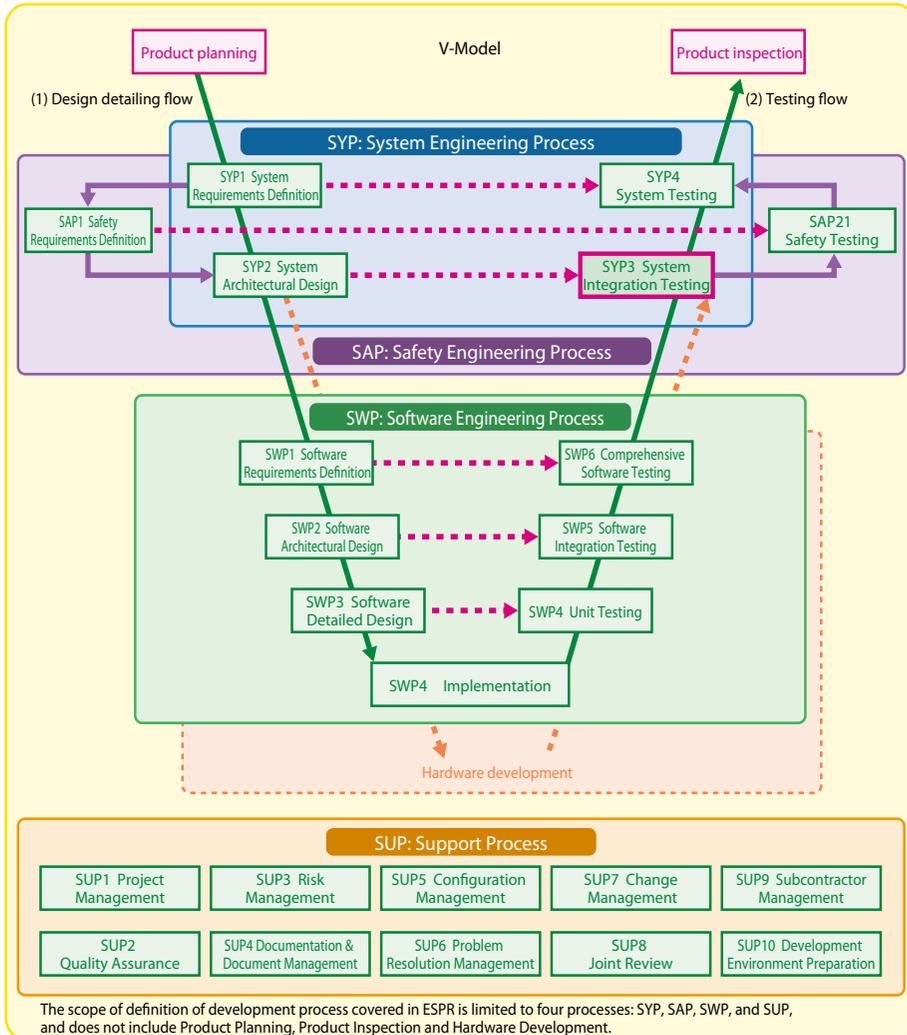


Figure 2.4 V-Model and Development Process (SYP3 System Integration Testing)



SYP3.1 Preparing for System Integration Test

Prepare for the system integration tests.

3.1.1 Preparing for System Integration

Input

- Real machine
- Software that has been tested
- (SY205) System Architectural Design Description

Outline

As one of the tasks to prepare for the finalization of system development, prepare for system integration by making the software that has gone through the Comprehensive Software Test phase in Software Engineering Process (SWP) and the real machine (hardware) used for the product available.

Output

(Software installed in ROM, etc)

Action

- (1) Be ready with the real machine (hardware) that is necessary to realize the system.
- (2) Be ready with the software that has passed the comprehensive software tests.
- (3) Set the above-mentioned software in the state that is executable by the computer inside the product.

Precaution

- ▶ “Software in the state that is executable by the computer” means the state in which the software is installed in ROM for target micro-computer.
- ▶ Keep an eye on the respective version of the software and hardware that will be integrated.
- ▶ Before integrating the system, there is a need to examine in which order the software and hardware should be integrated, and how to confirm that the integrated system works properly.
- ▶ Be prepared also with the procurement and installation of the real machine and in-circuit emulator (ICE).
- ▶ It is also important to arrange the integration schedule in such a way so that the engineers in charge of hardware and the engineers in charge of software can collaborate smoothly.

3.1.2 Preparing for System Integration Test

Input

- (SY205) System Architectural Design Description
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Prepare to conduct the integration test to the system when it is built into the final integrated form.

Output

- (SY301) System Integration Test Specifications
- (SY302) System Integration Test Data
- (SY303) Internal Confirmation Note (on System Integration Test Specifications)



Document template sample is available.

Action

- (1) List the items that need to be checked at the time of system integration, and create the test cases for System Integration Testing, based on these check items.
- (2) Devise test cases for checking the interface when the software and hardware are integrated.
- (3) Devise test cases for confirming that all the functionalities that need to be provided by the system can be actualized by the integrated system.
- (4) Create the test data used to conduct the above-mentioned test cases. Also have the various test criteria defined beforehand, including the criteria for judging the test results, the criteria for evaluating the System Integration Test on the overall, and the criteria for determining the satisfactory completion of the System Integration Testing activity.
- (5) Sort out the above-mentioned test cases and test data, and create a document called System Integration Test Specifications to describe them in an orderly manner.
- (6) Prepare the test cases for verifying the modifications (when modifications are implemented and need to be verified).
 - ▶ When a defect is detected during the System Integration Testing activity and modification has been implemented to resolve it, prepare test cases to verify that the modification was successful in eliminating the defect, and that no other problems have derived from this modification.
 - ▶ Determine the scope of the modification verification test and select the appropriate test cases, based on the description of the defect.

Precaution

- ▶ When testing the interface between the software and hardware, be especially attentive about the data input to and/or output from the software via the hardware.
- ▶ For integration of a complex system that uses more than one CPU, the hardware and software

that structure the system do not necessarily have to be integrated all at once. Therefore, for testing such complex systems, prepare test cases designed to be conducted one by one to check only the functionalities achieved by the specific portion of the system that has been integrated, by

experimentally activating the part of the system required to perform the given functionalities.

- ▶ Examine whether or not the pass/fail test criteria should also be included in the System Integration Test Specifications.
- ▶ Take account of the configuration management process, if necessary. Also think about the sequence of integrating the system partially (if integrated portion by portion) and the sequence of testing the integrated system portions.
- ▶ Since there are integration test cases designed to cover only a specific portion of the system as mentioned earlier, consider also using simulation test equipment, test programs and ICE, among others.
- ▶ Be sure to review the description of the System Integration Test Specifications.
- ▶ Points to consider when preparing the test for verifying the modification:

- Is the scope of implementation of this test covering all the potentially affected areas?
- Basically, reuse the test cases that have already been created. Consider preparing new test cases when the modification was large-scaled, and extensive areas were affected.

▶ Points to keep in mind in documentation:

- Attach the revision history and indicate clearly where have been revised;
- Clearly indicate who or which organization is responsible of the created document;
- Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management; SUP7 Change Management



SYP3.2 Conducting the System Integration Test

Conduct the System Integration Test.

3.2.1 Conducting the System Integration Test

Input

- Software that has been tested
- Real machine
- (SY301) System Integration Test Specifications
- (SY302) System Integration Test Data
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Integrate the real machine and the software that is built to be mountable in a real machine, and conduct the Integration Test.

Output

- (SY304) Integrated System
- (SY305) System Integration Test Results

Action

- (1) Embed the software built to be mountable in the product into the real machine (hardware).
- (2) Based on the test cases described in the System Integration Test Specifications, conduct the Integration Test one by one to check the functionalities achieved by operating the integrated system.
 - ▶ When the integrated system is tested by using an alternative test case or data, keep records of the test results, along with the reason(s) stating explicitly why the alternative test case or data had to be used.
 - ▶ When a prepared test case cannot be conducted, keep records of the event, along with the reason(s) stating explicitly why it was not executable. Moreover, determine the reasonableness of the stated reason(s).
- (3) Conduct the test for verifying the modification to:
 - ▶ Check whether the defect has been eliminated or not by the modification;
 - ▶ Check whether the modification to resolve the defect has led to any other defects or not.

Precaution

- ▶ When a defect that has been detected while testing is resolved by implementing a modification, there is a need to identify the functional blocks that are related to the defective functional block, and conduct all the tests that are necessary to confirm that they all function normally again.
- ▶ Before running the test for verifying the implemented modification, be sure to check that the latest modified version is tested.

3.2.2 Reviewing the System Integration Test Results

Input

- (SY301) System Integration Test Specifications
- (SY305) System Integration Test Results
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Review the results gained from the System Integration Test, and judge whether the tested system has passed or failed this test.

Output

- (SY306) System Integration Test Report 
 - (SU601) Defect Management Ticket 
-  Document template sample is available.

Action

- (1) Judge whether the system in its final integrated form has been successfully integrated or not, by referring to the criteria described in the System Integration Test Specifications.
 - (2) When a defect has been detected while running the System Integration Test, investigate the root cause of the defect (whether it was caused by or in the software or hardware, for example), and document the findings in Defect Management Ticket.
 - (3) In the System Integration Test Report, state explicitly the following set of information, among others:
 - ▶ Test methods, test environment, tools and data that have been used;
 - ▶ Number of defects that have been detected, which of them are considered critical, result of the defect analysis;
 - ▶ Judgment of whether the System Integration Test was completed successfully or not, and the grounds that.
- Related processes: SUP6 Problem Resolution Management

Precaution

- ▶ When a defect has been detected while testing, first try reproducing the defect, and clarify the context and the state the system was in.



SYP3.3 Reviewing the System Integration Test Results

Review the results of the System Integration Test from the standpoint of checking whether or not the integrated embedded system is capable of processing correctly what it is required to achieve as defined in the System Architectural Design.

3.3.1 Reviewing the System Integration Test Results Internally

Input

- (SY205) System Architectural Design Description
- (SY301) System Integration Test Specifications
- (SY303) Internal Confirmation Note (on System Integration Test Specifications)
- (SY306) System Integration Test Report
- (SU601) Defect Management Ticket

Outline

Review the contents of the System Integration Test Report, check whether or not there have been any issues that could not be solved at the stage of system integration, and examine the possible solutions.

Output

- (SY307) Internal Review Report (on System Integration Test) 

 Document template sample is available.

Action

(1) Review the results of the System Integration Test from the following perspectives:

- ▶ When an unsolved issue is found:
 - Evaluate the severity (level of importance) of the issue;
 - When the issue is evaluated to be a critical problem that affects the functionality, reliability and/or safety of the entire system, carry out concrete countermeasures by examining the following points, among others:
 - a. Return to relevant system development processes (software and/or hardware);
 - b. Add restrictions to the conditions of using the system;

c. Reconsider the resource plan.

Related processes: SUP8 Joint review; SUP1 Project Management

- ▶ Check whether there have been any test cases that were not carried out, and if there were any, investigate the reason(s) why they were not carried out, and examine the possible solutions.

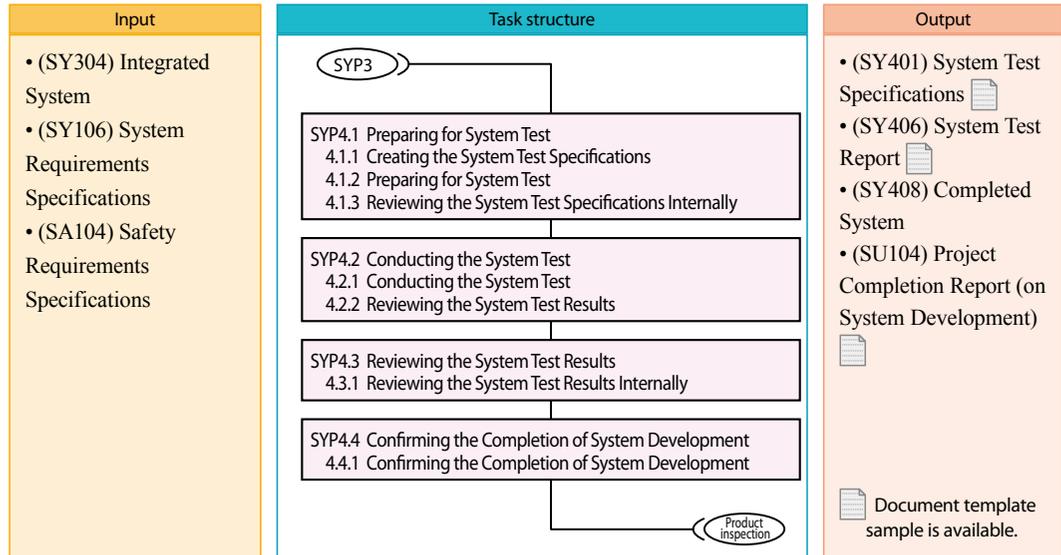
(2) Document the findings of the above check points orderly in the form of an Internal Review Report (SY307) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

- ▶ Check whether the number of defects that have been detected is acceptable or not, based on the quality criteria.
- ▶ When the test for verifying the modification has been conducted, check whether the scope of the test and the test environment have been appropriate or not.
- ▶ The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).actions (e.g., holding a joint review).

SYP4 System Testing

Confirm that the system requirements are fully met.



Description

In this activity:

- (4.1.1) (4.1.2) (4.1.3) Prepare the test specifications for checking and reviewing whether or not the fully integrated system is capable of operating the functionalities described in the System Requirements Specifications, and conduct a comprehensive set of tests collectively referred to as the System Test by actually operating the system according to the prepared test specifications (hereafter called the “System Test Specifications”);
- (4.2.1) (4.2.2) (4.3.1) Review the results of the System Test, based on the pass/fail criteria applied to this test, and document the findings orderly in the form of an Internal Review Report;
- (4.4.1) Make a final judgment as the organization in charge of product system development on whether the developed system has passed or failed the System Test, based on the test results documented in the System Test Report, and sort out the information deemed necessary for later product inspection.

System Testing can be regarded as the final activity in system development to test and review the developed system. Therefore, this activity must be carried out comprehensively without any omission, by assuming how the system actually operates when the product is used in the real world.

Consideration

- ▶ Before conducting the System Test, a particular attention should be given to the operating environment in which the system is tested (hereafter called the “test environment”). Conduct the System Test in a test environment assumed to be close to the environment in which the product is likely to be used by its users in the real world.
- ▶ Consider the System Test as the final opportunity to test and review the system that has been developed, and be careful not to leave out any test

cases, including not only the cases where the system is tested in conditions when the product is used in normal state but also the cases where the system is tested in abnormal conditions of use, as

well as the test cases for checking the compliance of the system to standards, conventions, laws and regulations that are relevant to the product.

■ <Reference> Techniques and Tools

- ▶ Automatic testing tool (automated regression test, etc)
- ▶ Bug management tool
- ▶ Reliability growth curve

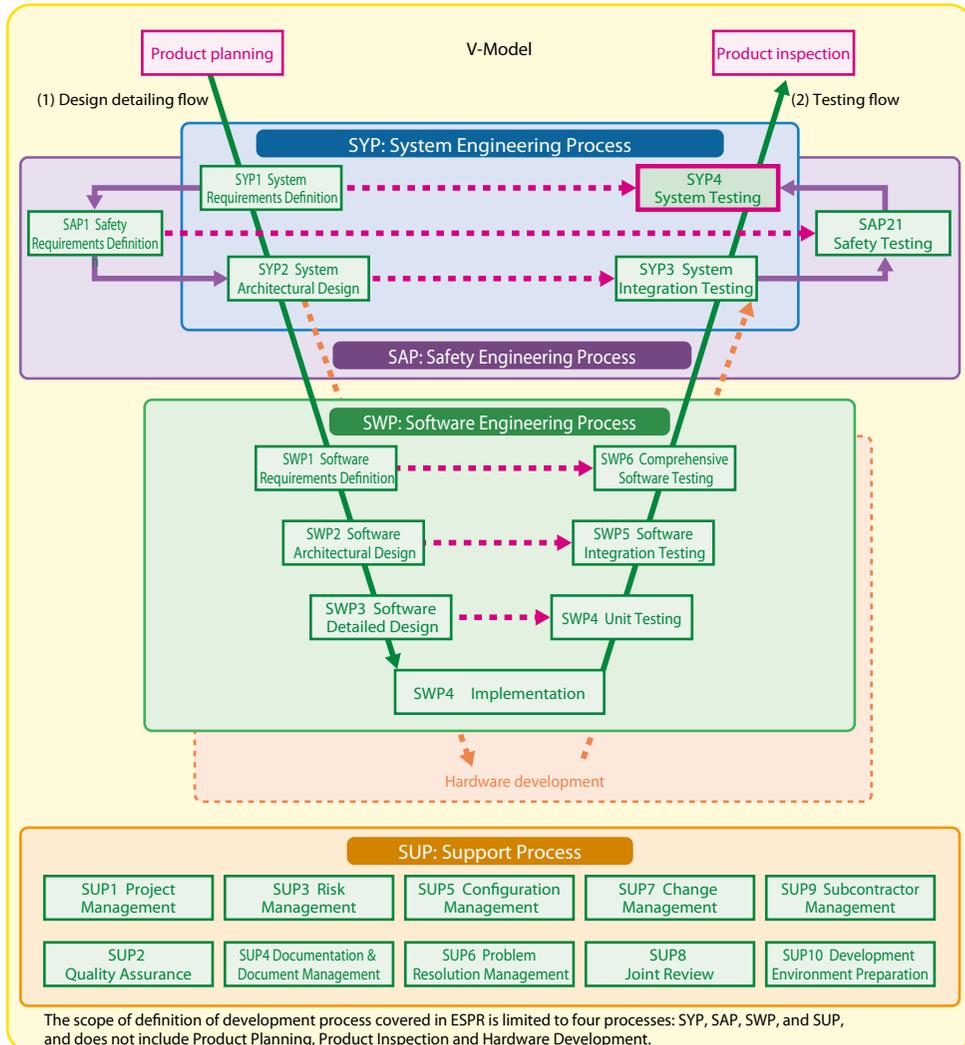


Figure 2.5 V-Model and Development Process (SYP4 System Testing)

SYP4.1 Preparing for System Test

Prepare for the System Test.

4.1.1 Creating the System Test Specifications

Input

- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications
- Product manuals
- Information (deliverables) necessary for test specifications

Outline

Identify the items that need to be tested as the product system based on the contents of System Requirements Specifications, and document them orderly in the form of System Test Specifications.

Output

- (SY401) System Test Specifications 

 Document template sample is available.

See also

System Test Specifications created in the past

Action

- (1) By keeping in mind that the System Test is intended to be conducted in a test environment assumed to be close to the user environment in which the developed system is actually used as the final product, create the System Test Specifications that consist of specifications for a set of tests devised from a wide variety of perspectives.

Structure the System Test with a set of tests devised from the following perspectives:

- ▶ Comprehensive tests that covers all the functionalities and services described in the product manuals;
- ▶ Tests pertaining to normal, quasi-normal and abnormal system operations;
- ▶ Tests pertaining to durability of the system in continuous use;
- ▶ Tests pertaining to system performance, including the processing capacity;

- ▶ Tests pertaining to usability of the system;
- ▶ Tests pertaining to maintainability of the system (online update, etc);
- ▶ Tests pertaining to system security;
- ▶ Tests pertaining to system safety.

Related processes: SAP2 Safety Testing

- ▶ Tests pertaining to compliance of the system to standards, conventions, laws and regulations that are relevant to the product.

- (2) By referring to the perspectives mentioned above, itemize the contents of the System Test Specifications into categories of test cases, prerequisites for the system's operating conditions (states), expected system behaviors, and other classifiable units.

- (3) Clarify the criteria for judging whether the tested system has passed or failed the System Test.

■ Precaution

- ▶ As regards the system's operating conditions (states), consider not only the conditions when the product is used in normal state but also the cases when the product is used under quasi-normal or abnormal conditions.
- ▶ If there are similar products developed in the past, take account of the defects that have been detected previously in those products as well.
- ▶ Review all the data related to the system without omitting any.
- ▶ Distinguish the tests conducted at the development side with the tests conducted at the user side.

 Safety

- ▶ It is desirable to examine and create the System Test Specifications during the upstream process (such as, during the design phase).
- ▶ Points to keep in mind in documentation:
 - Attach the revision history and indicate clearly where have been revised;
 - Clearly indicate who or which organization is responsible of the created document;
 - Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management; SUP7 Change Management

4.1.2 Preparing for System Test

Input

- (SY401) System Test Specifications System to be tested
- (SU601) Defect Management Ticket (when the test to verify the modification is conducted)

Outline

Prepare for the System Test (test data, test environment), based on the System Test Specifications.

Output

- (SY402) System Test Data
- (SY403) System Test Environment

Action

(1) Prepare the test environment.

- ▶ Since system behaviors in actual operating environment (i.e.: real machine environment) forms the basis of the System Test, prepare a test environment where the system can actually be used and operated.

(2) Prepare the test data.

- ▶ Prepare a set of data that would be necessary to carry out the test cases of the System Test (system input data, data operated by the user, etc).
- ▶ Examine the types of tests carried out in the System Test.
- ▶ Since some test cases may require the system to be in a very unusual operating environment (condition) or tested with a rare type of data, also examine how to be well prepared with all the necessary environment and data.

(3) Also have the various test criteria defined beforehand, including the criteria for judging the test results, the criteria for evaluating the System Test on the overall, and the criteria for determining the satisfactory completion of the System Testing activity.

(4) Prepare the test cases for verifying the modification (when modifications are implemented and need to be verified).

- ▶ When a defect is detected during the System Testing activity and modification has been implemented to resolve it, prepare test cases to verify that the modification was successful in eliminating the defect, and that no other problems have derived from this modification..
- ▶ Determine the scope of the modification verification test and select the appropriate test cases, based on the description of the defect.

Precaution

- ▶ Depending on the system, the organization in charge of development alone may not be able to prepare all the actual operating environments necessary to run the tests. Therefore, be sure to start preparing or procuring the test environments at an early stage, including the arrangements to gain support from others on the preparation, should there be any test environments that cannot be prepared alone.

If there are test environments that cannot be prepared physically by any means, consider preparing the necessary operating environment through feasible alternative methods, including the use of a simulator.

- ▶ Since the System Test is composed of a set of tests devised from various perspectives, the tests may be conducted by more than one tester. If multiple testers are assigned to handle a certain number of

test cases respectively, there is a need to prepare the same number of test environments and systems to be tested as the number of testers assigned to this activity.

- ▶ In some tests composing the System Test, the system behavior is decided at a subtle timing that is difficult to reproduce. Conceive special ways to enable the system to repeat its behavior in such delicate tests.
- ▶ Points to consider when preparing the test for verifying the modification
 - Is the scope of implementation of this test covering all the potentially affected areas?
 - Basically, reuse the test cases that have already been created. Consider preparing new test cases when the modification was large-scaled, and extensive areas were affected.

4.1.3 Reviewing the System Test Specifications Internally

Input

- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications
- (SY401) System Test Specifications
- System to be tested

Outline

Review the System Test Specifications.

Output

- (SY404) Internal Confirmation Notes (on System Test Specifications)

Action

Review the System Test Specifications that have been created.

- ▶ Check whether or not the test cases described in the System Test Specifications (SY401) can all be mapped respectively to the specific items described in the System Requirements

Specifications (SY106) for defining functional/non-functional requirements and operational constraints of the system.

Precaution

- ▶ Check the extent of the test perspectives described under SYP4.1.1 covered in the System Test Specifications.
- ▶ Confirm that the contents of the System Test

Specifications are fully covered and aligned with the information provided in the product manuals (e.g.: user manual).

SYP4.2 Conducting the System Test

Conduct the system test.

4.2.1 Conducting the System Test

Input

- (SY401) System Test Specifications
System to be tested
- (SY402) System Test Data
- (SY403) System Test Environment

Outline

Conduct the System Test by following the System Test Specifications.

Output

- (SY405) System Test Results

Action

Conduct the System Test by following the System Test Specifications.

(1) Conduct the System Test.

- ▶ Conduct the System Test, based on the System Test Specifications (SY401), and gain the test results as the output.
- ▶ When an alternative test is carried out, keep records of the test results, along with the reason(s) stating explicitly why the alternative test had to be carried out.
- ▶ When a defect is detected while testing, decide whether to continue conducting the remaining tests, or suspend them until the defect is resolved.
- ▶ Collect the output data of test results (various logs,

etc).

- ▶ When a prepared test case cannot be conducted, keep records of the event, along with the reason(s) stating explicitly why it was not executable. Moreover, determine the reasonableness of the stated reason(s).

(2) Conduct the test for verifying the modification to:

- ▶ Check whether the defect has been eliminated or not by the modification;
- ▶ Check whether the modification to resolve the defect has led to any other defects or not.

Precaution

- ▶ By considering the possibility of repeating the same tests later on, it is desirable to preserve all the test data necessary to reproduce the same test conditions for retesting.
- ▶ Before running the test for verifying the

implemented modification, be sure to check that the latest modified version is tested.

- ▶ When there is a need to correct or modify the system, do so after analyzing its impact.

4.2.2 Reviewing the System Test Results

Input

- (SY401) System Test Specifications
- System to be tested
- (SY405) System Test Results
- (SU601) Defect Management Ticket (when the test to verify the modification is conducted)

Outline

Review the results of the System Test, and judge whether the tested system has passed or failed this test.

Output

- (SY406) System Test Report 
- (SU601) Defect Management Ticket 

 Document template sample is available.

Action

Review the results gained from the System Test, and judge whether the tested system has passed or failed each test case that has been conducted.

- ▶ When a defect has been detected while testing, record the description of the defect in the Defect Management Ticket.
- ▶ If the defect has been confirmed by the test for verifying the modification that it has

been eliminated, record the findings (that the modification has been verified) in the Defect Management Ticket.

Related processes: SUP6 Problem Resolution Management

Precaution

- ▶ Do not take any test results for granted, and consider the possibility of incorrect information included in the System Test Specifications.
- ▶ Judge whether the tested system has passed or failed each test case, based on the test criteria described in the System Test Specifications.



SYP4.3 Reviewing the System Test Results

Review the results of the System Test from the standpoint of checking whether or not the system requirements defined in the System Requirements Definition have been correctly achieved by the developed system.

4.3.1 Reviewing the System Test Results Internally

Input

- (SY106) System Requirements Specifications
- (SA104) Safety Requirements Specifications
- (SY401) System Test Specifications
- (SY404) Internal Confirmation Notes (on System Test Specifications)
- (SY406) System Test Report
- (SU601) Defect Management Ticket

Outline

Review the contents of the System Test Report, check whether or not there have been any issues that could not be solved at the final stage of system development, examine the possible solutions, and document the findings orderly in the form of an Internal Review Report.

Output

- (SY407) Internal Review Report (on System Test) 

 Document template sample is available.

Action

(1) Review the results of the System Test from the following perspectives.

- ▶ When an unsolved issue is found:
 - Evaluate the severity (level of importance) of the issue;
 - When the issue is evaluated to be a critical problem that affects the functionality, reliability and/or safety of the entire system, carry out concrete actions by examining the following points, among others:
 - a. Return to relevant system development processes (software and/or hardware);
 - b. Add restrictions to the conditions of using the system;
 - c. Reconsider the release plan.

Related processes: SUP8 Joint review; SUP1 Project Management

- ▶ When there are any test cases that have been found to be omitted, investigate the reason(s) why they were not carried out, and examine the possible solutions.

Related processes: SUP6 Problem Resolution Management

(2) Document the findings of the above check points orderly in the form of an Internal Review Report (SY407) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

- ▶ Check whether the defect detection rate is acceptable or not, based on the quality criteria.
- ▶ When the test for verifying the modification has been conducted, check whether the scope of the test and the test environment have been appropriate or not.
- ▶ The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).



SYP4.4 Confirming the Completion of System Development

Evaluate the developed system among the stakeholders by checking whether the system requirements defined in the System Requirements Definition are correctly realized by the developed system or not.

4.4.1 Confirming the Completion of System Development

Input

- (SY106) System Requirements Specifications
- (SY401) System Test Specifications
- (SY406) System Test Report
- (SY407) Internal Review Report (on System Test)
- (SA104) Safety Requirements Specifications
- (SA201) Safety Test Specifications
- (SA206) Internal Review Report (on Safety Test)
- (SU101) Project Plan Description
- (SU601) Defect Management Ticket

Outline

Review the test results described in system test-related documents jointly among the stakeholders. Based on the findings from this joint review, make a final judgment as the organization in charge of product system development on whether the developed system has passed or failed the System Test, and sort out the information deemed necessary for later product inspection.

Output

- (SU801) Joint Review Records (on System Test) 
- (SU104) Project Completion Report (on System Development) 
- (SY408) Completed System

 Document template sample is available.

Action

(1) Hold joint meetings among stakeholders of system development (personnel in charge of product planning, software developers, hardware developers, system evaluators, personnel in charge of manufacturing, etc) to make a final judgment on whether the system has been successfully developed or not.

In these joint review meetings, focus on examining the following check points in particular:

- ▶ Check whether or not the developed system can win the assurance that it is certainly capable of satisfying both the functional and non-functional requirements when it is used as the product system by the intended users under the environment it is intended to be used in;

- ▶ Check whether there are still any pending issues in system development or not;
- ▶ Check whether the tested system has any quality-related issues or not, by using the metrics on, such as, the total number of defects detected during the System Testing activity, the total number of modifications implemented, the number of critical defects that have been identified and the number of modifications implemented to resolve these critical defects;
- ▶ Check whether or not the issues indicated in the reviews held in other testing activities prior to System Testing have all been appropriately addressed and solved.

Related processes: SUP2 Quality Assurance; SUP6 Problem Resolution Management; SUP8 Joint review

- (2) Record the results gained in this review, and create a document called the Completion Report, based on the review results.

For more information on the creation of the Completion Report, see “SUP1.4 Creating the Project

Completion Report”.

Distribute the created Completion Report to the stakeholders involved in the current system development.

Related processes: SUP1 Project Management

■ Precaution

- ▶ In this review, confirm that an adequate approach was taken to develop the system both quantitatively and qualitatively, and that the system developed as a result of this approach is valid as a newly created product.
- ▶ Since this review is the final opportunity to review

the system within the entire system development process, the judgment on whether the review results have been acceptable or not must be made by the high-ranking personnel responsible of the entire system development.

SWP : Software Engineering Process

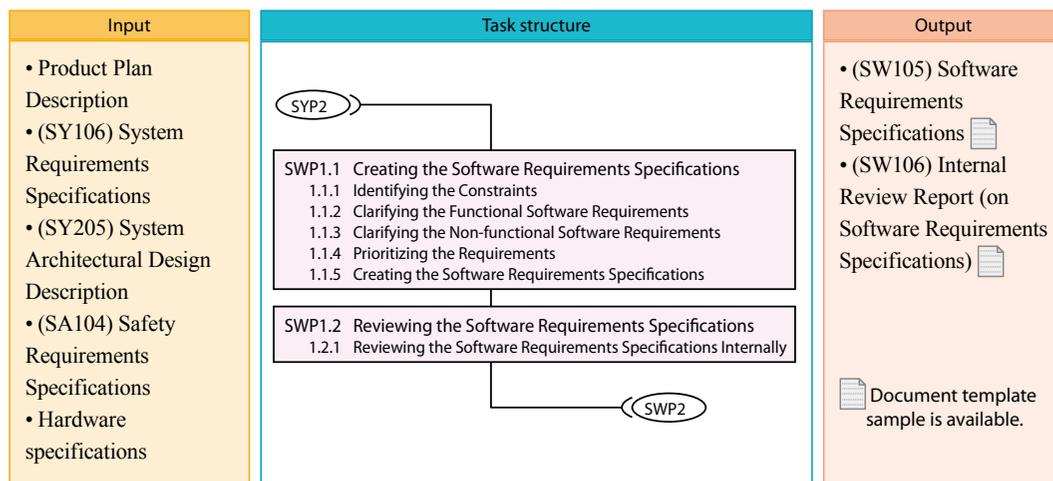
Among the various types of work pertaining to embedded software development, the activities and tasks ranging from Software Requirements Definition directly related to software production to Comprehensive Software Testing are defined in this process.

The following activities are included in this process:

ID	Activity	Outline of the activity	Comprising tasks	
SWP1	Software Requirements Definition	Clarify the requirements that the software must fulfill to realize the targeted product.	SWP1.1	Creating the Software Requirements Specifications
			SWP1.2	Reviewing the Software Requirements Specifications
SWP2	Software Architectural Design	Decide on the architecture (= behavior and structure) of the embedded software to be developed.	SWP2.1	Creating the Software Architectural Design Description
			SWP2.2	Reviewing the Software Architectural Design
			SWP2.3	Jointly Reviewing the Software Architectural Design
SWP3	Software Detailed Design	By dividing the functional units defined in the Software Architectural Design into program units, design the detailed behaviors and logical structure of the software.	SWP3.1	Creating the Functional Unit Detailed Design Description
			SWP3.2	Reviewing the Software Detailed Design
			SWP3.3	Checking the Consistency with Hardware Specifications
SWP4	Implementation & Unit Testing	Implement the respective units that structure the software and test the behaviors of the software at unit level.	SWP4.1	Preparing for Implementation and Unit Test
			SWP4.2	Conducting the Implementation and Unit Test
			SWP4.3	Reviewing the Implementation and Unit Test Results
SWP5	Software Integration Testing	Assemble the individual program units one by one, and test the functionalities expected to be provided by the program units respectively when they are executed in combinations.	SWP5.1	Preparing for Software Integration Test
			SWP5.2	Conducting the Software Integration Test
			SWP5.3	Reviewing the Software Integration Test Results
SWP6	Comprehensive Software Testing	Conduct the Comprehensive Software Test, using the software in the state where all the functional units that structure the software are fully integrated.	SWP6.1	Preparing for Comprehensive Software Test
			SWP6.2	Conducting the Comprehensive Software Test
			SWP6.3	Reviewing the Comprehensive Software Test Results
			SWP6.4	Confirming the Completion of Software Development

SWP1 Software Requirements Definition

Clarify the requirements that the software must fulfill to realize the targeted product.



Description

In this activity:

- (1.1.1) Find out what the constraints in specifying the software requirements, based on the contents of the System Requirements Specifications and hardware specifications;
- (1.1.2) Clarify the functional requirements of the software;
- (1.1.3) Also clarify the non-functional requirements of the software;
- (1.1.4) Prioritize the individual functional and non-functional requirements by taking account of the constraints in actual software development, including the development period and resources, and the system constraints identified earlier in (1.1.1);
- (1.1.5) Create the Software Requirements Specifications by organizing the information gained from the above in an orderly manner;
- (1.2.1) Review the created Software Requirements Specifications, based on the pre-defined check points, and document the outcome of this review orderly in the form of an Internal Review Report.

Consideration

- ▶ Keep in mind the following points as the prerequisites for commencing the activity to define the software requirements:
 - Product specifications: The contents of the product specifications are already fixed and detailed to the descriptive level of user manual;
 - Schedule: The overall schedule is already fixed
- Product planning: Product strategies (such as, the end users' needs) are clearly defined;

- (product release/launch date, milestones shared with external stakeholders, etc);
- System architecture: Matters pertaining to the following are all clearly defined:
 - Division of functional role played by the software or hardware (software requirements are clearly defined), hardware structure, external interfaces, methods of achieving the required performance, maintenance-related functionalities, and security.
 - Prerequisites: Software (OS, libraries, etc) and existing products that will be used.
 - ▶ In defining the software requirements, keep in mind the following matters that have been examined in the System Requirements Definition:

- In case of embedded system, take account of the results gained from analyzing the external environment (such as, the system's operating environment) and the functional analysis on how the system is capable of responding to abnormalities caused by the external environment
- Take account of not only the functional aspects but also the non-functional aspects (such as, performance and maintainability);
- Also keep in mind the external system that operates closely or in conjunction with the system that is going to be developed;
- In addition, take account of the functionalities that the product is expected to continue providing in long term.

■ <Reference> Techniques and Tools

- ▶ OOA (Object Oriented Analysis)
- ▶ Structured analysis
- ▶ DFD (Data Flow Diagram: Analytical technique used in structured analysis)
- ▶ Scenario analysis
- ▶ Prototyping
- ▶ Quality function deployment

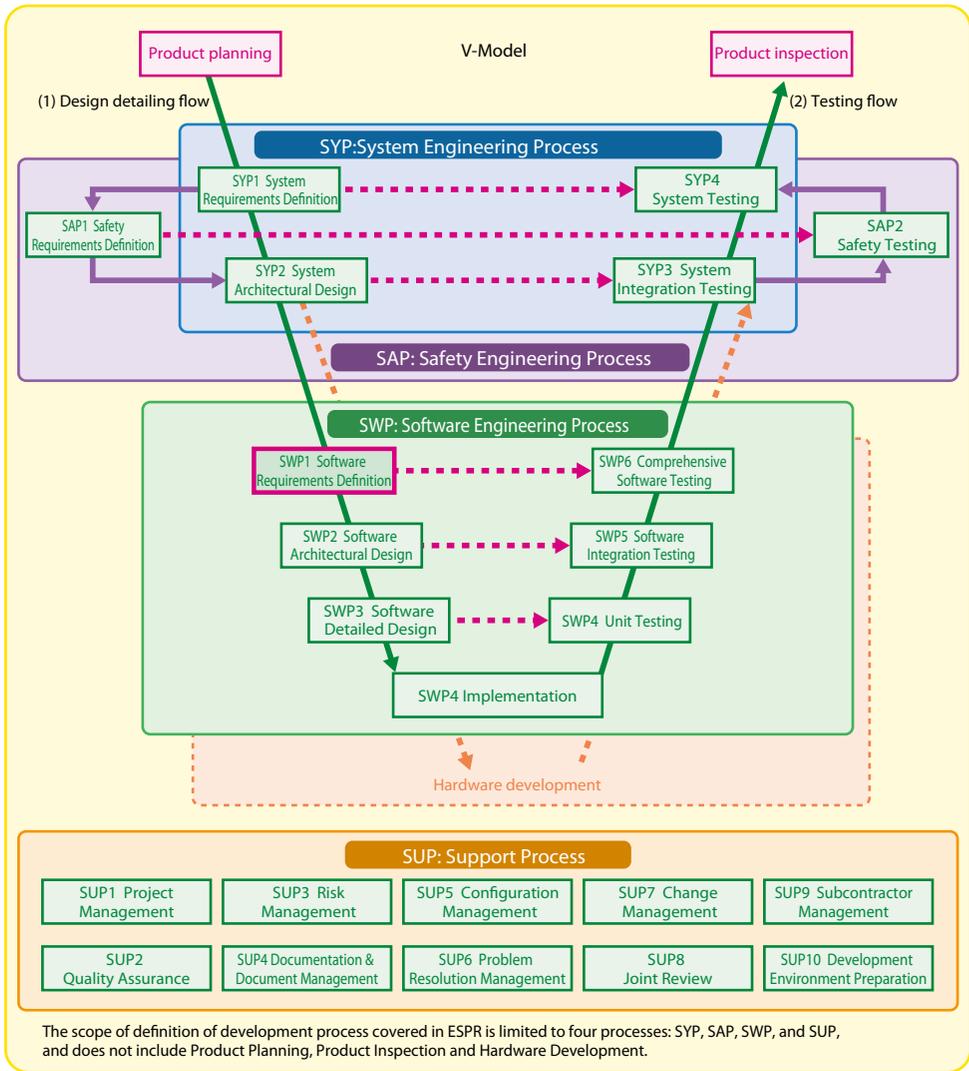


Figure 2.6 V-Model and Development Process (SWP1 Software Requirements Definition)



SWP1.1 Creating the Software Requirements Specifications

Clarify the items that the software must realize based on the contents of System Requirements Specifications, and document them orderly in the form of Software Requirements Specifications.

1.1.1 Identifying the Constraints

Input

- Product Plan Description
- (SY106) System Requirements Specifications
- (SY205) System Architectural Design Description
- (SA104) Safety Requirements Specifications
- Hardware specifications

Outline

Clarify the constraints that need to be considered when examining the software requirements, and sort them out in the form of a List of Constraints.

Output

- (SW101) List of Constraints

See also

- ISO/IEC 9126-1*(Software Quality Model)
- ISO 9241*(Usability)
- ISO 13407*(Human-centered Design Process)
- IEC 61508**(Safety)

Action

Based on the information clarified below from (1) to (6), document the findings orderly in the form of a List of Constraints:

- (1) Clarify the product plan and product development strategies, as well as the product objectives that must be taken into account when defining the software requirements, including the following:
 - ▶ Any new characteristic functionalities?
 - ▶ Will the product line development approach be taken?
- (2) Clarify the product characteristics like:
 - ▶ Requirements on reliability and safety; **Safety**
 - ▶ Serviceable life (durable years) and product life cycle;
 - ▶ Situation and environment in which the product is assumed to be used;
 - ▶ Standards and conventions that the product must comply with, if any.
- (3) Identify the stakeholder*** of the product.
 - ▶ Identify the stakeholders of the product (such as, service, sales, planning, hardware development, manufacturing departments/divisions).
 - ▶ Clarify the end users of the product and the characteristics of each user group;
 - ▶ Clarify the constraints that apply only to specific stakeholders only and must therefore be addressed separately, and include them in the List of Constraints.
- (4) Clarify the product structure, including the following:
 - ▶ Hardware structure and its constraints;
 - ▶ OS and middleware that are going to be used, and

• Terminology

* ISO/IEC 9126-1 (Software Quality Model), ISO 9241 (Usability), ISO 13407 (Human-centered Design Process) : Standards related to usability of the software, such as, the easiness to handle and operate the software.

** IEC 61508 (Safety) : Standard that defines what needs to be done to achieve safety during the development of a system consisting of one or more computers that must be highly safe to use (such as, a plant control system and automotive system)

*** Stakeholders :Individuals with interest in the product, ranging from those belonging to corporate entities to end users

Safety : Work related to safety

their respective constraints (include them in the List of Constraints);

- ▶ Interfaces for interconnecting the product with the peripheral software, systems, and hardware (such as, sensors and actuators).

(5) Check whether any existing software is going to be reused or not:

- ▶ Examine the need of reusing any existing software or not;
- ▶ If yes, clarify the specifications and characteristics

■ Precaution

● **Product planning and product development strategies:**

- ▶ Grasp the needs of the end users, product variations, medium- to long-term market strategies, etc;
- ▶ Grasp the overall schedule (product release/launch date, milestones shared with external stakeholders, etc), and clarify the length of time that can be allocated for software development;
- ▶ Also identify the functionalities that will enable the target product to be superior over the competing products;
- ▶ Be knowledgeable about the technical progress made in hardware platform, middleware and peripheral devices, and grasp the product strategies, including the timeframe on when advanced products are expected to be released.
- ▶ Review the product specifications (check whether they are fixed to the extent that can be published as user manual or not);
- ▶ Identify the TBD (To Be Determined) matters regarding the level of performance and functionalities that need to be achieved.

● **Product characteristics:**

- ▶ **Reliability:** For defining the level of reliability that will be required, examine the specific values that need to be met in terms of MTBF (Mean Time Between Failure), MTTR (Mean Time To Repair) and other relevant metrics.
- ▶ **Operating environment:** Clarify the context

of the existing software that is going to be used, as well as the policy on reuse.

(6) Clarify the environment for development, testing and installing the software:

- ▶ Tools used for development;
- ▶ Test environment, tools for testing, test methods, and test data, and their respective availability;
- ▶ Also clarify in advance the constraints faced at the time of installation and other occasions.

in which the product is going to be used (e.g.: temperature, noise, static electricity);

For embedded system, it is not enough just to examine the context in which the system behaves in a normal state. It is also mandatory to examine the context in which the system may behave in an abnormal state.

- ▶ **Maintenance-related environment:** Examine when and how the maintenance should be performed.
- ▶ **Standards/conventions** that need to comply with: Product Liability (PL) law, environmental standards, etc;
- ▶ **Safety:** System requirements related to safety (required level of safety integrity and functionalities to achieve that level) Safety

● **Stakeholders**

- ▶ As far as the users are concerned, identify not only the primary users of the product, but also the secondary users.

● **Product structure:**

- ▶ Type of MPU/MCU to be used;
- ▶ Available memory capacity;
- ▶ Input/output devices;
- ▶ OS and libraries;
- ▶ Division of functional roles provided by the hardware and software.

● **Software to be reused**

- ▶ As regards the reuse of software, bear in mind the granularity of the reuse (also consider the extent of reusable architecture).

Safety : Work related to safety

1.1.2 Clarifying the Functional Software Requirements

Input

- Product Plan Description
- (SY106) System Requirements Specifications
- (SY205) System Architectural Design Description
- (SA104) Safety Requirements Specifications
- Hardware specifications

Outline

Examine the functional requirements that the software must achieve when fully developed, and sort them out in the form of a List of Functional Software Requirements.

Output

- (SW102) List of Functional Software Requirements

See also

- ISO/IEC 9126-1 (Software Quality Model)
- ISO 9241 (Usability)
- ISO 13407 (Human-centered Design Process)
- IEC 61508 (Safety)

Action

Among the functionalities achieved and/or provided by the system, clarify which functionalities need to be achieved by the software, and sort them out in the form of a List of Functional Software Requirements.

In identifying the functional requirements, examine the functionalities that need to be achieved by the software by also taking account of the result of use case analysis.

Precaution

- Points to consider investigating the software functional requirements
 - ▶ Clearly distinguish the functionalities to be achieved by the software from others, by referring to system requirements and the architectural design of the system;
 - ▶ Clarify the hardware functionalities and platform that are related to software functionalities;
 - ▶ Examine which functionalities will enable the target product to be superior over the competing products;
 - ▶ When it seems very unlikely for the software to meet the required level of performance in some functionalities, use the hardware to achieve them.
- ▶ During the use case analysis, create use case scenarios and diagrams, as well as an activity diagram, among others.
- ▶ Double-check that no functional software requirements have been left out.
- ▶ Confirm that abnormal cases are taken into consideration sufficiently.
- Examples of functional software requirements related to safety ◀ Safety
 - ▶ Fail-safe mechanism;
 - ▶ Data protection in the event of failure.

Terminology

* Functional requirements : Functionalities that the software is required to provide to meet the needs of the targeted product and intended users. For instance, capabilities like “Can input the data named xxx” and “Can send emails” are functional requirements.

◀ Safety : Work related to safety

Related Standards

ISO/IEC 9126-1: 2001 Software engineering -- Product quality -- Part 1: Quality Model (JIS X0129-1)

Year of establishment / revision : ISO/IEC 2001, JIS 2003

Purpose: Quality metrics for software products

Scope: Software products

URL: <http://www.iso.org/>

Source in Japan: Japanese Standards Association

Overview

Standard that defines the characteristics of software quality, by dividing them into six categories (Functionality, Reliability, Usability, Efficiency, Maintainability, Portability), and further defining multiple subcharacteristics to each of them.

Software Quality Characteristics

Characteristics	Subcharacteristics
Functionality	Suitability, Accuracy, Interoperability, Security, Functionality Compliance
Reliability	Maturity, Fault Tolerance, Recoverability, Reliability Compliance
Usability	Understandability, Learnability, Operability, Attractiveness, Usability Compliance
Efficiency	Time Behavior, Resource Utilization, Efficiency Compliance
Maintainability	Analyzability, Changeability, Stability, Testability, Maintainability Compliance
Portability	Adaptability, Installability, Co-existence, Replaceability, Portability Compliance

1.1.3 Clarifying the Non-functional Software Requirements

Input

- Product Plan Description
- (SY106) System Requirements Specifications
- (SY205) System Architectural Design Description
- (SA104) Safety Requirements Specifications
- Hardware specifications

Outline

Examine the non-functional requirements that the software must achieve when fully developed, and sort them out in the form of a List of Non-functional Software Requirements.

Output

- (SW103) List of Non-functional Software Requirements

See also

- ISO/IEC 9126-1 (Software Quality Model)
- ISO 9241 (Usability)
- ISO 13407 (Human-centered Design Process)

Action

Clarify the non-functional requirements of the software perceived to be related to the requirements to achieve the system functionalities, and sort them out in the form of a List of Non-functional Software Requirements.

- ▶ Requirements on reliability
- ▶ Requirements on usability
- ▶ Requirements on efficiency
- ▶ Requirements on maintainability
- ▶ Requirements on portability
- ▶ Other non-functional software requirements

Precaution

● Examples of matters examined as the requirements on reliability:

- ▶ Examine the situations when the hardware or the software is forced to behave in unexpected manner in specific operating conditions of the system, and decide on how the system deals with abnormality management;
- ▶ Examine what the software can do to enable the system to continue offering the functionalities that are minimally required even when the system encounters an undesirable situation;
- ▶ Clarify the procedure and methods to recover the

system from abnormal operating mode.

● Examples of matters examined as the requirements on usability:

By bearing in mind that many embedded systems are often used by a large indefinite number of end users, examine what the software should achieve in terms of usability (for instance, standardize the user interface used for the entire system).

Preferably, also examine, from the standpoint of usability, what the software can do to support the hardware in achieving its requirements (for

• Terminology

* Non-functional requirements : Requirements on, such as, efficiency, usability and portability that are required by the software. For instance, capabilities like “Can complete processing within nn seconds”, “Enable the users to operate without relying on manuals”, and “Can be reused” are non-functional requirements.

instance, on the time required for screen display or calculation).

- Examples of matters examined as the requirements on efficiency:

- ▶ Consider the requirements on system performance (e.g.: processing speed, start-up time, response time). Pay particular attention to system hardware constraints and time constraints attributable to external operating environment;
- ▶ Pay attention to resource utilization of the system hardware (e.g.: memory capacity, data size). Also take account of the period the data used in the system continues to exist;

- Examples of matters examined as the requirements on maintainability:

Examine the troubleshooting mechanism that will make it possible to analyze the cause of trouble when it occurs in the field. Also give some thoughts to how the software can support the mechanism to record the event log information. In

addition, examine the maintenance method to be applied (such as, remote maintenance) and how to execute this method.

- Examples of matters examined as the requirements on portability:

Consider also the portability of the software when OS, CPU or peripheral circuits are changed. Consider up-front the independence of software units, based on the assumption that some of the existing software may be reused.

- Examples of matters examined as other non-functional software requirements:

- ▶ Identify the mechanism and/or architecture required for reuse;
- ▶ Requirements on security (e.g.: data encryption, user authentication; anti-virus measures);
- ▶ Interoperability (e.g.: communication protocol);
- ▶ External interface requirements (e.g.: function interface with linking software, communication protocol, user interface);
- ▶ Data definition.

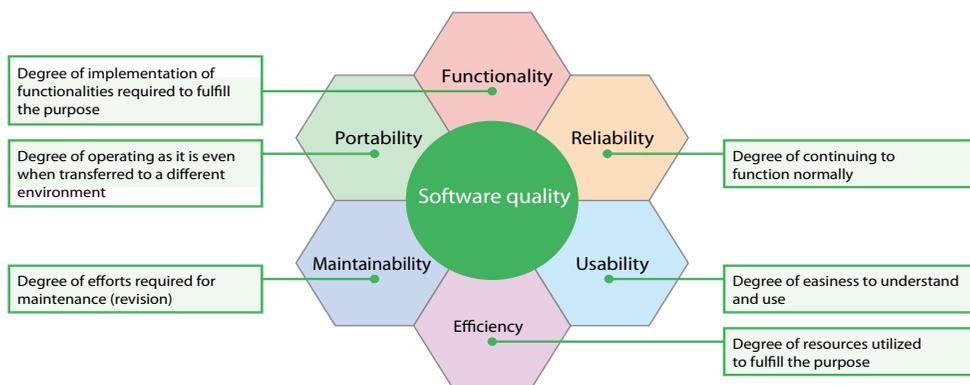


Figure 2.7 ISO/IEC9126 (JIS X 0129-1) Software Quality Characteristics

1.1.4 Prioritizing the Requirements

Input

- (SW102) List of Functional Software Requirements
- (SW103) List of Non-functional Software Requirements

Outline

Decide on the order of priority of software requirements.

Output

- (SW104) Prioritized List of Software Requirements

Action

Prioritize the software requirement items listed in (SW102) and (SW103).

- ▶ The software requirements should be classified preferably into four priority levels: Mandatory / High / Low / Optional. Upon prioritization, keep records on the grounds or reason(s) why these requirements have been respectively set to the specified priority levels.

Precaution

- Points to consider in prioritization:
 - In prioritizing the software requirements, consider the degree of development risk in meeting the individual requirements (including project management viewpoints) and evaluate the requirements from the following perspectives:
 - ▶ Length of time, amount of budget and resources allocated for the development project;
 - ▶ Newly introduced technologies and technical proficiency;
 - ▶ Business needs.
 - Moreover, evaluate the feasibility of high-risk requirements also from the standpoint of long-term product strategies.

1.1.5 Creating the Software Requirements Specifications

Input

- (SW101) List of Constraints
- (SW102) List of Functional Software Requirements
- (SW103) List of Non-functional Software Requirements
- (SW104) Prioritized List of Software Requirements

Outline

Sort out the software requirements and document them orderly in the form of Software Requirements Specifications.

Output

- (SW105) Software Requirements Specifications



Document template sample is available.

See also

IEEE Std 830 (Recommended Practice for Software Requirements Specifications)

Action

Sort out the items listed in (SW101) through (SW104), and document them orderly in the form of Software Requirements Specifications (SW105).

- ▶ Create this document by organizing the set of information gained as outputs while defining the software requirements in a systematic order.
- ▶ Also confirm that the issues raised in reviews (held internally or jointly) are adequately reflected in this document.

Precaution

● Points to keep in mind in documentation:

- ▶ Create software-related manuals prepared for use by the end-users on as-needed basis.
- ▶ Attach the revision history and indicate clearly where have been revised;
- ▶ Clearly indicate who or which organization is

responsible of the created document;

- ▶ Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management; SUP7 Change Management



SWP1.2 Reviewing the Software Requirements Specifications

Confirm that the defined software requirements satisfy the system requirements and other relevant requirements.

1.2.1

Reviewing the Software Requirements Specifications Internally

Input

- Product Plan Description
- (SY106) System Requirements Specifications
- (SY205) System Architectural Design Description
- (SA104) Safety Requirements Specifications
- (SW105) Software Requirements Specifications

Outline

Check whether or not the contents of Software Requirements Specifications cover all what is required to realize the software, and document the findings orderly in the form of an Internal Review Report.

Output

- (SW106) Internal Review Report (on Software Requirements Specifications)
-  Document template sample is available.

Action

Review the Software Requirements Specifications (SW105) internally, based on the following perspectives (1) through (7). Document the findings orderly in the form of an Internal Review Report (SY107) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project:

- Evaluate the validity.
 - ▶ Are the requirements described in the Software Requirements Specifications (hereafter simplified to “this document”) valid in light of system requirements and other relevant requirements?
- Evaluate the feasibility.
 - ▶ Are the requirements described in this document feasible or testable based on the technical capacity held by the organization(s) in charge of software development?
- Evaluate the testability (whether the software requirements can be tested or not).
 - ▶ Can the requirements be tested to check the system operations in the assumed fields of operations?
- Evaluate the operability and maintainability.
 - ▶ Are the concepts and methods applied in updating the software appropriate?
 - ▶ Are appropriate methods to address and solve the defects when they occur being examined?
- Evaluate the traceability.
 - ▶ Are the software requirements respectively tagged with an identifier so that they can be traced to confirm that the subsequent development work is proceeding in accordance with the software requirements defined in this document?
 - ▶ Are the deliverables used as the input source of individual requirements stated explicitly in this document?

(6) Evaluate the consistency.

- ▶ Are there any conflicts in the contents of the relevant deliverables? Are there any conflicting descriptions existing within this document?

(7) Evaluate the integrity.

- ▶ Are all the necessary information on functionalities, performance, design-related

constraints, attributes and external interface , among others, described in this document?

- ▶ Are the software behaviors in all the assumable contexts of system operation defined in this document?
- ▶ Are the requirements described uniquely in a way that cannot be interpreted otherwise?

■ Precaution

● Consider the following points from the standpoint of validity:

- ▶ Are functionalities that satisfy the needs of the end users being provided?
- ▶ Are functionalities that meet the product development strategies being provided?
- ▶ Are the requirements complying with the constraints on, such as, hardware used for the system?
- ▶ Are the requirements in line with system requirements specifications?

● Consider the following points from the standpoint of feasibility:

- ▶ Is the use of hardware being considered in achieving some functionalities instead of the software when it seems very unlikely for the software to meet the required level of performance?
- ▶ Are the constraints on reused software being considered?

● Consider the following points from the standpoint of testability:

- ▶ Is it possible to build the test environment?
- ▶ Is the number of labor units (effort) required for building the test environment not exceeding the total labor units allocated for product development?
- ▶ Regarding the tests using the real machine, can

they be carried out without damaging the real machine?

- ▶ Is it possible to prepare the test environment including the hardware and create the test data?

● Consider the following points from the standpoint of operability and maintainability:

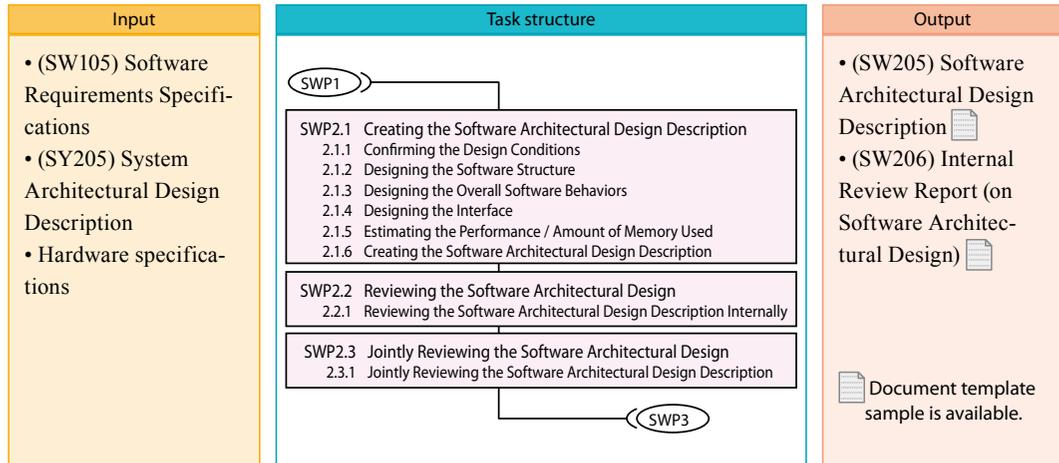
- ▶ Can the modified software be delivered via the network?
- ▶ Is the software maintained through, such as, manual replacement of ROM performed by the maintenance personnel?

● The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).

● Issues found in the early stage of development when the requirements are defined should be addressed as soon as possible to prevent them from growing or leading into bigger problems in the latter half of the development process. Therefore, the information mentioned in the Internal Review Report should be shared with the stakeholders that include the project manager and development team leader, and their consensus should also be built at this early stage.

SWP2 Software Architectural Design

Decide on the architecture (= behavior and structure) of the embedded software to be developed, and create the Software Architectural Design Description.



Description

In this activity:

- (2.1.1) Confirm the functional and non-functional software requirements that must be achieved, based on the Software Requirements Specifications. Moreover, confirm the constraints in achieving the software requirements from the standpoint of how to realize the specifications;
- (2.1.2) (2.1.3) Examine the behavior and structure of the software for achieving the requirements specifications;
- (2.1.4) Design the interface between the functional units that constitute the software;
- (2.1.5) Examine the desirable level of performance and estimate the required memory capacity by taking account of the hardware in which the software will be implemented;
- (2.1.6) Create the Software Architectural Design Description by arranging the results of the above sub-tasks in an orderly format;
- (2.2.1) Review the created Software Architectural Design Description internally, based on the pre-defined check points, and document the outcome of the internal review orderly in the form of an Internal Review Report;
- (2.3.1) Also hold joint review meetings with relevant stakeholders, and document the outcome of the joint review orderly in the form of a Joint Review Report.

Also eye the possibility of using software components or reusing past software assets.

In this guidebook, the embedded software is broken down hierarchically into “functional units” and “program units”, as described below:

Embedded software = Aggregate of functional units

Functional unit = Aggregate of program units

■ Consideration

- ▶ The behavior of embedded system varies depending on the operating environment and conditions (context*). Therefore, analyze the various contexts in which the system is expected to be used, and design the software behaviors based on the result of this analysis.
- ▶ Extract the functional units by localizing and commonalizing the functionalities that are commonly processed.
- ▶ Preferably, the result of architectural designing should be visualized, by utilizing modeling techniques.
- ▶ In designing the architecture, take the required level of safety integrity into consideration, and also examine the following points: **Safety**
 - Clarify the design methods that will be applied and the reason(s) for selecting them;
 - Clarify the prerequisites and constituents of the architecture;
 - Clarify the relationship and interactions between the software and hardware;
 - Avoid ambiguous expressions in describing the architectural design;
 - Identify all the data used by the system and clarify the test methods for checking the designed architecture.

■ <Reference> Techniques and Tools

- ▶ OOD (Object Oriented Design)
- ▶ Modeling techniques
 - UML (Unified Modeling Language), etc
- ▶ SADT (Structured Analysis and Design Technique)

• Terminology

* Context : Context is something written or spoken that immediately precedes or follows a word or passage, background information, or an interrelated condition, situation or circumstance that helps clarify an idea or its meaning. In this guidebook, this term is used, for example in particular section, to mean the purpose, background or strategy of the embedded system to be developed.

Safety : Work related to safety

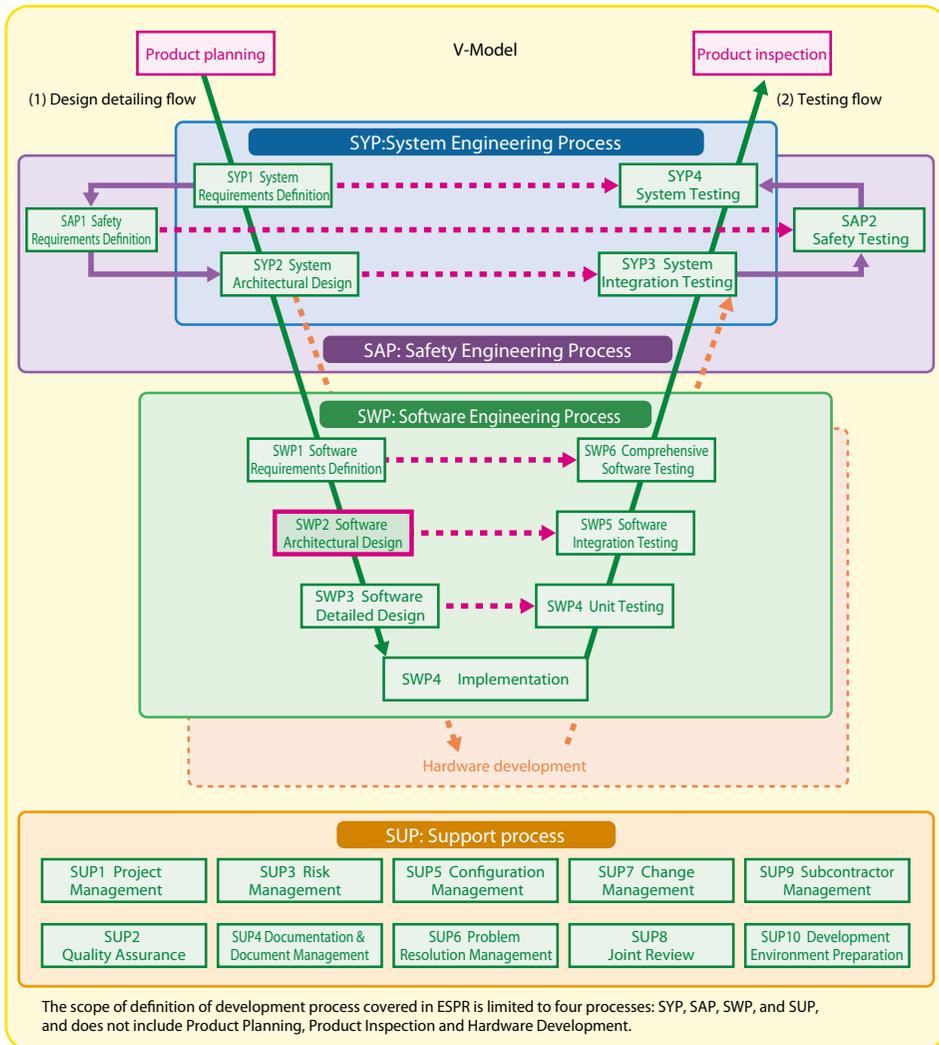


Figure 2.8 V-Model and Development Process (SWP2 Software Architectural Design)



SWP2.1 Creating the Software Architectural Design Description

Examine how to achieve the requirements defined in the Software Requirements Specifications (i.e.: software architecture), and document the findings orderly in the form of Software Architectural Design Description.

2.1.1 Confirming the Design Conditions

Input

- (SW105) Software Requirements Specifications
- (SY205) System Architectural Design Description

Outline

Confirm what the requirements and conditions are in designing the software architecture.

Output

(Design Conditions Confirmation Note)

See also Hardware specifications

Action

In proceeding with software designing, confirm the functional and non-functional software requirements, and the constraints that serve as prerequisites described in the Software Requirements Specifications (SW105).

- (1) Confirm the functional software requirements.
 - ▶ Based on the contents of the functional software requirements described in the Software Requirements Specifications, identify the functionalities required to be achieved by the software.
- (2) Confirm the non-functional software requirements.
 - ▶ Confirm the non-functional software requirements described in the Software Requirements Specifications.
- (3) Confirm the constraints.
 - ▶ Re-examine the software functionalities

described in the Software Requirements Specifications, and confirm the constraints, including those listed below, from the standpoint of designing an architecture that will enable the software to achieve the required functionalities:

- Conditions for software creation (type of OS and language that are going to be used, among other)
- Hardware conditions (interrupt method, etc)
- Performance conditions, operating environment, etc
- Safety requirements Safety

Precaution

- Focus on confirming the following points in particular:
 - ▶ Specific conditions on performance-related requirements and specifications on abnormal and exceptional behaviors;
 - ▶ Memory capacity, and types and capacity of external storage media
 - ▶ Types, specifications and processing capacity of hardware equipment and devices;
 - ▶ Result of the investigation on the use of existing software (including OS).
- Also examine the system requirements pertaining to security;
- Confirm the system requirements pertaining to safety (required level of safety integrity, and the functionalities to achieve the required safety integrity level). Safety
- Check whether there are any information missing and/or TBD (To Be Determined) matters still remaining in the Software Requirements Specifications.

Safety : Work related to safety

2.1.2 Designing the Software Structure

Input

- (SW105) Software Requirements Specifications
- (SY205) System Architectural Design Description

Outline

Design the software structure and functionalities provided by each functional unit.

Output

- (SW201) Software Structure Design Description
- (SW202) Functional Unit Design Description

See also
Hardware specifications

Action

Examine how the software should be broken down into functional units in order to achieve the software requirements as specified in the Software Requirements Specifications, and sort out the functionalities to be provided by each functional unit.

(1) Extract the functional units.

- ▶ Clarify the functionalities to be achieved by the software, and extract the functional units by localizing and commonalizing the functionalities that are commonly processed. In extracting the functional units, also give adequate attention to non-functional software requirements, including

those listed below:

- ▶ Reliability (on recovery, data assurance, etc)
- ▶ Maintainability (on tracing the defects, etc)

(2) Refine the functional units.

- ▶ Refine each functional unit in depth to the level (granularity) that is specific enough to work on the detailed design of the software.

Precaution

● In designing the software structure, give attention to the following points:

- ▶ Maintainability, development efficiency, test efficiency, reliability, scalability and safety;
- ▶ Localization/encapsulation of functionalities based on the consideration given to the cost spent on changes/modifications and quality management;
- ▶ Componentization/commonalization based on the consideration given to development efficiency/memory utilization;
- ▶ Development of built-in event log/retrieval mechanism for debugging/ testing/maintenance;
- ▶ Development of built-in mechanism for modifying/updating the software after it is shipped out;
- ▶ Development of built-in fail-safe mechanism.

● Method of proceeding with the design:

- ▶ In designing the software architecture, refine each functional unit in depth to the level (granularity) that is specific enough to assign engineers to work on the detailed design of the software;
- ▶ Work on the structural design in conjunction with behavioral design and interface design.

● Other points to consider:

- ▶ Create a structural drawing that shows how the functional units are positioned and co-related (functional unit association chart);
- ▶ Clearly compartmentalize the core functional units of the software, by bearing in mind the future possibility of developing new versions and variants of the product in series.

2.1.3 Designing the Overall Software Behaviors

Input

- (SW105) Software Requirements Specifications
- (SY205) System Architectural Design Description

Outline

Design the overall software behaviors.

Output

- (SW203) Software Behavioral Design Description

See also
Hardware specifications

Action

Study and sort out the dynamic behaviors of the system including the hardware.

- ▶ Clarify the software behaviors that drive the system behaviors.
- ▶ Clarify the context that becomes the prerequisite for software behavior, and examine the scenarios and sequences described in the Software Requirements Specifications. Also consider the smooth sequence of system behaviors driven by specific software behaviors. Especially in case of embedded software, take account of the following matters:
 - Concurrent processing: Hardware interrupt handling, task priority, concurrent processing / task state transition, etc;
 - Abnormalities / exceptions: Overload, hardware failure, etc;
 - Flow of data processing: From input to output.

Precaution

- Express the behaviors and controls in ways that are easy to understand, including the concurrent use of visuals.
 - ▶ Sort out the behaviors by using state transition models and/or other methods.
- Bear in mind the following points when designing the behaviors of embedded software:
 - ▶ Performance: Concurrent processing and task priority, SRAM / cache allocation;
 - ▶ Real-time processing: Interrupt control, hardware interrupt level setting;
 - ▶ Multi-tasking: Exclusive control (allocating / releasing / racing resources);
 - ▶ Treatment of abnormalities / exceptions: Counteractions against DoS attacks, recovery operation, data violation caused by noise, malfunctioning of MPU, chattering;
 - ▶ Memory layout: Resident or non-resident, ROM or RAM;
 - ▶ Multi-processor structure: Load balancing, exclusive resource control;
 - ▶ Clarify the internal state of each functional unit, definition of relational states between the functional units and state transitions;
 - ▶ Time constraints of each functional unit.

2.1.4 Designing the Interface

Input

- (SW105) Software Requirements Specifications
- (SY205) System Architectural Design Description

Outline

Design the interface between functional units.

Output

- (SW204) Software Interface Design Description

See also

Hardware specifications

Action

Design the interface between the functional units that structure the software.

- (1) Design the memory layout.
 - ▶ Design the layout of the entire memory.
- (2) Design the memory layout in detail by specifying the memory spaces and areas.
 - ▶ Decide on the specific memory areas to be used by the software.
 - Decide on the name / structure / size of each memory area.
 - Define the name / location / type / size / meaning / default value / access source of each fields in the memory area.
- (3) Design the interface between the functional units.
 - ▶ Design the calling sequence, parameters, callback information, etc.
- (4) Centralize and give logical values to common information.
 - ▶ Centralize and give logical values to information that have to be heavily modified when change arises.
 - Table offset
 - Common constants, etc

Precaution

- Points to consider in interface designing
 - ▶ Extract the abnormal / exceptional codes and messages as early as possible. Sort them out and create a table that lists them systematically.
 - ▶ Clarify the read/write sources of common data that need to be processed exclusively.
 - ▶ Regarding the common resources, clarify the timing to allocate / release them.
 - ▶ Regarding the bit structure, take account of its scalability.

Input

- (SY205) System Architectural Design Description
- (SW201) Software Structure Design Description
- (SW203) Software Behavioral Design Description
- (SW204) Software Interface Design Description

Outline

Estimate the performance and the amount of memory that would be used by examining the various conditions that may affect them.

Output

- (Materials for estimating the performance)
- (Materials for estimating the amount of memory used)

Action**(1) Estimate the performance.**

- ▶ Estimate the performance as accurately as possible by considering the worst-case conditions.
 - For critical elements, use the data based on actual measurement as the reference.
 - Calculate the throughput of the entire system at times of overload.

(2) Estimate the amount of memory used.

- ▶ Include all possible memory usages in the

estimate, including the stack at times of overload.

- Both the code segment for programs, and the data segment used, such as, for tables and buffers
- Estimate from a dynamic standpoint
- ROM, RAM (including SRAM), cache and all other available memory devices.
- ▶ If necessary, also include other hardware resources like hard disks into the estimate.

Precaution**● Points to consider when estimating the performance:**

- ▶ Characteristic properties of the hardware (processing / response speed of input / output devices, memory wait, etc)
- ▶ Time taken for start-up (take the worst-case conditions into consideration)
- ▶ OS processing time, etc

Carefully examine the processing and response time (speed) achieved by the software, including the methods of achieving the desirable

performance levels, by bearing in mind the impact on users' experience (such as, in terms of usability).

● Points to consider when estimating the amount of memory used:

- ▶ Degree of multi-tasking, stack size based on the number of nests, OS area (task control table, etc)
- ▶ Use the result of size-based estimation as the basis for calculating the estimate memory space used for codes.

2.1.6 Creating the Software Architectural Design Description

Input

- (SW201) Software Structure Design Description
- (SW202) Functional Unit Design Description
- (SW203) Software Behavioral Design Description
- (SW204) Software Interface Design Description
- (Materials for estimating the performance)
- (Materials for estimating the amount of memory used)

Outline

Sort out all the matters pertaining to software architectural design, and document them orderly in the form of Software Architectural Design Description.

Output

- (SW205) Software Architectural Design Description 

 Document template sample is available.

Action

Sort out the information outputted in (SW201) through (SW204), and document them orderly in the form of Software Architectural Design Description (SW205).

- ▶ Sort out the information outputted in various materials in the course of designing the software architecture, and document them systematically.
- ▶ Confirm that the created document adequately reflects all the points indicated in internal and joint reviews.

Precaution

● Regarding the reference materials:

- ▶ Keeping the documents that can be used as evidences for proving the validity of the design (materials for estimating the performance and the amount of resources used, documents that describe the control methods, etc) and documents for explaining the perspectives applied in designing (i.e.: design policy) as separate materials would help smoothen the efforts to deal with changes in specifications and software-related problems when they occur.
- ▶ It is desirable to perform version management that will enable the readers to know which part of the document has been revised or added, and

attach the list of past versions.

● Points to keep in mind in documentation:

- ▶ Sort out the names of functional units, tasks / processes systematically.
- ▶ Attach the revision history and indicate clearly where have been revised;
- ▶ Clearly indicate who or which organization is responsible of the created document;
- ▶ Ensure that the created document is managed properly by performing configuration management and change management.
Related processes: SUP5 Configuration Management; SUP7 Change Management



SWP2.2 Reviewing the Software Architectural Design

Review the designed software architecture to check whether it fully satisfies the system and software requirements or not.

2.2.1 Reviewing the Software Architectural Design Description Internally

Input

- (SW105) Software Requirements Specifications
- (SW205) Software Architectural Design Description
- (SW201) Software Structure Design Description
- (SW202) Functional Unit Design Description
- (SW203) Software Behavioral Design Description
- (SW204) Software Interface Design Description, etc

Outline

Check whether or not the software architectural design satisfies the software requirements and the behaviors of the equipment / system meet the system specifications, and document the findings orderly in the form of an Internal Review Report.

Output

- (SW206) Internal Review Report (on Software Architectural Design) 
-  Document template sample is available.

See also Hardware specifications, etc

Action

(1) Check whether the contents of the Software Architectural Design Description (SW205) are appropriate or not. Document the findings orderly in the form of Internal Review Report (SW206), including the issues raised during the review as well as the personnel in charge of handling these issues, and distribute the report to the relevant stakeholders.

- ▶ Check whether the functional units that structure the software can collectively realize the system and software requirements correctly or not. Some of the key check points include the following:
 - Definiteness and validity of the functionalities provided by the functional units;
 - Definiteness and validity of the behaviors realized collectively by the functional units;
 - Definiteness and validity of the interface between functional units.
- ▶ Evaluate the software architecture design from the standpoint of whether the non-functional software requirements are adequately reflected

in the design or not, and whether the software, when developed, will be fully capable of being operated, tested and maintained or not.

- Whether the designed software architecture is easy to understand or not by taking account of the intended users' skill levels, and whether safety matters are also adequately taken into consideration or not.
- Whether the software's operating conditions (performance, peak load, long continuous operation) are taken into account or not.
- Whether testability / maintainability are taken into account or not.

(2) Envision the detailed design of the functional units.

- ▶ Evaluate the software architectural design of the individual functional units by taking account of the following criteria:
 - Validity of the functionalities, interface and behaviors;
 - Low-levelness and feasibility of the detailed design;

- Compliance with design standards (design methods / annotations / terminology / readability / notation);
- Adequate investigation of existing software / commercial software / open source software, in case they are planned to be used;
- Easiness to transfer to a different platform

(portability) and extend the functionalities (scalability).

- (3) Check whether the respectively designed portions of software architecture can be mapped with corresponding software requirements or not (traceability).

■ Precaution

- Points to bear in mind when holding meetings to review the software architecture:

- ▶ When to hold : Keep in mind the importance of avoiding extensive re-designing necessitated by review feedbacks when setting the meeting schedule. Carry out reviews at interim check gates when the design is partially completed rather than waiting for the entire design to be completed.
- ▶ What to review: Mainly focus on the critical areas and areas that have been designed by new participants.
- ▶ Who to call as reviewers: Assign members that have high technical knowledge and skills. If necessary, consider inviting hardware designers as well.
- ▶ What to record : It is desirable to comply with the project standards and record useful information that contributes to the improvement of design quality (date & time of review meetings, modules that have been evaluated,

the names of reviewers, the problems and issues raised in the meetings, their possible causes and solutions that have been examined, among others).

- The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).
 - ▶ Issues found in the early stage of development when the architecture is designed should be addressed as soon as possible to prevent them from growing or leading into bigger problems in the latter half of the development process. Therefore, the information mentioned in the Internal Review Report should be shared with the stakeholders that include the project manager and development team leader, and their consensus should also be built at this early stage.

Review is the most convenient and reliable way of ensuring software quality.

Check the problematic areas, using quantitative data gained in the course of designing and implementation.

Complex areas
Areas where there is considerable delay in development
Areas created by inexperienced developers
Areas where the interface between the software and peripherals is intricate
Areas where the usage is not clearly defined

Check the key areas, using past incidents (defects and failures) as the reference.

Areas in similar products where bugs have been found
Areas where the customer frequently use
Areas where handling of abnormalities is important

Reviews provide the stakeholders the opportunity to exchange information and get assurance through face-to-face meetings

Figure 2.9 Check Points in Review Sessions



SWP2.3 Jointly Reviewing the Software Architectural Design

Evaluate the designed software architecture among the stakeholders by holding joint review meetings to check whether the architecture is fully meeting the system and software requirements or not.

2.3.1 Jointly Reviewing the Software Architectural Design Description

Input

- (SW205) Software Architectural Design Description
- (SW206) Internal Review Report (on Software Architectural Design)

Outline

Confirm among the stakeholders the validity of the design for achieving software requirements by holding joint review* meetings to check the contents of the Software Architectural Design Description.

Output

- (SU801) Joint Review Records (on Software Architectural Design) 
 - (SW207) Software Architectural Design Joint Review Report 
-  Document template sample is available.

See also

- (SY106) System Requirements Specifications
- (SY205) System Architectural Design Description
- (SW105) Software Requirements Specifications

Action

- (1) Based on the project plan, confirm among all the members involved in the development project that the software architecture design is valid for the equipment / system in which the software is embedded.
 - ▶ In the review, take account of the following points in particular:
 - Feasibility of the software requirements, and clarity of the changes made to and /or constraints of the requirements;
 - Responsiveness to actual operation, including maintenance and security;
 - Smooth transferability to subsequent process phases;
 - Compliance with the project plan, standards and conventions, including hardware (on project schedule, documents, records on quality, techniques, methods, etc)
- (2) Hold a joint review when an issue that requires coordination with stakeholders arises, including the following cases:
 - ▶ Hold the joint review when there is a need to work together with members of other development groups to resolve project-wide problems that occurred or have been identified while designing the software architecture;
 - ▶ Focus on resolving these problems as early as possible to avoid delays in the designing phase and the necessity of re-designing.
- (3) Based on the review records, create a document called Joint Review Report.
 - In the Joint Review Report, also explicitly state the issues and their possible solutions raised in the review meetings, and distribute this report to the relevant members of the development project.

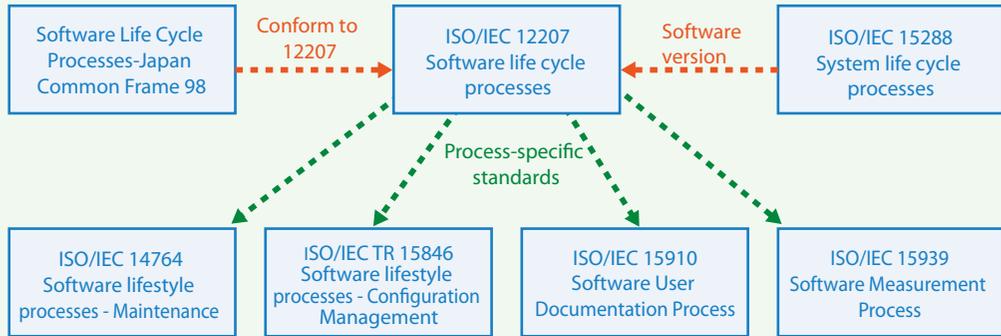
Related processes: SUP8 Joint review

Terminology

* Joint review : A form of review held by the members of the development project to check appropriate outcome has been produced or not at each milestone of the development process from both the technical and administrative standpoints. Joint review is an opportunity not only for the engineers in charge of creating the deliverables to cross-check them, but also for other stakeholders involved in the product development to participate and check the deliverables from multiple perspectives.

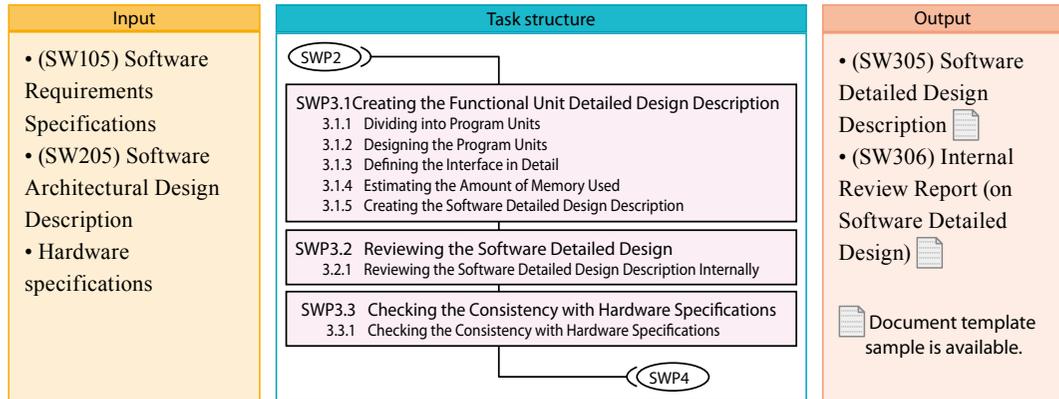
Related Standards

Standards Related to Development Process



SWP3 Software Detailed Design

By dividing the functional units defined in the Software Architectural Design into program units, design the detailed behaviors and logical structure of the software.



Description

In this activity:

- (3.1.1) To enable the embedded software to achieve the functionalities provided by the system to be developed, divide the software into several implementable portions (called the “program units” hereafter), based on the software requirements specifications and software architectural design;
- (3.1.2) Clarify what each program unit processes and the logic applied to that processing;
- (3.1.3) Define how the program units are interrelated respectively, and decide on the interface between the program units;
- (3.1.4) Estimate the amount of memory used, from the standpoint of implementation;
- (3.1.5) Sort out the findings of the above and document them in the form of Software Detailed Design Description;
- (3.2.1) Review the created Software Detailed Design Description, based on the pre-defined check points, and document the outcome of this review orderly in the form of an Internal Review Report;
- (3.3.1) Moreover, check among the project members the consistency of software detailed design with the hardware specifications, and document the findings in the form of Report on Hardware Specifications Consistency Check Result.

Consideration

- ▶ In case of embedded system, examine the details of the program units by giving sufficient attention to the data determined by the operating environment of the system (such as, the ambient temperature of the air conditioning system).
- ▶ Also closely examine the hardware environment and conditions in which the software is designed to operate.
- ▶ Give attention to the information on matters coordinated with the hardware design group.

- ▶ In commencing the work to create the detailed design of the software, confirm that the safety requirements of the system, the software

architectural design, and the method of assuring the safety of the software are all documented.

Safety

<Reference> Techniques and Tools

- ▶ OOP (Object Oriented Programming)
- ▶ SADT (Structured Analysis and Design Technique)

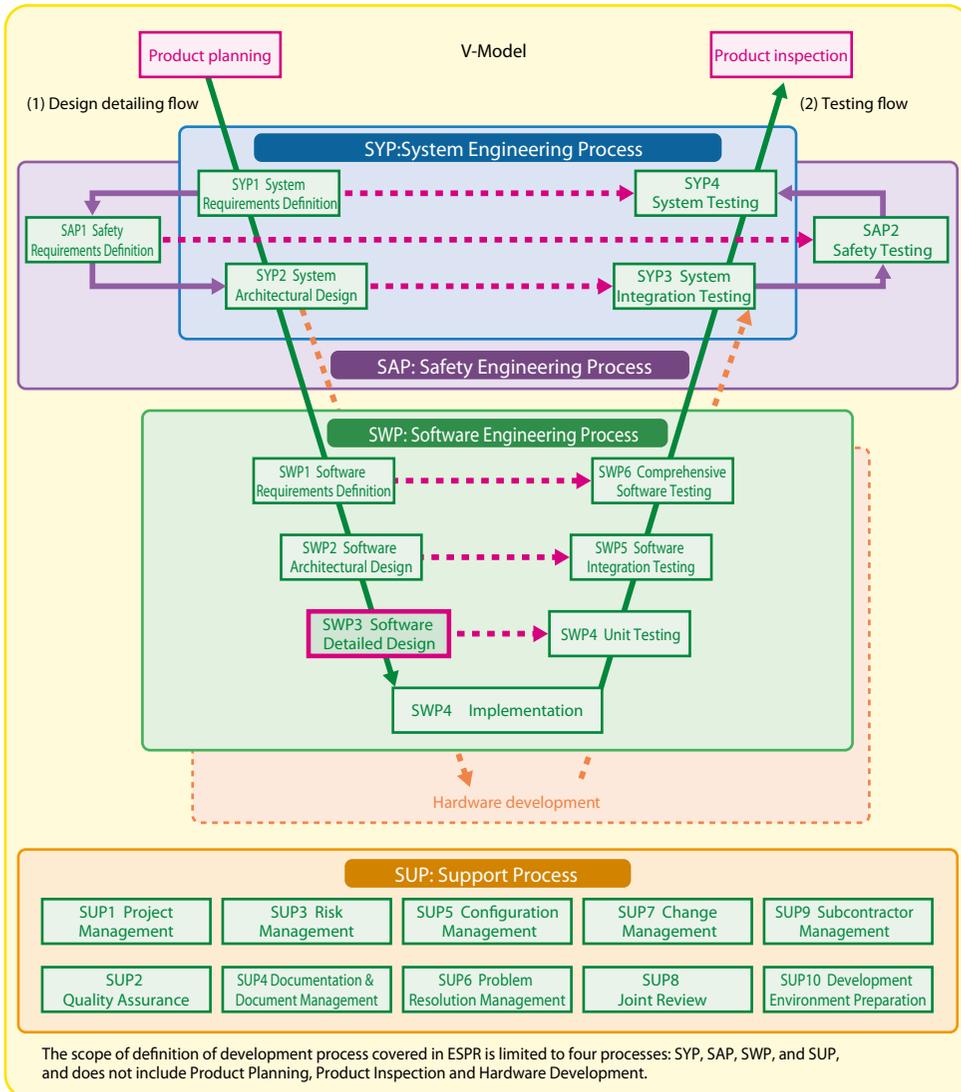


Figure 2.10 V-Model and Development Process (SWP3 Software Detailed Design)

Safety : Work related to safety



SWP3.1 Creating the Functional Unit Detailed Design Description

Based on the specifications of the functional units described in the Software Architectural Design Description, design the software in more specific details by dividing the software into implementable low-level portions (program units) and defining the design of these units, and document the designed details orderly in the form of Software Detailed Design Description.

3.1.1 Dividing into Program Units

Input

- (SW105) Software Requirements Specifications
- (SW205) Software Architectural Design Description

Outline

Divide each functional unit into program units.

Output

- (SW301) Program Unit Functional / Structural Design Description

Action

Divide each functional unit into program units, and define the structure and functionalities of each program unit.

- ▶ Program unit is the lowest unit level that is implemented, compiled and tested.

Precaution

- Points to consider when dividing the functional units into program units:
 - ▶ Focus on the data flow (flow to process data);
 - ▶ Identify the functionalities that are common to multiple program units, and define them as common functionalities;
 - ▶ Take note of the independence (encapsulation) of each program unit, and the binding degree between each other;
 - ▶ Testability, maintainability, scalability;
 - ▶ Easiness to review;
 - ▶ Reusability;
 - ▶ Be careful not to leave out any processing, and prevent any processings from causing conflicts.

3.1.2 Designing the Program Units

Input

- (SW205) Software Architectural Design Description
- (SW301) Program Unit Functional / Structural Design Description

Outline

Define what each program unit processes in detail that is as low-leveled as possible to implement.

Output

- (SW302) Program Unit Design Description

See also

Hardware specifications

Action

(1) Define what each program unit processes in detail that is as low-leveled as possible to implement.

Matters to define in detail include the following:

- ▶ Methods, timing and set values to control the hardware;
- ▶ Argument values for OS system call and utilities (like generic libraries and common functionalities);
- ▶ Conditions and techniques for high-speed processing (when necessary);
- ▶ State management (for managing the states that determine the software behavior based on event

inputs)

- ▶ Error processing
- ▶ Resource definition (on resources used in the program units)
- ▶ System initialization processing

(2) Examine the detailed definition of program unit functionalities used for analyzing defects.

Matters to design in detail include the following:

- ▶ State after controlling the hardware (especially the information when error occurs)
- ▶ Means of confirming the software execution states

Precaution

- ▶ Extract the values that are commonly used, and specify the concrete numerical values (used as #define values). Also specify the concrete values for items defined in the Software Architectural Design Description. Examples of commonly

used values are as follows:

- Memory size (tables, buffers, etc)
- Error values
- Conditions for compiling
- ▶ Take account of events and exclusive control

• Terminology

* Product line : Refers to software engineering techniques for developing new software efficiently by developing small software segments (domains) and combining the existing domains. Product line is effective when there is a need to develop numerous software that share similar specifications.

Safety Work related to safety

between the program units that behave asynchronously.

- ▶ Based on the requirements definitions and architecture design, consider the future possibility of developing the product in series (new versions / variants), and clearly distinguish the program units that structure the core portion of the software with program units that correspond to product variants.
 - Product line*

- ▶ Regarding the program units for realizing the user interface, take account of quality aspects
- ▶ like user-friendliness (consider designing these program units in such a way that can prevent the users from to use making erroneous operations) .
- ▶ Take account of required safety integrity levels (on accuracy, verifiability, fault tolerance, etc).

Safety

3.1.3 Defining the Interface in Detail

Input

- (SW205) Software Architectural Design Description
- (SW302) Program Unit Design Description

Outline

Define the interfaces between the functional units and the interfaces between the program units in detail that are as low-level as possible to implement.

Output

- (SW303) Program Unit Interface Design Description

Action

Define the interfaces between the functional units that structure the software and the interfaces between the respective program units in detail.

- (1) Define the interfaces between the functional units in detail that are as low-level as possible to implement.

- ▶ Define the specifications of the interfaces defined in the architecture design in detail that are as low-level as possible to implement. Clarify the structure, size, meaning and default values of the following:
 - Inputs into functional units
 - Outputs from functional units
 - Memory to be referenced / configured (tables, buffers, etc)

- (2) Design the interfaces between the program units.

- ▶ Define the interfaces between the program units in detail that are as low-level as possible to implement. Clarify the structure, size, meaning and default values of the following:
 - Inputs into program units
 - Outputs from program units
 - Memory to be referenced / configured (tables, buffers, etc)

Precaution

- ▶ Prevent the memory (tables, buffers, etc) from causing conflict, due to, such as, bad timing

when used for processing interrupts.

3.1.4 Estimating the Amount of Memory Used

Input

- (SW205) Software Architectural Design Description
- (SW301) Program Unit Functional / Structural Design Description
- (SW302) Program Unit Design Description
- (SW303) Program Unit Interface Design Description

Outline

Estimate in detail how much memory may be necessary, and judge from the estimated amount whether the software can be implemented or not.

Output

- (SW304) Amount of Memory Used (Notes)

Action

Estimate the memory capacity necessary to develop the software as designed.

- (1) Estimate in detail how much memory may be necessary.
 - ▶ Estimate in detail the amount of memory expected to be used. Calculate the estimate amount of memory that may be necessary for each memory type (ROM, RAM, stack area, etc).
- (2) Check whether the software can be implemented or not
 - ▶ Check whether the software can be implemented

or not, based on the estimated amount of memory that may be necessary for each memory type. When the estimated amount exceeds the capacity of the physical memory intended to be installed, reconsider the software detailed design, or see if the physical memory capacity can be enlarged by consulting with those in charge of hardware development.

Precaution

- Be careful of the following points when estimating how much memory may be necessary:
 - ▶ Is the estimated memory capacity taking future possibilities of specification changes and additions into consideration? (Is the capacity

adequate (with enough free space) to handle them if they occur?)

- ▶ Has the environment to analyze defects (area for emulator, etc) been taken into consideration as well?

Input

- (SW301) Program Unit Functional / Structural Design Description
- (SW302) Program Unit Design Description
- (SW303) Program Unit Interface Design Description
- (SW304) Amount of Memory Used (Notes)
- (SW309) Report on Hardware Specifications Consistency Check Result (after the indicated matters are reflected)

Outline

Sort out all the matters pertaining to software detailed design, and document them orderly in the form of System Software Detailed Design Description.

Output

- (SW305) Software Detailed Design Description 

 Document template sample is available.

Action

Sort out the information outputted in (SW301) through (SW304), and document them orderly in the form of Software Detailed Design Description (SW305).

- ▶ Sort out the information outputted in various materials in the course of designing the software in detail, and document them systematically.
- ▶ Confirm that the created document adequately

reflects all the points indicated in internal reviews and reviews for checking the consistency with hardware specifications.

Precaution

● Points to keep in mind in documentation:

- ▶ Attach the revision history and indicate clearly where have been revised;
- ▶ Clearly indicate who or which organization is responsible of the created document;

- ▶ Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management; SUP7 Change Management



SWP3.2 Reviewing the Software Detailed Design

Confirm that the detailed software design is at an implementable level.

3.2.1 Reviewing the Software Detailed Design Description Internally

Input

- (SW205) Software Architectural Design Description
- (SW305) Software Detailed Design Description

Outline

Check whether or not the software detailed design is reflecting the architectural design requirements adequately and is also implementable, and document the findings orderly in the form of Internal Review Report.

Output

- (SW306) Internal Review Report (on Software Detailed Design) 
-  Document template sample is available.

Action

(1) Review the contents of Software Detailed Design Description from the following perspectives:

- ▶ Is the software divided into units correctly? Are the processing contents of each unit clearly defined?
- ▶ Is the information on interfaces between units consistent?
- ▶ Is there any issue regarding the relationship

between individual units and hardware?

(2) Document the findings of the above check points orderly in the form of an Internal Review Report (SW306) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

● Other points that should be reviewed include:

- ▶ Validity of the program unit structure;
 - ▶ Consistency between functional units and program units;
 - ▶ Consistency of the interfaces between functional units;
 - ▶ Consistency of the interfaces between program units;
 - ▶ Structure and exclusive control of the memory (tables, buffers, etc) and wasteful areas within the memory;
 - ▶ Various set values, argument values (for OS system calls, generic libraries, etc)
 - ▶ Method of achieving high-speed processing (if necessary);
 - ▶ Methods and set values to control the hardware;
 - ▶ Effective use of hardware functionalities (Has high-speed processing being examined?);
 - ▶ Infinite loop, deadlock (Check whether there are any conditions that may lead to their occurrence);
 - ▶ Interrupt handling (nested interrupts, exit routine, etc);
 - ▶ Are all the information necessary for implementation covered in Software Detailed Design Description without fail?
- The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).



SWP3.3 Checking the Consistency with Hardware Specifications

Check whether or not the software detailed design is consistent with the specifications of both the software and hardware.

3.3.1 Checking the Consistency with Hardware Specifications

Input

- (SW205) Software Architectural Design Description
- (SW305) Software Detailed Design Description

Outline

Check whether or not the software detailed design is consistent with the specifications of both the software and hardware, and document the findings orderly in the form of Report on Consistency Check Result.

Output

- (SW307) Report on Hardware Specifications Consistency Check Result

Action

Lay out the specifications of both the hardware and software, and check whether they are consistent with each other or not. Document the findings orderly in the form of Report on Hardware Specifications Consistency Check Result (SW307) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

- ▶ Check if there have been any parts in the specifications of the software and hardware found newly to be unclear or misinterpreted during the software and hardware design phase, to prevent the need for redesigning arising in subsequent process phases. Matters that should be checked at this stage include:
 - Hardware initialization procedure;
 - Timing
 - Set values of the hardware.
- ▶ Inconsistencies between the software and hardware existing in the detailed design level will lead directly to problematic conditions of the system. Therefore, there is a need to distribute the Report on Hardware Specifications Consistency Check Result created here, not only to those involved in software development, but also to those engaged in hardware development, and gain their acknowledgment.

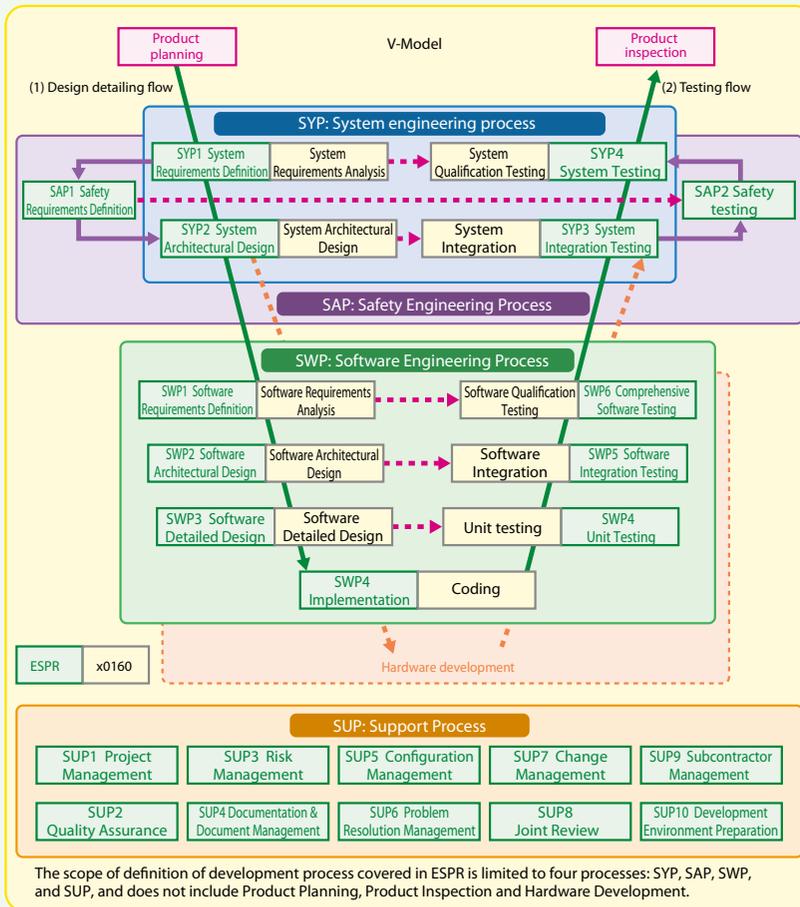
Related Techniques

V-Model

The relationship between the activities described in this guidebook can be visually represented in the form of a V-shaped model (referred to as simply as "V-model" hereafter).

V-Model divides the development process into two streams, the design detailing flow (see arrow (1) in the diagram below) where activities to break down the requirements into implementable units and implement the software are carried out, and the testing flow (see arrow (2) in the diagram below) where activities to check whether the software functions correctly as required or not are carried out, and shows the relationship between the activities carried out in the former stream (design detailing flow) and the corresponding activities carried out in the latter stream (testing flow).

During the design detailing flow, documents (specifications and design descriptions) are created as deliverables of the activities performed in this stream. During the testing flow, the software is tested to check whether it functions correctly or not by referencing the documents created as deliverables of corresponding activities in the former stream. Therefore, when creating these documents, it is important to describe the functional and non-functional requirements clearly in them, among others, with an awareness that that they will be used in the subsequent testing flow as the reference sources for checking the functionalities of the software.

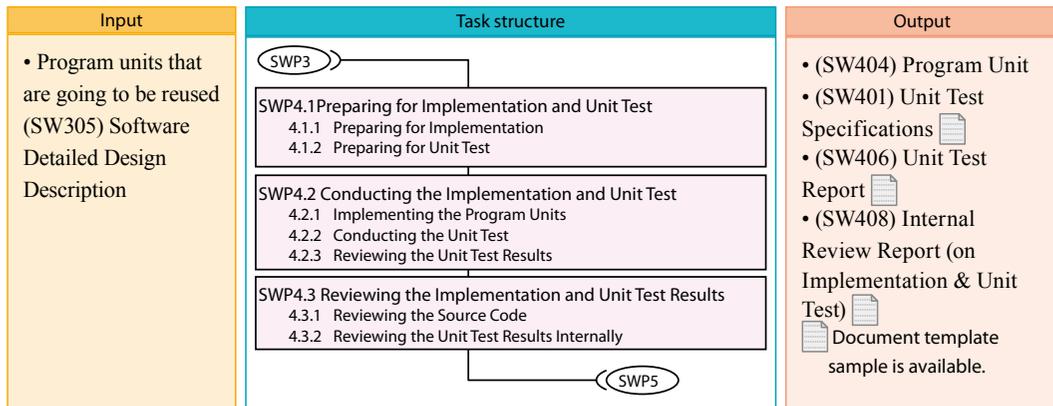


The scope of definition of development process covered in ESPR is limited to four processes: SYP, SAP, SWP, and SUP, and does not include Product Planning, Product Inspection and Hardware Development.

Scope covered by this guide (Ver.2.0)

SWP4 Implementation & Unit Testing

Implement the respective program units that structure the software and test the behaviors of the software at unit level.



Description

In this activity:

- (4.1.1) Prepare the development environment for implementing embedded software program units, and existing program units that are intended to be reused, among others;
- (4.1.2) Prepare for testing the implemented program units at unit level;
- (4.2.1) (4.2.2) (4.2.3) Implement the program units according to the Software Detailed Design Description and conduct the Unit Test;
- (4.3.1) (4.3.2) Moreover, review the source code of the program units that have been fully implemented and the test results gained from the Unit Test, based on the pre-defined check points, and document the outcome of this review orderly in the form of an Internal Review Report.

Consideration

- | | |
|--|--|
| <ul style="list-style-type: none"> ▶ For implementation, pre-define the coding practices and rules to be adopted. ▶ Bear in mind the importance of preparing the | <p>test environment for the Unit Test according to pre-defined schedule, since the preparatory work tends to be delayed.</p> |
|--|--|

<Reference> Techniques and Tools

- | | |
|---|---|
| <ul style="list-style-type: none"> ▶ Programming tools (compiler, linker) ▶ Source code management tools ▶ Debugging tools (debugger, in-circuit emulator (ICE)) ▶ Test support tools ▶ Static and dynamic source code check tools | <ul style="list-style-type: none"> ▶ Quality metrics (source code) measurement tools ▶ Code clone detection tools (technologies used for revising source codes, refactoring, quality evaluation, detection of code plagiarism, etc) |
|---|---|

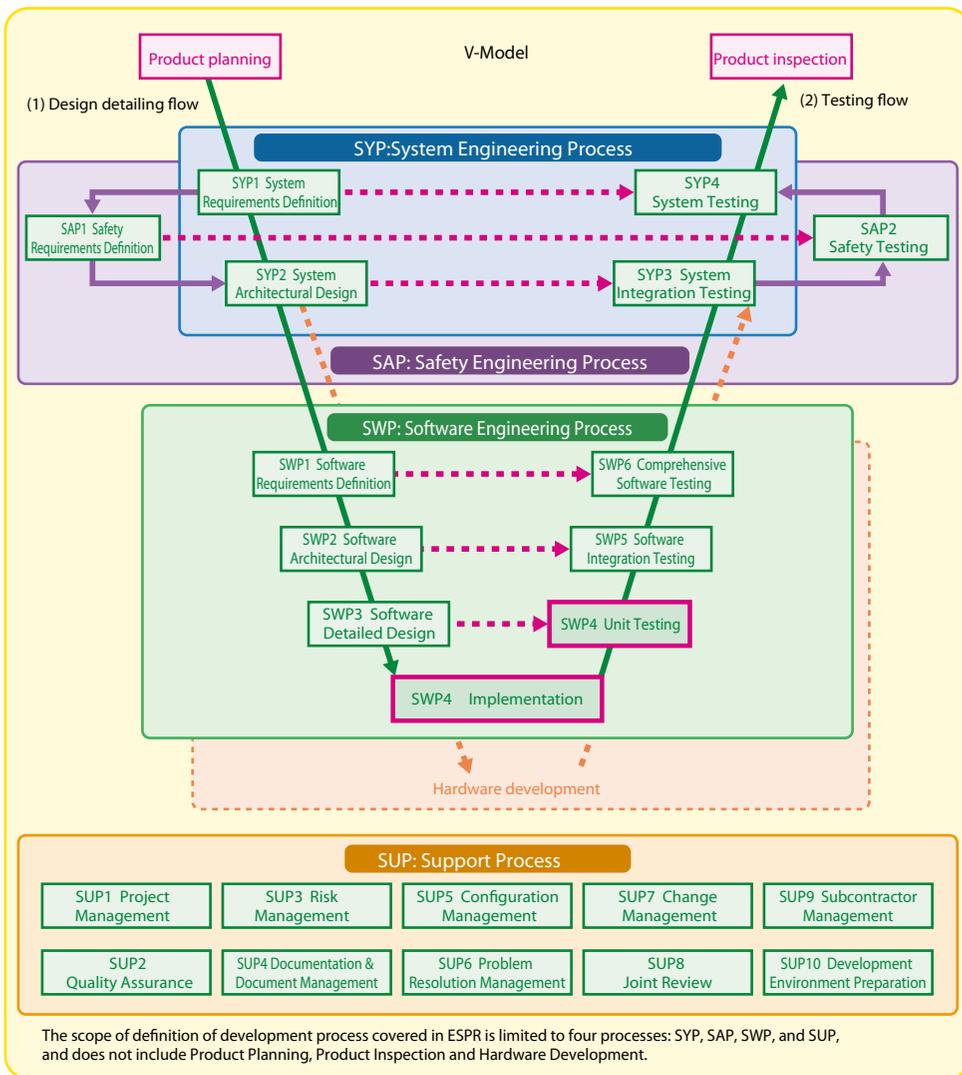


Figure 2.11 V-Model and Development Process (SWP4 Implementation & Unit Testing)



SWP4.1 Preparing for Implementation and Unit Test

Prepare for the implementation of program units as defined in the Software Detailed Design Description as well as for the Unit Test.

4.1.1 Preparing for Implementation

Input

Program units intended to be reused*

Outline

Be ready to implement the program units by preparing the development environment for implementing the program units and the existing program units that are intended to be reused, among others;

Output

• (SU1002) Software Development Environment

Action

(1) Prepare the program units that are intended to be reused.

- ▶ In case of using program units that have already been developed, after making them available,

check that they are in the usable state.

(2) Prepare the development environment.

- ▶ Prepare an environment for development that is suitable for implementing program units.

Precaution

● Regarding the program units intended to be reused, also check the following points:

- ▶ Version;

- ▶ Quality (already known defects, etc)
- ▶ Restriction of usage

• Terminology

* Program units intended to be reused: Refers to "components" of existing software to be reused for new software development.

4.1.2 Preparing for Unit Test

Input

- (SW305) Software Detailed Design Description
- (SU1002) Software Development Environment
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Create the Unit Test Specifications, and be ready to conduct the Unit Test.

Output

- (SW401) Unit Test Specifications 
- (SW402) Unit Test Data
- (SW403) Internal Confirmation Note (on Unit Test Specifications)

 Document template sample is available.

Action

Get ready to conduct the Unit Test by preparing the following items (1)-(4).

(1) Prepare the test cases for the Unit Test.

- ▶ Prepare the test cases to check whether the detailed software design can be implemented correctly as defined in the Software Detailed Design Description (SW305), and document the prepared test cases orderly in the form of Unit Test Specifications. The test cases that should be prepared include:
 - Functional test;
 - Condition coverage test / data boundary-value test.

(2) Prepare the test data.

- ▶ Create the data (specific input data, etc) necessary to run the test cases prepared above in (1).

(3) Create the stubs / test drivers (pseudo-software).

- ▶ If necessary, create the stubs / test drivers to operate the functional units that need to be tested.

(4) Also have the various test criteria defined beforehand, including the criteria for judging the test results, the criteria for evaluating the Unit Test on the overall, and the criteria for determining the satisfactory completion of the Unit Testing activity.

(5) Prepare the test cases for verifying the modification (when modifications are implemented and need to be verified).

- ▶ When a defect is detected during Unit Testing and modification has been implemented to resolve it, prepare test cases to verify that the modification was successful in eliminating the defect, and that no other problems have derived from this modification.
 - Determine the scope of the modification verification test and select the appropriate test cases, based on the description of the defect.

Precaution

● Review the Unit Test Specifications.

Matters that should be reviewed include the:

- ▶ Adequacy of the number of test cases;

- ▶ Coverage of the test cases (instructions, conditions, judgment);

- ▶ Contradictions / duplications of test cases;

- ▶ Method of testing the hardware driver and interrupt handler;
- ▶ Method of testing the program unit that requires high-speed processing;
- ▶ Error processing.
- Points to consider regarding the input values used in the test cases:
 - ▶ Normal data;
 - ▶ Data without any set value (NULL value);
 - ▶ Values / conditions far greater or smaller than the boundary-values / specified values;
 - ▶ Minimum value, maximum value.
- Points to consider regarding the preparation of the modification verification test:
 - ▶ Is the test covering all the areas that are affected by the modification?
- ▶ Basically, reuse the test case that has already been developed. However, if a wide area has been modified or affected by the modification, consider preparing a new test case. (Bear in mind the area used for global variables / shared memory, etc).
- Points to keep in mind in documentation:
 - ▶ Attach the revision history and indicate clearly where have been revised;
 - ▶ Clearly indicate who or which organization is responsible of the created document;
 - ▶ Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management; SUP7 Change Management



SWP4.2 Conducting the Implementation and Unit Test

Implement the program units and conduct the Unit Test as defined in the Software Detailed Design Description.

4.2.1 Implementing the Program Units

Input

- (SW305) Software Detailed Design Description
- (SU1002) Software Development Environment
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Implement the program units.

Output

- (SW404) Program Unit

See also
Coding practices and conventions

Action

(1) Implement the program units.

- ▶ Implement the program units according to the relevant contents defined in the Software Detailed Design Description.
- ▶ Compile, and if there are syntax errors, modify the errors.

(2) Check the program units that are going to be used for the current software development.

- ▶ If existing software assets or commercial software products are going to be used, check

their source codes and decide on which part of the source code to use for the ongoing development.

(3) Modify the defect.

- ▶ When the program unit is found to be defective in source code reviews and tests, modify the defect by referencing the information on this defect described in the Defect Management Ticket.

Precaution

● Matters to be careful in implementation include the:

- ▶ Use of techniques to achieve high-speed processing (such as, inline assembler);
- ▶ Use of preferable optimization techniques that vary depending on the extent of memory constraints;
- ▶ Consideration of making the source code easy to read and test.
- ▶ Obligation to follow the coding conventions,

etc;

- ▶ Consideration to enter comments in the source code to make the processing contents easy to understand;
- ▶ When modifications are implemented to resolve the defects, attach the revision history and clearly indicate where has been revised.

- After completing the implementation, bear in mind the need to perform source code version management (check-in / check-out).

4.2.2 Conducting the Unit Test

Input

- (SW401) Unit Test Specifications
- (SW402) Unit Test Data
- (SW404) Program Unit
- (SU1002) Software Development Environment
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Conduct the Unit Test by following the Unit Test Specifications.

Output

- (SW405) Unit Test Results (notes)

Action

(1) Conduct the Unit Test.

- ▶ Conduct the Unit Test, based on Unit Test Specifications (SW401), and gain the test results as the output.
- ▶ When an alternative test is carried out, keep records of the test results, along with the reason(s) stating explicitly why the alternative test had to be carried out.
- ▶ When a defect is detected while testing, decide whether to continue conducting the remaining tests, or suspend them until the defect is resolved.
- ▶ Collect the outputted test results (various logs, etc).

- ▶ When a prepared test case cannot be conducted, keep records of the event, along with the reason(s) stating explicitly why it was not executable. Moreover, determine the reasonableness of the stated reason(s).

(2) When modification has been implemented to resolve the defect detected while testing (such as, during Unit Test), conduct the test for verifying the modification to:

- ▶ Check whether the defect has been eliminated or not;
- ▶ Check whether the modification to resolve the defect has led to any other defects or not.

Precaution

- Before running the test for verifying the implemented modification, be sure to check that

the latest modified version is tested.

4.2.3 Reviewing the Unit Test Results

Input

- (SW401) Unit Test Specifications
- (SW405) Unit Test Results (notes)
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Review the results gained from the Unit Test, and judge whether the tested unit has passed or failed this test.

Output

- (SW406) Unit Test Report 
- (SU601) Defect Management Ticket 

 Document template sample is available.

Action

(1) Review the results of the Unit Test

- ▶ Review the test results to judge whether the tested software has passed or failed each test case that has been conducted.
- ▶ When the test for verifying the modification has been carried out and the result shows that the defect has been resolved through the

modification, record the findings in the Defect Management Ticket as the evidence that the modification has been verified.

(2) Record the defects.

- ▶ When a defect is detected, record the contents of the defect in the Defect Management Ticket.

Precaution

- Do not take any test results for granted, and consider the possibility of incorrect information included in the Unit Test Specifications.

(Unit Test Report may be substituted by the Unit Test Specifications when the outcome of the Unit Test (pass/fail judgment) is entered in the latter document).



SWP4.3 Reviewing the Implementation and Unit Test Results

Review the implementation and the test results gained from the Unit Test to check that the implemented program units can process correctly as defined in the Software Detailed Design Description.

4.3.1 Reviewing the Source Code

Input

- (SW305) Software Detailed Design Description
- (SW404) Program Unit

Outline

Check whether or not the implementation (coding) has been completed correctly to achieve the functionalities described in the Software Detailed Design Description.

Output

- (SW407) Internal Confirmation Note (on Source Code)

Action

Check whether the source code for realizing the program units has been implemented correctly or not.

- ▶ Check whether the functionalities described in the Software Detailed Design Description can be achieved or not.
- ▶ Compare the source code with the coding practices and coding conventions adopted in the organization engaged in the current software development to check whether the source code has any problems or not.
 - For functionalities that must be processed at high speed, is the implemented code capable of meeting this requirement?
 - Are any techniques applied to optimize the source code in accordance with the memory constraints?
- Is the source code easy to read and test?
- Is the source code following the coding conventions?
- Are there any comments entered in the source code to make the processing contents easy to understand?
- In case the source code had to be modified to resolve the defect that was found, is the revision history available to check what and where has been revised?

4.3.2 Reviewing the Unit Test Results Internally

Input

- (SW305) Software Detailed Design Description
- (SW401) Unit Test Specifications
- (SW403) Internal Confirmation Note (on Unit Test Specifications)
- (SW406) Unit Test Report
- (SW407) Internal Confirmation Note (on Source Code)
- (SU601) Defect Management Ticket

Outline

Check whether or not there are pending issues that are still not solved and/or any test cases in the Unit Test that have not been tested yet, and document the findings orderly in the form of the Internal Review Report.

Output

- (SW408) Internal Review Report (on Implementation & Unit Test) 

 Document template sample is available.

Action

(1) Review the results of the Unit Test from the following perspectives:

- ▶ Check whether there are any pending issues or not, and if there are any, gain an understanding on why they are still unsolved, and decide whether they should be solved immediately or carried over as they are to the next testing phase.
- ▶ Check whether there have been any test cases that were not carried out, and if there were any,

investigate the reason(s) why they were not carried out, and examine the possible solutions.

- (2) Document the findings of the above check points orderly in the form of an Internal Review Report (SW408) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

● Regarding the results gained from the Unit Test, also check the following points:

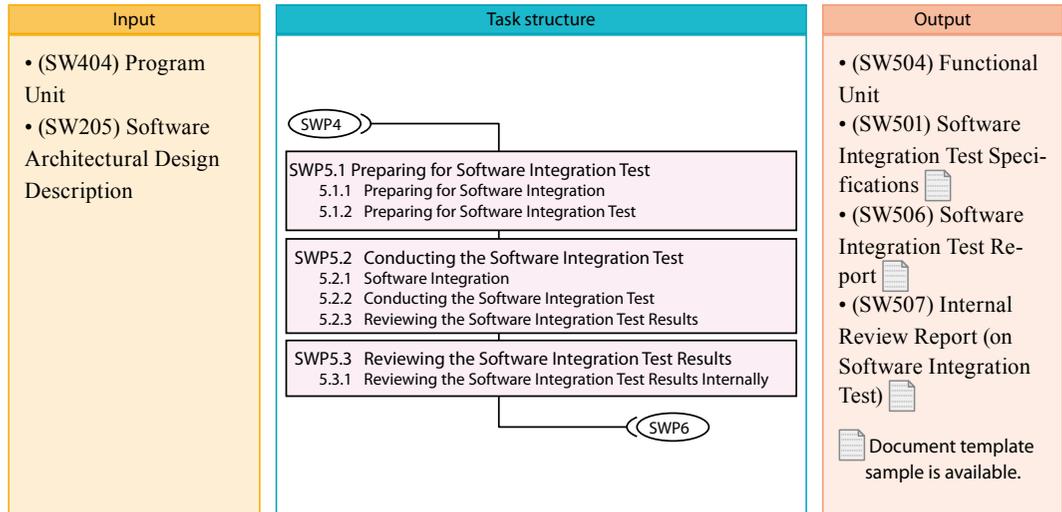
- ▶ Check whether the number of defects that have been detected is acceptable or not, based on the quality criteria;
- ▶ When the test for verifying the modification has been conducted, check whether the scope of the test and the test environment have been

appropriate or not.

- The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).

SWP5 Software Integration Testing

Assemble the individual program units that structure the software one by one, and test the functionalities expected to be provided by the program units respectively when they are executed in combinations.



Description

In this activity:

- (5.1.1) Prepare for software integration by, such as, creating make file used for integrating the individual program units;
- (5.1.2) Proceed with the preparation of Software Integration Test;
- (5.2.1) (5.2.2) (5.2.3) Integrate the individual program units, conduct the integration test, and review the test results;
- (5.3.1) Review the results of the Software Integration Test, based on the pre-defined check points, and document the outcome of this review orderly in the form of an Internal Review Report.

In the context of embedded software, Software Integration Test can be positioned as the opportunity to confirm that the software integration has been achieved in accordance with the software requirements and software architectural design.

Consideration

- ▶ In case of embedded software integration, program units are integrated one by one by using stubs and test drivers.
- ▶ When a defect is detected as a result of the integration test, bear in mind the need for version management of the source code of program units that are integrated.

■ <Reference> Techniques and Tools

- ▶ Boundary-value / limit-value analysis method, etc
- ▶ Code clone detection tool
- ▶ Cleanroom method

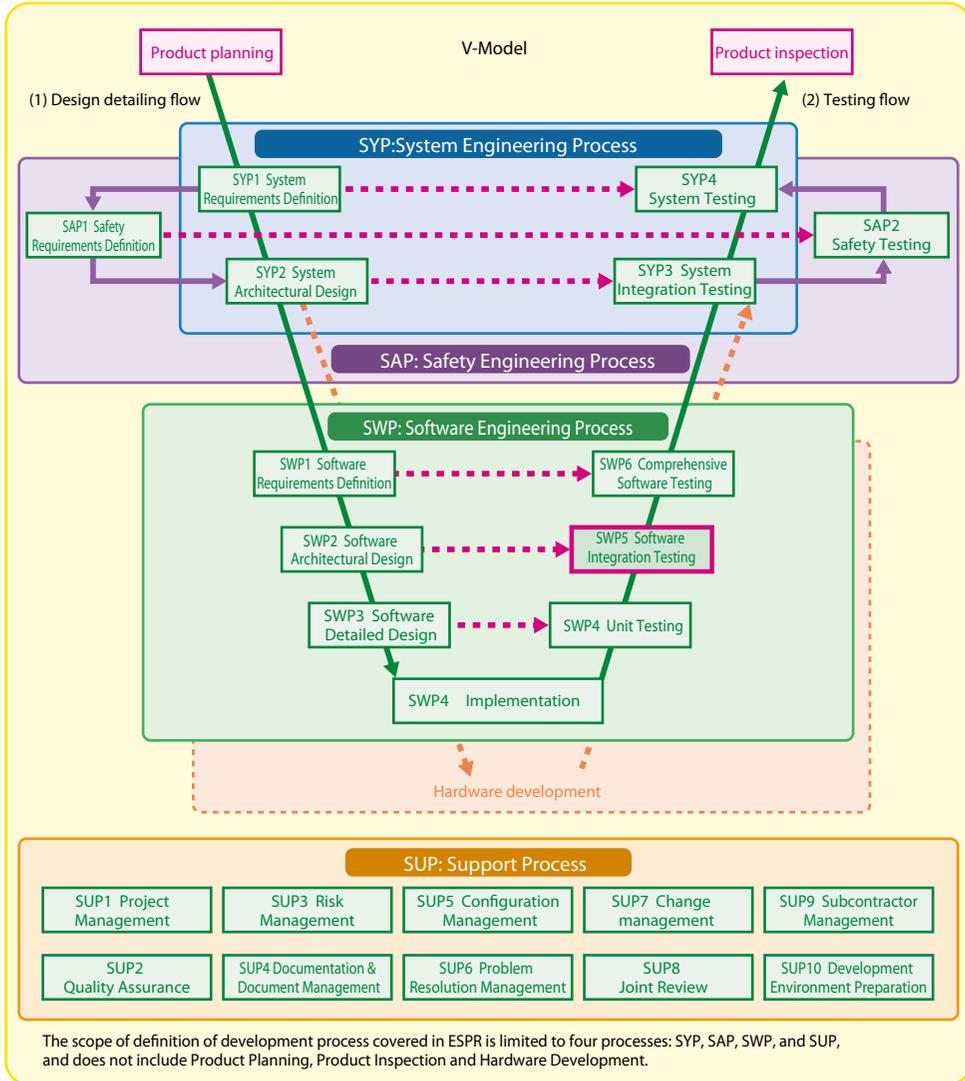


Figure 2.12 V-Model and Development Process (SWP5 - Software Integration Testing)



SWP5.1 Preparing for Software Integration Test

Prepare for software integration as well as for conducting the Software Integration Test.

5.1.1 Preparing for Software Integration

Input

- (SW404) Program Unit
- (SW205) Software Architectural Design Description

Outline

Prepare the program units that are going to be integrated, and the environment to perform the integration.

Output

- (SU1002) Software Development Environment (make file, etc)

Action

Prepare for the software Integration.

- ▶ Prepare the program units that are going to be integrated.
- ▶ Prepare the environment for integrating the software, including the compile and link environments (creating the make file, etc).

Precaution

- Check the program units to be integrated for the following points:
 - ▶ Version
 - ▶ Quality (already known defects, etc)
- ▶ Whether the conditions set for compile and link (allocating registers for local variables, whether memory efficiency or execution speed is prioritized, etc) are appropriate or not.

Bug Curve and Reliability

The reliability of software is affected by the bugs that may be hidden in the software. However, it is extremely difficult to detect all the bugs.

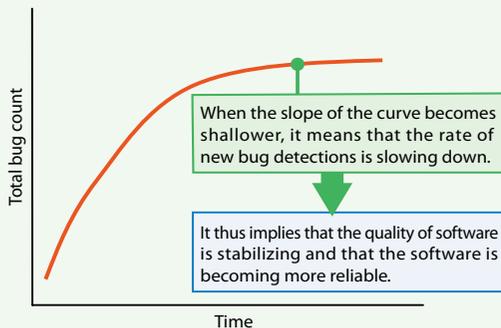
In general, many bugs in the software are found soon after commencing the tests. The number of bug detections tends to decrease gradually along with the progress of the tests and eventually becomes almost nil by the end of the testing phase. Due to this tendency, the quality of software is generally considered to be stabilizing when the number of newly detected bugs added to the cumulative total number of bug detections (referred to simply as the “total bug count” hereafter) within a certain length of time starts decreasing steadily. In other words, when the number of new bugs detected within the given period does not decrease steadily, this statistical data suggests that the software quality is not yet stabilizing.

The changes in the bug detection rate can be confirmed easily through a visual check, using a graph showing the “bug curve” (see below). The bug curve, also known as the “reliability growth curve”, represents the number of bugs detected as time passes. When the slope of this curve is shallow, it means that the number of new bugs detected in the course of testing is increasing only gradually, implying that the software is becoming more reliable. There are various types of reliability growth curves, including the better known ones called the “Gompertz curve” and “logistic curve”.

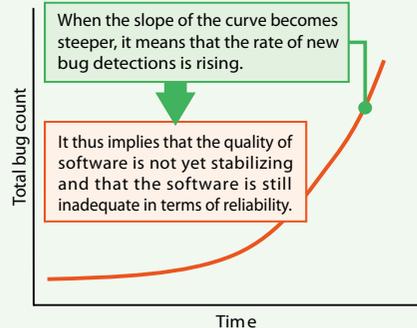
A special attention should be given to the bugs detected near the end of the testing phase to correctly determine their significance to software quality. For instance, the total bug count may temporarily increase sharply when numerous typographical errors are found in one of the final tests conducted to check the quality of the Help contents that can be accessed via a user interface. However, since typos are minor issues, they should not be weighed heavily even when quite many of them are detected, and the software quality can still be considered to be stabilizing, if no serious bugs are detected. On the other hand, the quality of the software under development should definitely be judged as not yet stabilizing, if the detected bugs are found to be critical that they require the specifications to be revised.

The bug curve can be used to measure the reliability of the software. At the same time, it is also important to weigh the severity of the detected bugs in order to determine the software quality correctly.

- (1) When the total bug count stops increasing rapidly in the latter period of the testing phase



- (2) When the total bug count continues to increase rapidly in the latter period of the testing phase



5.1.2 Preparing for Software Integration Test

Input

- (SW205) Software Architectural Design Description
- (SU1002) Software Development Environment
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Get ready to conduct the Software Integration Test.

Output

- (SW501) Software Integration Test Specifications 
 - (SW502) Software Integration Test Data
 - (SW503) Internal Confirmation Note (on Software Integration Test Specifications)
-  Document template sample is available.

Action

Get ready to conduct the Software Integration Test by preparing the following items (1) (5).

(1) Prepare the test cases for the Software Integration Test.

- ▶ Prepare the test cases to check whether the functionalities defined in the Software Architectural Design are correctly realized or not, and organize them in the form of Software Integration Test Specifications. The test cases that should be prepared include:
 - Functional test;
 - Condition coverage test;
 - Data boundary-value test;
 - Interface test for testing the interface between the program units and the interface between the functional units.

(2) Prepare the test data.

- ▶ Create the data (specific input data, etc) necessary to run the test cases prepared above in (1).

(3) Create the stubs / test drivers* (pseudo-software).

- ▶ If the test plan is to first integrate some program units that partially comprise the software and test this integrated portion,

prepare the stubs / test drivers and perform the compile and link to create a testable state.

(4) Also have the various test criteria defined beforehand, including the criteria for judging the test results, the criteria for evaluating the Software Integration Test on the overall, and the criteria for determining the satisfactory completion of the Software Integration Testing activity.

(5) Prepare the test cases for verifying the modification (when modifications are implemented and need to be verified).

- ▶ When a defect is detected during the Software Integration Testing activity and modification has been implemented to resolve it, prepare test cases to verify that the modification was successful in eliminating the defect, and that no other problems have derived from this modification.
- ▶ Determine the scope of the modification verification test and select the appropriate test cases, based on the description of the defect.

• Terminology

- * Stub/test driver : Stub is a placeholder for substituting the lower-level module that is called by the tested component (program unit). Test driver is a placeholder for substituting the higher-level module that transmits the test data to the tested component (program unit).

■ Precaution

- Review the contents of the Software Integration Test Specifications. Matters that should be reviewed at this stage include:
 - ▶ Adequacy of the number of test cases based on the scale and size of the software to be developed;
 - ▶ Combination of the functionalities
 - ▶ Coverage of the test cases (corresponding to the contents of the Software Architectural Design Description);
 - ▶ Contradictions / duplications of test cases;
 - ▶ Method of testing the hardware driver and interrupt handler (nested interrupts, etc);
 - ▶ Method of testing the program unit that requires high-speed processing;
 - ▶ Validity of the criteria for evaluating non-functional software requirements (performance, etc);
 - ▶ Error processing.
- Regarding the input values used in the test cases, consider the following, and also take account of singularity from the standpoint of system operation:
 - ▶ Normal data;
 - ▶ Data without any set value (NULL value);
 - ▶ Values / conditions far greater or smaller than the boundary-values / specified values;
 - ▶ Minimum value, maximum value.
- Points to consider regarding the preparation of the modification verification test:
 - ▶ Is the test covering all the areas that are affected by the modification?
 - ▶ Basically, reuse the test case that has already been developed. However, if a wide area has been modified or affected by the modification, consider preparing a new test case. (Bear in mind the area used for global variables / shared memory, etc).
- Points to keep in mind in documentation:
 - ▶ Attach the revision history and indicate clearly where have been revised;
 - ▶ Clearly indicate who or which organization is responsible of the created document;
 - ▶ Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management;
SUP7 Change Management



SWP5.2 Conducting the Software Integration Test

Integrate the software (program units) and conduct the Software Integration Test.

5.2.1 Software Integration

Input

- (SW404) Program Unit
- (SW404) Program Unit
- (SW205) Software Architectural Design Description
- (SU1002) Software Development Environment

Outline

Integrate the program units.

Output

- (SW504) Functional Unit

Action

Perform the compile and link, and integrate the program units.

Precaution

- Points to consider regarding the software integration:
 - ▶ Confirm that the program unit of appropriate version is used for integration.

5.2.2 Conducting the Software Integration Test

Input

- (SU1002) Software Development Environment
- (SW504) Functional Unit
- (SW501) Software Integration Test Specifications
- (SW502) Software Integration Test Data
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Conduct the Software Integration Test, according to the Software Integration Test Specifications.

Output

- (SW505) Software Integration Test Results

Action

(1) Conduct the Software Integration Test.

- ▶ Conduct the Software Integration Test, based on the Software Integration Test Specifications (SW501), and gain the test results as the output.
- ▶ When an alternative test is carried out, keep records of the test results, along with the reason(s) stating explicitly why the alternative test had to be carried out.
- ▶ When a defect is detected while testing, decide whether to continue conducting the remaining tests, or suspend them until the defect is resolved.
- ▶ Collect the outputted test results (various logs,

etc).

- ▶ When a prepared test case cannot be conducted, keep records of the event, along with the reason(s) stating explicitly why it was not executable. Moreover, determine the reasonableness of the stated reason(s).

(2) Conduct the test for verifying the modification to:

- ▶ Check whether the defect has been eliminated or not by the modification, and whether the modification to resolve the defect has led to any other defects or not.

Precaution

- Before running the test for verifying the implemented modification, be sure to check that the latest modified version is tested.

5.2.3 Reviewing the Software Integration Test Results

Input

- (SW501) Software Integration Test Specifications
- (SW505) Software Integration Test Results
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Review the results gained from the Software Integration Test, and judge whether the tested software has passed or failed this test.

Output

- (SW506) Software Integration Test Report 
- (SU601) Defect Management Ticket 

 Document template sample is available.

Action

(1) Review the results gained from the Software Integration Test.

- ▶ Review the test results to judge whether the tested software has passed or failed each test case that has been conducted.
- ▶ When the test for verifying the modification has been carried out and the result shows that the defect has been resolved through the

modification, record the findings in the Defect Management Ticket as the evidence that the modification has been verified.

(2) Record the defects.

- ▶ When a defect is detected, record the contents of the defect in the Defect Management Ticket.

Related processes: SUP6 Problem Resolution Management

Precaution

- When reviewing the test results, bear in mind the possibility that the contents of the Software Integration Test Specifications may be incorrect.



SWP5.3 Reviewing the Software Integration Test Results

Review the results of the Software Integration Test from the standpoint of checking whether or not the integrated software is capable of processing correctly what it is required to achieve as defined in the Software Architectural Design.

5.3.1 Reviewing the Software Integration Test Results Internally

Input

- (SW205) Software Architectural Design Description
- (SW501) Software Integration Test Specifications
- (SW503) Internal Confirmation Note (on Software Integration Test Specifications)
- (SW506) Software Integration Test Report
- (SU601) Defect Management Ticket

Outline

Check whether or not there are pending issues that are still not solved and/or any test cases in the Software Integration Test that have not been carried out, and document the findings orderly in the form of the Internal Review Report.

Output

- (SW507) Internal Review Report (on Software Integration Test) 

 Document template sample is available.

Action

(1) Review the results of the Software Integration Test from the following perspectives:

- ▶ Check whether there are any pending issues or not, and if there are any, gain an understanding on why they are still unsolved, and decide whether they should be solved immediately or carried over as they are to the next testing phase.

Related processes: SUP8 Joint Review; SUP1 Project Management

- ▶ Check whether there have been any test cases

that were not carried out, and if there were any, investigate the reason(s) why they were not carried out, and examine the possible solutions.

Related processes: SUP6 Problem Resolution Management

- (2) Document the findings of the above check points orderly in the form of an Internal Review Report (SW507) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

- Check whether the number of defects that have been detected is acceptable or not, based on the quality criteria.
- When the test for verifying the modification has been conducted, check whether the scope of the test and the test environment have been appropriate or not.
- The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).

Related Standards

ISO/IEC 12207 Information technology— Software life cycle processes (JIS X 0160)

Year of establishment / revision: ISO/IEC 1995, AMD.1 2002, AMD.2 2004, JIS 1996

Purpose: Standardization of the concepts of software life cycle processes

Scope: Software development process

URL: <http://www.iso.org/>

Source in Japan: Japanese Standards Association

Overview:

Standard that organizes the software product development processes systematically, and serves as the fundamental standard pertaining to these processes.

This standard has been established for the purpose of setting the common language, such as, the names used to refer to the various types of work related to the development of software that is nowadays not only used in the field of information technology but is also extending its usages broadly in industrial systems.

This standard lays down the framework of software development in which the processes ranging from the initial planning stage to the final retirement stage when the users stop using the software product are sorted out orderly according to the life cycle of the software, and prescribes the matters concerning the management and improvement of these processes.

This standard defines these processes conceptually by breaking down the various types of work involved in software development hierarchically into “processes”, “activities” and “tasks”. Based on this hierarchy, this guidebook focuses on discussing the specific processes for embedded software development in an orderly manner.

Process description

- [Process name]

- List of activities

[Activity 1]

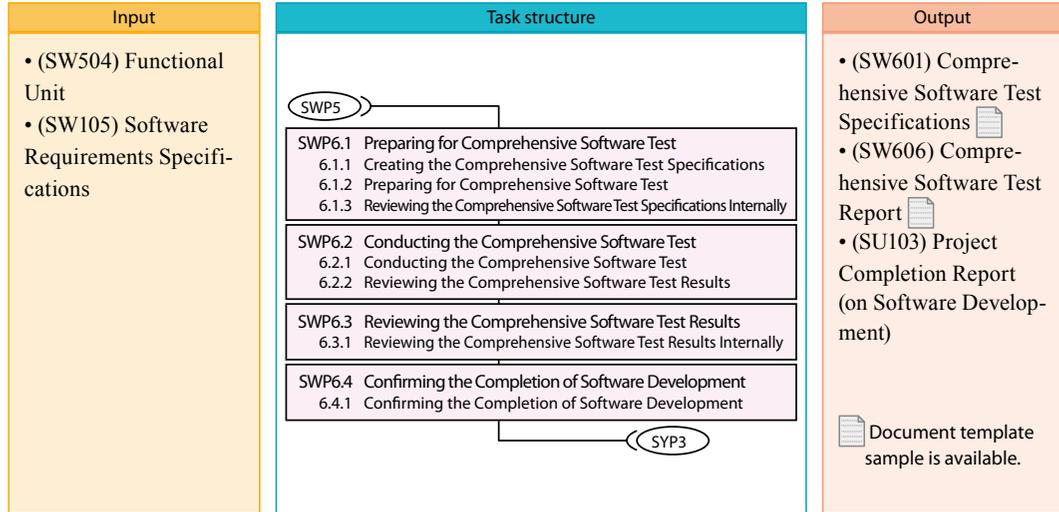
- Task 1

- Task 2

- Task 3

SWP6 Comprehensive Software Testing

Conduct the Comprehensive Software Test, using the software in the state where all the functional units that structure the software are fully integrated.



Description

In this activity:

- (6.1.1) (6.1.2) (6.1.3) Prepare the test specifications for checking and reviewing whether the fully integrated software is capable of providing the functionalities and operations described in the Software Requirements Specifications;
- (6.2.1) (6.2.2) Conduct a comprehensive set of tests collectively referred to as the Comprehensive Software Test by actually operating the software according to the prepared test specifications (hereafter called the “Comprehensive Software Test Specifications”);
- (6.3.1) Review the results of the Comprehensive Software Test, based on the pass/fail criteria applied to this test, and document the findings orderly in the form of an Internal Review Report;
- (6.4.1) Moreover, hold joint meetings with relevant stakeholders to review the results of the Comprehensive Software Test and also carry out the final review of the current software development project. Upon reaching a positive conclusion in these meetings, document the outcome of the joint review orderly in the form of Software Development Project Completion Report.

Comprehensive Software Testing can be regarded basically as the final test phase in software development to check and review the developed software. Therefore, this review must be carried out comprehensively without any omission, by assuming how the software actually operates when the product is used in the real world.

■ Consideration

- ▶ Before conducting the Comprehensive Software Test, a particular attention should be given to the operating environment in which the software is tested (hereafter called the “test environment”). Conduct the Comprehensive Software Test in a test environment assumed to be close to the environment in which the software is likely to operate in the real world.
- ▶ Consider the Comprehensive Software Test as the final opportunity to check and review the software that has been developed, and be careful not to leave out any test cases, including the test cases for conducting the regression test that would be necessary when a defect is detected while testing.
- ▶ Defects that have been detected while testing must be managed according to pre-defined procedure and rules, including the appropriate management of information regarding where they were detected and how they have been resolved, among others.

■ <Reference> Techniques and Tools

- ▶ Simulators for simulating system behaviors, etc
- ▶ Measuring instruments (logic analyzer, oscilloscope, etc)
- ▶ Automatic testing tool (automated regression test, etc)
- ▶ Bug management tool
- ▶ Reliability growth curve

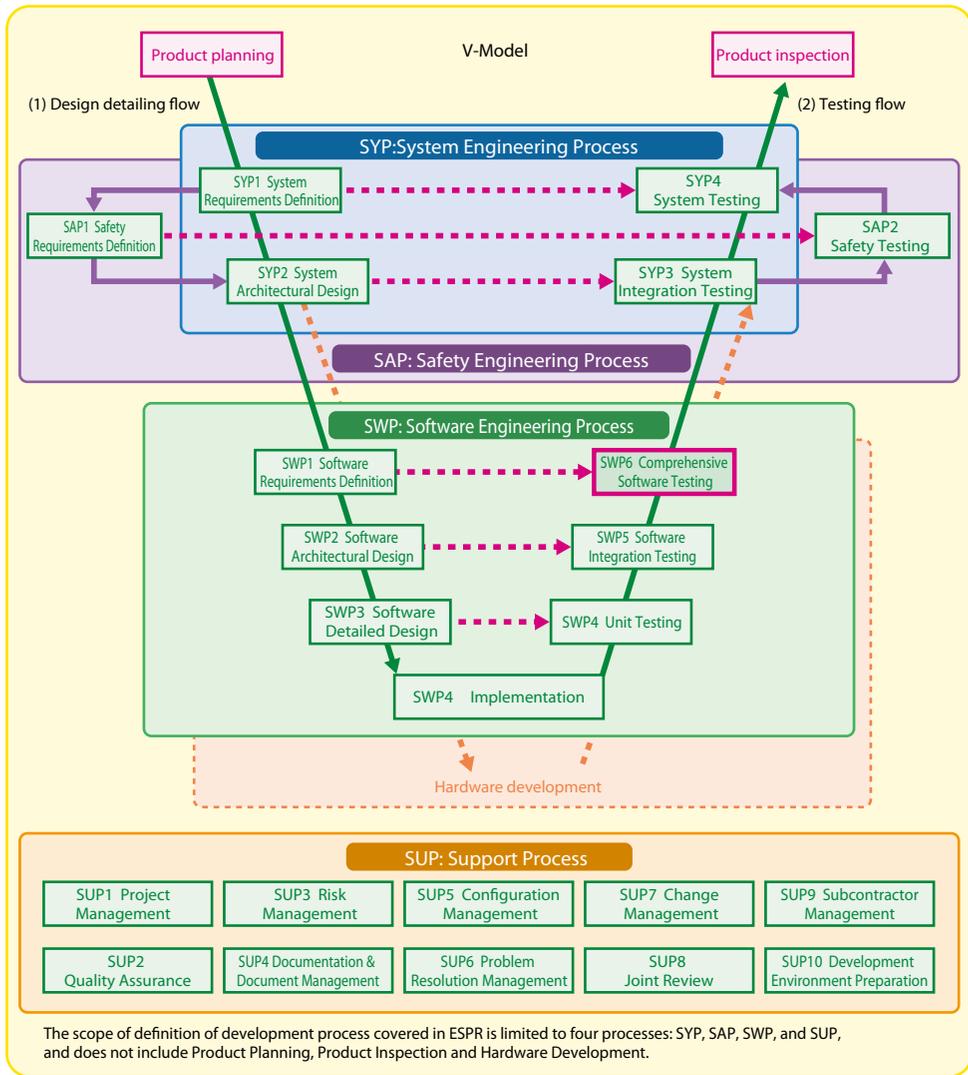


Figure 2.13 V-Model and Development Process (SWP6 Comprehensive Software Testing)



SWP6.1 Preparing for Comprehensive Software Test

Prepare for the Comprehensive Software Test.

6.1.1 Creating the Comprehensive Software Test Specifications

Input

- (SW105) Software Requirements Specifications
- Information necessary to create the test specifications (deliverables)

Outline

Create the Comprehensive Software Test Specifications.

Output

- (SW601) Comprehensive Software Test Specifications 
-  Document template sample is available.

See also [Comprehensive Software Test Specifications created in the past](#)

Action

Create the Comprehensive Software Test Specifications based on the contents of the Software Requirements Specifications.

- ▶ Prepare the test cases to check whether the software can correctly realize all the required functionalities described in the Software Requirements Specifications. When preparing these test cases, be sure to cover all the functional and non-functional software requirements.
- ▶ Have the criteria for pass/fail judgment of each test case clearly defined beforehand.

Precaution

- ▶ Examine whether the Comprehensive Software Test Specifications created in the past are reusable or not.
 - ▶ Check whether the past incidents on defects are taken into consideration.
 - ▶ If the current software is built for a product planned to be developed in series, check whether the test cases on new functionalities are sufficient, and whether the impact of changes in the functionalities is clearly grasped or not.
 - ▶ Extract all the necessary test cases without fail, by utilizing, such as, state transition table and matrix coverage methodology.
 - ▶ Check whether the test procedures and criteria for judging the completion of the Comprehensive Software Test are complying with relevant laws, regulations and standards, among others.
 - ▶ Check whether the expected outcome of the outputs is clearly described or not.
 - ▶ Create test cases that take account of the situations in which the end users actually use the product (software).
 - ▶ Create test cases for testing normal operations as well as for testing abnormal / exceptional cases.
 - ▶ Take account of the consistency with other systems as well as with hardware.
 - It is desirable to examine and create the Comprehensive Software Test Specifications during the upstream process (such as, during the design phase).
 - Points to keep in mind in documentation:
 - ▶ Attach the revision history and indicate clearly where have been revised;
 - ▶ Clearly indicate who or which organization is responsible of the created document;
 - ▶ Ensure that the created document is managed properly by performing configuration management and change management.
- Related processes: SUP5 Configuration Management; SUP7 Change Management

6.1.2 Preparing for Comprehensive Software Test

Input

- (SW601) Comprehensive Software Test Specifications
- (SW504) Final executable source code
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Prepare the test data and test environment that would be necessary to conduct the Comprehensive Software Test.

Output

- (SW602) Real Machine
- (SW603) Comprehensive Software Test Data

Action

Prepare the test data and test environment that would be necessary to conduct the Comprehensive Software Test.

(1) Create the test data

- ▶ Create the data (specific input data, etc) necessary to run the test cases prepared in 6.1.1.

(2) Prepare the test environment.

- ▶ Prepare the test environment that would be necessary to conduct the Comprehensive Software Test.
 - Create the testing jigs (communication target, etc).
 - Create the simulation software.
- ▶ Convert the source code into executable format and write it to the semiconductor memory.
- ▶ Mount the memory and circuits, and be ready with real machines (or prototypes) to run the software in the real physical environment.

(3) Also have the various test criteria defined beforehand, including the criteria for judging the test results, the criteria for evaluating the Comprehensive Software Test on the overall, and the criteria for determining the satisfactory completion of the Comprehensive Software Testing activity.

(4) Prepare the test cases for verifying the modification (when modifications are implemented and need to be verified).

- ▶ Prepare the test cases to verify that the modification was successful in eliminating the defect, and that no other problems have derived from this modification.
 - Decide on the scope of the test and choose which test case to conduct, depending on the description of the defect.

Precaution

- ▶ Keep in mind to prepare a wide variation of test data that can be used to test the overall workings of the software, and also to test the software especially in events that are related to abnormal processing in case of embedded software
- ▶ When preparing the test jigs, confirm that the precision level is within the acceptable tolerance limits.
- ▶ Check whether the environment is suitable for the hardware to operate normally (noise, power supply, etc).
- ▶ Confirm that the software to be tested is the

correct version.

- ▶ As for the test environment, contrive ways to facilitate test reproducibility to help analyze what the cause(s) may be when a defect is detected.
- ▶ Points to consider when preparing the test for verifying the modification
 - Is the scope of this test covering all the potentially affected areas?
 - Basically, reuse the test cases that have already been created. Consider preparing new test cases when the modification was large-scaled, and extensive areas were affected.

6.1.3 Reviewing the Comprehensive Software Test Specifications

Input

- (SW105) Software Requirements Specifications
- (SW601) Comprehensive Software Test Specifications

Outline

Review the Comprehensive Software Test Specifications.

Output

- (SW604) Internal Confirmation Note (on Comprehensive Software Test Specifications)

Action

Review the Comprehensive Software Test Specifications that have been created.

- ▶ Review the Comprehensive Software Test Specifications (SW601) within the development group to check whether or not the functional and non-functional requirements defined in the Software Requirements Specifications (SW105) can be verified sufficiently by the test cases described in the Comprehensive Software Test Specifications.

Precaution

- Matters that should be reviewed include the:
 - ▶ Adequacy of the number of test cases based on the size of the software;
 - ▶ Coverage of the test cases (corresponding to the contents of the Software Requirements Specifications);
 - ▶ Contradictions / duplications of test cases;
 - ▶ Validity of the criteria for evaluating non-functional software requirements (performance, etc);
 - ▶ Error processing;
 - ▶ Fail-safe processing;
 - ▶ Expected values
- When the test cases are created by the developers, closely check whether the test cases that they picked out and the pass/fail criteria for evaluating the test result are strict enough (adequate) or not.



SWP6.2 Conducting the Comprehensive Software Test

Conduct the Comprehensive Software Test.

6.2.1 Conducting the Comprehensive Software Test

Input

- (SW601) Comprehensive Software Test Specifications
- (SW603) Comprehensive Software Test Data
- (SW602) Real Machine
- (SU1002) Software Development Environment

Outline

Conduct the Comprehensive Software Test, based on the Comprehensive Software Test Specifications.

Output

- (SW605) Comprehensive Software Test Results

Action

Conduct the Comprehensive Software Test by following the Comprehensive Software Test Specifications.

(1) Conduct the Comprehensive Software Test.

- ▶ Conduct the Comprehensive Software Test, based on Comprehensive Software Test Specifications (SW601), and gain the test results as the output.
- ▶ When an alternative test is carried out, keep records of the test results, along with the reason(s) stating explicitly why the alternative test had to be carried out.
- ▶ When a defect is detected while testing, decide whether to continue conducting the remaining tests, or suspend them until the defect is resolved.
- ▶ Collect the outputted test results (various logs,

etc).

- ▶ When a prepared test case cannot be conducted, keep records of the event, along with the reason(s) stating explicitly why it was not executable. Moreover, determine the reasonableness of the stated reason(s).

(2) When a defect occurred and modification has been implemented to resolve it, conduct the test for verifying the modification to:

- ▶ Check whether the defect has been eliminated or not.
- ▶ Check whether the modification to resolve the defect has led to any other defects or not.

Precaution

- Before running the test for verifying the implemented modification, be sure to check that the latest modified version is tested.

6.2.2 Reviewing the Comprehensive Software Test Results

Input

- (SW601) Comprehensive Software Test Specifications
- (SW504) Final executable source code
- (SW605) Comprehensive Software Test Results
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Review the results gained from the Comprehensive Software Test, and judge whether the tested software has passed or failed this test.

Output

- (SW606) Comprehensive Software Test Report 
- (SU601) Defect Management Ticket 

 Document template sample is available.

Action

Review the results gained from the Comprehensive Software Test, and judge whether the software has passed or failed each test case that has been conducted.

- ▶ When a defect is detected, record the description of the defect in the Defect Management Ticket.
- ▶ When the test for verifying the modification has been carried out and the result shows that the defect has been resolved through the

modification, record the findings in the Defect Management Ticket as the evidence that the modification has been verified.

Related processes: SUP6 Problem Resolution Management

Precaution

- When reviewing the test results, bear in mind the possibility that the contents of the Comprehensive Software Test Specifications may be incorrect.
- Judge from the test results whether the software

has passed or failed the Comprehensive Software Test, based on the criteria defined in the Comprehensive Software Test Specifications.



SWP6.3 Reviewing the Comprehensive Software Test Results

Review the results of the Comprehensive Software Test from the standpoint of checking whether or not the software has correctly realized its requirements defined in the Software Requirements Definition.

6.3.1 Reviewing the Comprehensive Software Test Results Internally

Input

- (SW105) Software Requirements Specifications
- (SW601) Comprehensive Software Test Specifications
- (SW604) Internal Confirmation Note (Comprehensive Software Test Specifications)
- (SW606) Comprehensive Software Test Report
- (SU601) Defect Management Ticket

Outline

Check whether or not there are pending issues that are still not solved and/or any test cases in the Comprehensive Software Test that have not been carried out, and document the findings orderly in the form of the Internal Review Report.

Output

- (SW607) Internal Review Report (on Comprehensive Software Test) 

 Document template sample is available.

Action

(1) Review the results of the Comprehensive Software Test Results from the following perspectives:

- ▶ Check whether there are any pending issues or not, and if there are any, gain an understanding on why they are still unsolved, and decide whether they should be solved immediately or carried over as they are to the next testing phase.

Related processes: SUP8 Joint Review; SUP1 Project Management

- ▶ Check whether there have been any test cases

that were not carried out, and if there were any, investigate the reason(s) why they were not carried out, and examine the possible solutions.

Related processes: SUP6 Problem Resolution Management

- (2) Document the findings of the above check points orderly in the form of an Internal Review Report (SW607) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

- Check whether the number of defects that have been detected is acceptable or not, based on the quality criteria.

- ▶ When the test for verifying the modification has been conducted, check whether the scope of the test and the test environment have been appropriate or not.

- The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).



SWP6.4 Confirming the Completion of Software Development

Evaluate the developed software among the stakeholders by checking whether the software requirements defined in the System Requirements Definition are correctly realized by the developed software or not, and carry out the final review of software development.

6.4.1 Confirming the Completion of Software Development

Input

- (SW105) Software Requirements Specifications
- (SW601) Comprehensive Software Test Specifications
- (SW606) Comprehensive Software Test Report
- (SW607) Internal Review Report (on Comprehensive Software Test)
- (SU101) Project Plan Description
- (SU601) Defect Management Ticket

Outline

Review the test results described in the Comprehensive Software Report jointly among the stakeholders. Based on the findings from this joint review, make a final judgment as the organization in charge of software development on whether the developed software has passed or failed the Comprehensive Software Test, and sort out the information deemed necessary for later product inspection.

Output

- (SU801) Joint Review Records (on Software Integration Test) 
- (SU103) Project Completion Report (on Software Development) 

 Document template sample is available.

Action

Hold joint meetings to review the results gained from the Comprehensive Software Test according to the following procedure, and create the Completion Report when successful completion of the software development project can be confirmed.

- (1) Gather the stakeholders of software development (personnel in charge of product planning, software developers, hardware developers, system evaluators, personnel in charge of manufacturing, etc) to review the developed software and make a final judgment on whether the software has been successfully developed or not.

In these joint review meetings, focus on examining the following check points in particular:

- ▶ Check whether or not the developed software used by the intended users under the expected environment is:
 - Certainly capable of satisfying the functional requirements as much as expected;
 - Certainly capable of satisfying the non-functional requirements as much as expected.
- ▶ Check whether there are still any pending issues in software development or not.
- ▶ Check whether the tested software has any quality-related issues or not, by using the metrics on, such as, the number of defects detected during the Comprehensive Software Test, the number of modifications implemented, and the number of critical defects that have been identified.

- ▶ Check whether or not the issues indicated in the reviews held in other testing activities prior to Comprehensive Software Testing have all been appropriately addressed and solved.

Related processes: SUP2 Quality Assurance; SUP6 Problem Resolution Management; SUP8 Joint Review

- (2) When the results gained in this review are all positive and satisfactory, create a document called the Completion Report, based on the review records.

For more information on the creation of the Completion Report, see “SUP1.4 Creating the Project Completion Report”.

Distribute the created Completion Report not only to the stakeholders involved in the current software development, but also to those involved in the current hardware development, and those that preside over the entire system.

Related processes: SUP1 Project Management

■ Precaution

- ▶ In this review, confirm that an adequate approach was taken to develop the software both quantitatively and qualitatively, and that the software developed as a result of this approach is valid as a newly created product.
- ▶ Since this review is the final opportunity to review the software within the entire software development process, the judgment on whether the review results have been acceptable or not must be made by the high-ranking personnel responsible of the entire software development.

SAP : Safety Engineering Process

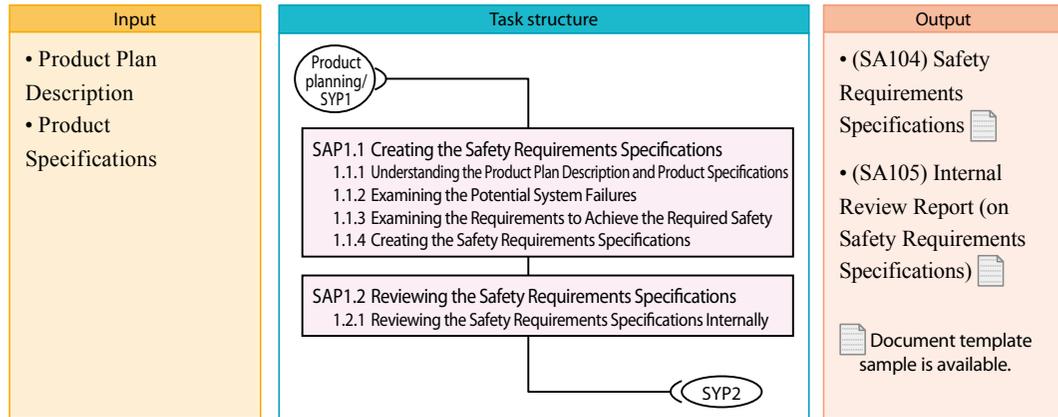
In this process, the safety that the product is required to secure under all conditions in which it is assumed to be used is defined, along with the various types of work to confirm that the product is definitely safe to use in any of the expected usages.

The following activities are included in this process:

ID	Activity	Outline of the activity	Comprising tasks
SAP1	Safety Requirements Definition	Identify the product requirements pertaining to safety, and document them orderly in the form of Safety Requirements Specifications.	SAP1.1 Creating the Safety Requirements Specifications SAP1.2 Reviewing the Safety Requirements Specifications
SAP2	Safety Testing	Carry out a set of tests on the developed product from the standpoint of safety.	SAP2.1 Preparing for Safety Test SAP2.2 Conducting the Safety Test SAP2.3 Reviewing the Safety Test Results

SAP1 Safety Requirements Definition

Clarify the product requirements pertaining to safety.



Description

In this activity:

- (1.1.1) Clarify the situations in which the system is expected to be used and the intended users, by grasping the functionalities to be achieved and services to be provided by the product, based on the contents of the Product Plan Description and Product Specifications;
- (1.1.2) Examine the potential system failures;
- (1.1.3) Examine the requirements on safety that the system is required to meet (i.e.: safety functions), and determine the level of safety that the system is required to secure in each functionality;
- (1.1.4) Create the Safety Requirements Specifications by organizing the information gained from the above tasks in an orderly manner;
- (1.2.1) Review the created Safety Requirements Specifications, based on the pre-defined check points, and document the outcome of this review orderly in the form of an Internal Review Report.

Consideration

- ▶ Keep in mind the following points as the prerequisites for commencing the activity to define the safety requirements:
 - Product planning: Product strategies (such as, the end users' needs) are clearly defined;
 - The situations (contexts) in which the product system is expected to be used are clearly defined;
- ▶ Gain a general knowledge on the level of impact when the safety of the system is compromised and the potential frequency of occurrence of system safety-impairing incidents;
- ▶ Also gain a general knowledge on the extent of impact and the length of time the impact is expected to linger when the safety of the system is compromised;
- ▶ When defining the safety requirements, also examine who and/or where are responsible of achieving the system safety, and the mechanism for ensuring it.

■ <Reference> Techniques and Tools

- ▶ Hazard analysis techniques
 - FTA (Fault Tree Analysis)
 - FMEA (Failure Modes and Effects Analysis)

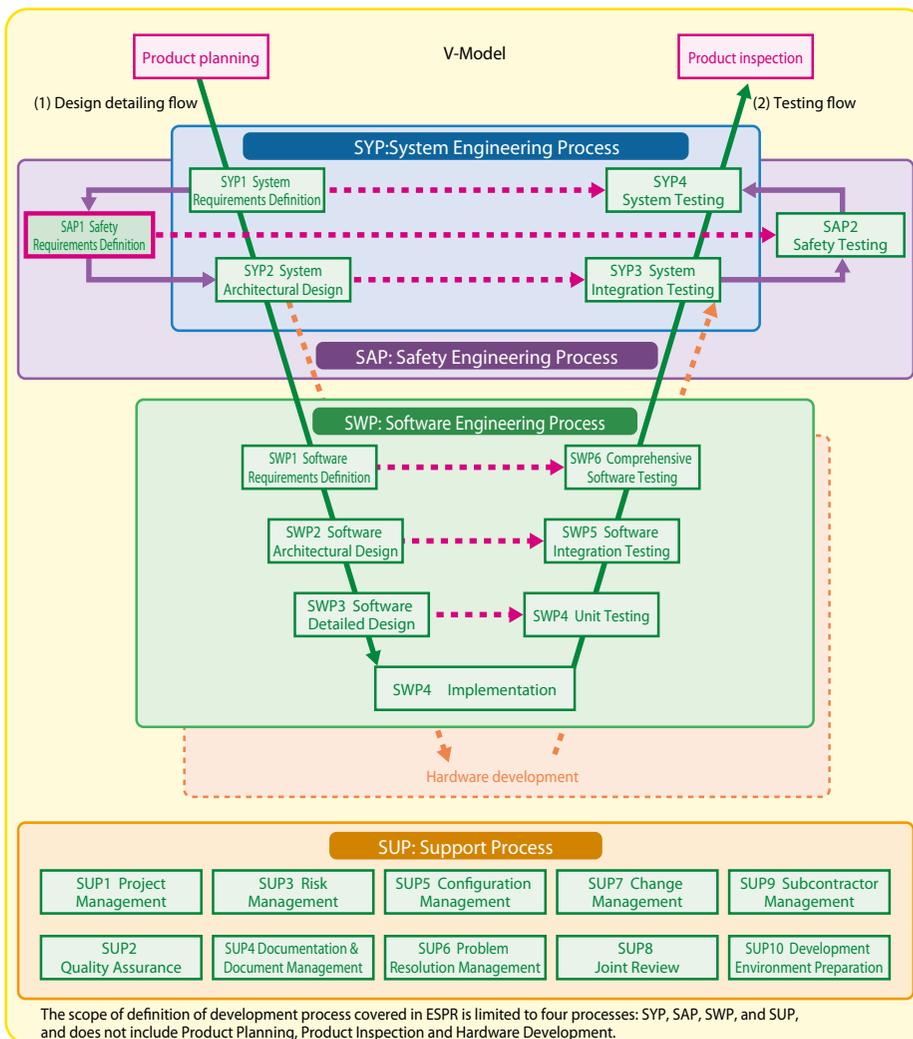


Figure 2.14 V-Model and Development Process (SAP1 Safety Requirements Definition)

SAP1.1 Creating the Safety Requirements Specifications

Clarify the safety items that the system is required to fulfill, based on the contents of the Product Plan Description and Product Specifications, and organize them orderly in the form of Safety Requirements Specifications.

1.1.1 Understanding the Product Plan Description and Product Specifications

Input

- Product Plan Description
- Product Specifications

Outline

From the standpoint of system safety, confirm the contents of the Product Plan Description and Product Specifications created by the organization (department, division, etc) in charge of product planning.

Output

(Product Plan Description, Specifications Confirmation Note)

Action

- (1) Confirm the following points stated in the Product Plan Description:
 - ▶ Product outline and characteristics, functional differences from existing products, etc;
 - ▶ Standards, conventions, laws and regulations related to the product.
- (2) Confirm the following points stated in the Product Specifications;
 - ▶ Product vision and concept;
 - ▶ Intended users and system requirements;
 - ▶ Situations and context in which the product is used;
 - ▶ Constraints in realizing the product.

Precaution

- Points to give particular attention to when confirming the contents of the Product Plan Description include the following:
 - ▶ Since Product Plan Description is a document normally prepared by the product planning or sales department / division to provide the overview of the product to be developed, keep in mind that it does not necessarily state the information pertaining to the safety of the product and system adequately or appropriately;
 - ▶ Since the information stated in the Product Plan Description may be revised due to various factors, including the changes in the intended users and market trends, keep in mind that product safety requirements also have various aspects to be covered, even within the scope of a single product.
- Points to give particular attention to when confirming the contents of the Product Specifications include the following:
 - ▶ Product vision and concept:
 - What kind of functionalities and services are the product supposed to provide?

- If the product cannot provide these functionalities and services safely, how will the system and its surroundings be affected by the lack of product safety?
- ▶ Intended users and system requirements:
 - What kind of users are expected to use the system? Moreover, how are primary, secondary and tertiary users of the system expected to co-relate or interact with each other?
 - Also analyze how the system is likely to affect those who are not direct users of the system when its safety is compromised.
- ▶ Situations and context in which the product is used:
 - Identify all the operating modes of the system used in every conceivable context, including the situations when the intended users use the product in:
 - 1) Normal state;
 - 2) Unexpected state;
and examine the safety aspects of the system in all these situations;
 - Clarify who and/or where are responsible of safety-related matters while the product is being developed and the same after the product is released.
- ▶ Constraints in realizing the product:
 - Identify all the elements that structure the product (software, hardware, etc) and sort out the relationship between each other and their constraints;
 - Also identify the systems and hardware that are prerequisites for using the system to be developed (i.e.: peripheral systems and hardware), and sort out the constraints that they impose on the system's operating modes and functionalities.

1.1.2 Examining the Potential System Failures

Input

- Product Plan Description
- Product Specifications

Outline

Examine the frequency of potential defects and failures occurring in the product or the system, and the level of impact these incidents may have when they occur.

Output

- (SA101) List of Potential System Failures

Action

- (1) List the potential defects and failures that may occur in the product or the system, and examine how frequently any of these incidents may occur.
- (2) Examine how and to what level these defects and failures mentioned above may impact the users and their surroundings when they occur.

Precaution

- ▶ In a normal system, safety functions aimed at preventing defects and failures from occurring and mitigating the impact even if they occur are taken into account. Consider in addition, the cases when these functionalities do not work normally.
- ▶ Examine the impact on safety caused by defects and failures from a broad perspective, including fatal accidents, injury accidents, economic losses and social confusion, among others.
- ▶ Since defects and failures are not only attributable to embedded software that has become defective, but also to peripheral hardware failures that may have served as the trigger, estimate the frequency of failure occurrences by thoroughly examining the system components and related elements.
- ▶ Consider the possibilities of system safety being inhibited, the potential impact, and conceivable countermeasures from a broad perspective, including not only the cases when the safety is compromised due to human errors caused by the users while they are operating or using the product or system, but also the cases caused by various other external factors (disturbances).
- ▶ Especially in case of failures caused by hardware that adversely affect the software and make system operation unsafe, perceive hardware failure occurrences from the standpoint of probabilistic logic.

1.1.3 Examining the Requirements to Achieve the Required Safety

Input

- Product Plan Description
- Product Specifications
- (SA101) List of Potential System Failures

Outline

Examine the requirements (on safety functions) for achieving the safety that the product and the system are required to provide, and determine the level of safety integrity to be achieved by each safety function.

Output

- (SA102) List of System Safety Requirements
- (SA103) System Safety Integrity Level

Action

- (1) Based on Product Plan Description, Product Specifications, and List of Potential System Failures (SA101), give thought to the safety that the product is required to provide, and identify the requirements on safety functions that the system is required to provide to achieve the product safety.
- (2) Consider the safety functions not just for the software but also from the standpoint of maintaining the safety of the entire system that takes account of related hardware behaviors as well. Sort out the safety requirements for system and software architectures by giving particular attention to the following:
 - ▶ Self-monitoring function of the software;
 - ▶ Monitoring function of the hardware, sensors, and actuators.
- (3) Identify and sort out the safety functions that correspond to each operating mode of the system (start-up, automatic, manual, semi-automatic, steady, non-steady, etc).
- (4) Examine the implementation of logic duplication, fail-safe and foolproof mechanisms as the system's safety mechanisms.
- (5) Examine a mechanism to minimize the impact of possible safety-inhibiting factors including disturbances to the system, input data errors and users' operation errors, and maintain the safety of the system.
- (6) In defining the system safety requirements, examine not only the necessary safety functions implemented in the system, but also the methods and approaches to ensure safety of the system while it is used or in operation (such as, how the system administrator and the users must respond when the system is in abnormal state).
- (7) Sort out the techniques and tools to be used to develop a safe product / system. Especially when the required level of safety is high, clarify the compilers and tools to be used for development.
- (8) Examine to what extent the functionality of each safety function need to be guaranteed, by taking account of the frequency of potential defects and failures occurring and the level of impact these incidents may have when they occur, and determine the required level of safety of each safety function (Safety Integrity Level*).

Precaution

- ▶ Examine the specifications of each safety function thoroughly and in detail to ensure that the system safety requirements can be fully met, and sort out the result of this elaborate examination in an orderly manner.
- ▶ Define the requirements on safety functions clearly and strictly, and with consideration on making these definitions usable for verification and testing purposes, such as, in Functional Safety Evaluation and Safety Testing.
- ▶ List all the constraints on safety existing between the software and hardware.
- ▶ In examining the compilers and tools used for development, take their reliability and past performances into consideration.
- ▶ In conceptualizing the levels of safety to be applied to the system under development, refer to widely accepted concepts like "Safety Integrity Level*" advocated in IEC61508.

• Terminology

* Safety Integrity Level (SIL): IEC61508 calls for the use of SIL as the measure for managing the tolerable levels of the rate in which the safety functions that a system is required to achieve become dysfunctional.

1.1.4 Creating the Safety Requirements Specifications

Input

- Product Plan Description
- Product Specifications
- (SA101) List of Potential System Failures
- (SA102) List of System Safety Requirements
- (SA103) System Safety Integrity Level

Outline

Sort out the requirements on product and system safety, and describe them orderly in the form of a document called Safety Requirements Specifications.

Output

- (SA104) Safety Requirements Specifications



Document template sample is available.

Action

- (1) Create the Safety Requirements Specifications by taking account of the items listed in (SA101) and (SA102), and the safety levels defined in (SA103).
- ▶ If multiple alternative draft plans of Safety Requirements Specifications have been examined in prior, the most desirable plan must be selected and finalized in this sub-task.
 - ▶ In conjunction with the above, also organize the

system safety design guidelines and the principles for ensuring safety when the system is used or in operation.

- ▶ State explicitly in the Safety Requirements Specifications, the safety characteristics (safety functions and SIL) that the product is required to provide, including the software requirements on safety functions.

Precaution

- ▶ The requirements pertaining to the following points should be clearly specified in the Safety Requirements Specifications:
 - Impact when the safety is compromised due to, such as, system failure;
 - Mechanism of the system to prevent failures and other safety-inhibiting incidents from occurring;
 - Mechanism to ensure safety when the system is used or in operation;
 - Safety integrity levels that the system is required to meet.
- ▶ Add the outcome of use case analysis (use case diagram, use case scenario, etc) on as-needed basis.
- ▶ The following set of information should also be included to describe the system's safety design guidelines:

- Safety-related design constraints;
- Applicable techniques for safe designing;
- System platform that ensures system safety;
- Warranty of safety gained by reusing the existing system.

- ▶ When there are still uncertainty factors, they should also be stated explicitly in the Safety Requirements Specifications.

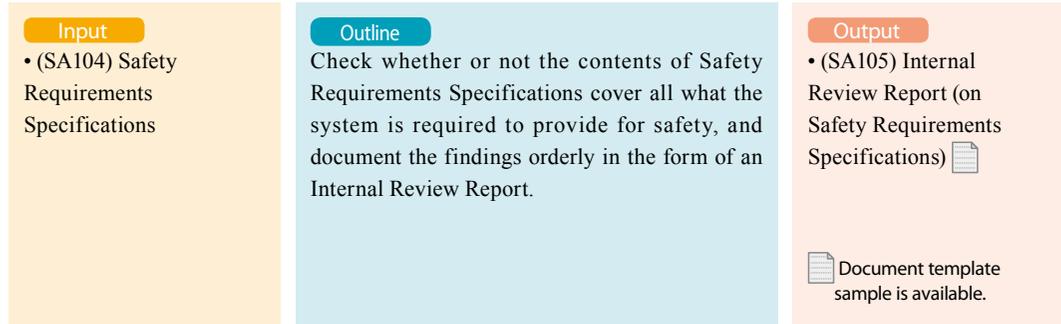
- ▶ Points to keep in mind in documentation:
 - Attach the revision history and indicate clearly where have been revised;
 - Clearly indicate who or which organization is responsible of the created document;
 - Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management; SUP7 Change Management

■ SAP1.2 Reviewing the Safety Requirements Specifications

Confirm that the defined system safety requirements satisfy the product safety requirements.

1.2.1 Reviewing the Safety Requirements Specifications Internally



■ Action

(1) Review the Safety Requirements Specifications (SA104) internally, based on the following perspectives:

- ▶ Are the extent and level of impact on the users and the surroundings when the product or system currently under development becomes defective or causes a failure being examined?
- ▶ Are the safety integrity levels required to apply to the system already determined after considering the frequency of defect and failure occurrences and the level of impact?
- ▶ Is the mechanism to minimize the impact of possible safety-inhibiting factors including disturbances to the system, input data errors and users' operation errors, and maintain the safety of the system being examined?

- ▶ In defining the system safety requirements, are the necessary safety functions implemented in the system as well as the methods and approaches to ensure safety of the system while it is used or in operation (such as, how the system administrator and the users must respond when the system is in abnormal state) being examined?
- ▶ Are the functional system requirements on safety, including the requirements mentioned above, clearly defined?

(2) Document the findings of the above check points orderly in the form of an Internal Review Report (SA105) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

■ Precaution

- ▶ In reviewing the Safety Requirements Specifications, it is desirable to invite the following members as reviewers:
 - Developers and engineers engaged in the current system development;
 - Members who participated in the study group to

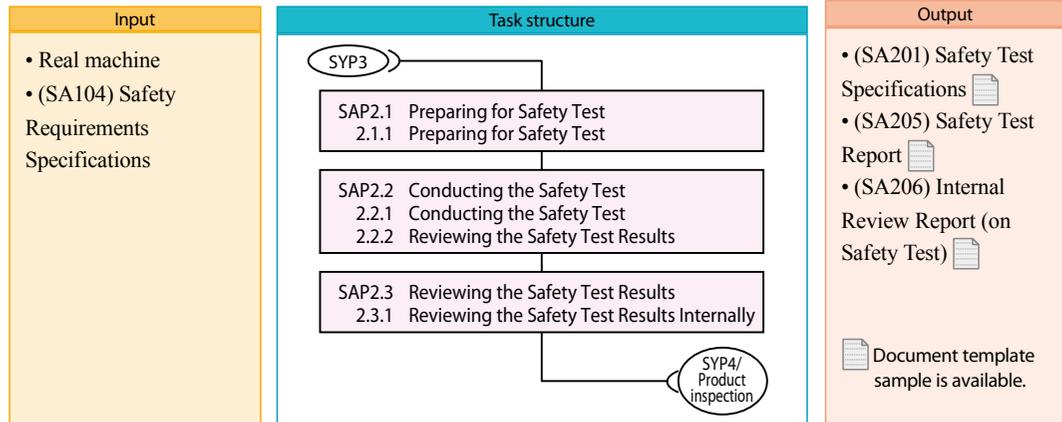
review the Product Specifications and Product Plan Description that serve as the basis for defining the system to be developed;

- Engineers who have been involved in similar system development projects in the past.

- ▶ Review the Safety Requirements Specifications from various perspectives, including the standpoint of system architecture, hardware, safety, performance efficiency and usability.
- ▶ Give attention to the traceability of system safety.
- ▶ The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues.
- ▶ Issues found in the early stage of development when the requirements are defined should be addressed as soon as possible to prevent them from growing or leading into bigger problems in the latter half of the development process. Therefore, the information mentioned in the Internal Review Report should be shared with the stakeholders that include the project manager, development team leader and personnel in charge of product planning, and their consensus should also be built at this early stage.

SAP2 Safety Testing

Check whether the system is satisfying its safety requirements or not.



Description

In this activity:

- (2.1.1) (2.2.1) Prepare the test specifications for checking and reviewing whether or not the system is capable of providing the functionalities and operations described in the Safety Requirements Specifications, and conduct a set of tests collectively referred to as the Safety Test by actually operating the system according to the prepared test specifications (hereafter called the “Safety Test Specifications”);
- (2.2.2) (2.3.1) Review the results of the Safety Test, based on the pass/fail criteria applied to this test, and document the findings orderly in the form of an Internal Review Report. Make a final judgment as the organization in charge of product system development on whether the developed system has passed or failed the Safety Test, based on the test results documented in the Safety Test Report, and sort out the information deemed necessary for later product inspection.

Safety Testing can be regarded as an important opportunity in system development to review the safety of the product. Therefore, in order to prevent the product from causing any harm to the users, this activity must be carried out exhaustively by assuming a wide variety of failures that may occur.

Consideration

- ▶ Before conducting the Safety Test, a particular attention should be given to the operating environment in which the system is tested (test environment). Conduct the Safety Test in a test environment assumed to be close to the environment in which the product is likely to be used by its users in the real world.
- ▶ Consider the System Test as the opportunity in

system development to review the safety of the product, and be careful not to leave out any test cases, including not only the cases where the system is tested in conditions when the product is used in normal state but also the cases where the system is tested in abnormal conditions of use, as well as in defective conditions.

■ <Reference> Techniques and Tools

- ▶ Devices for simulating the system behaviors, etc
- ▶ Reliability growth curve
- ▶ Bug management tool

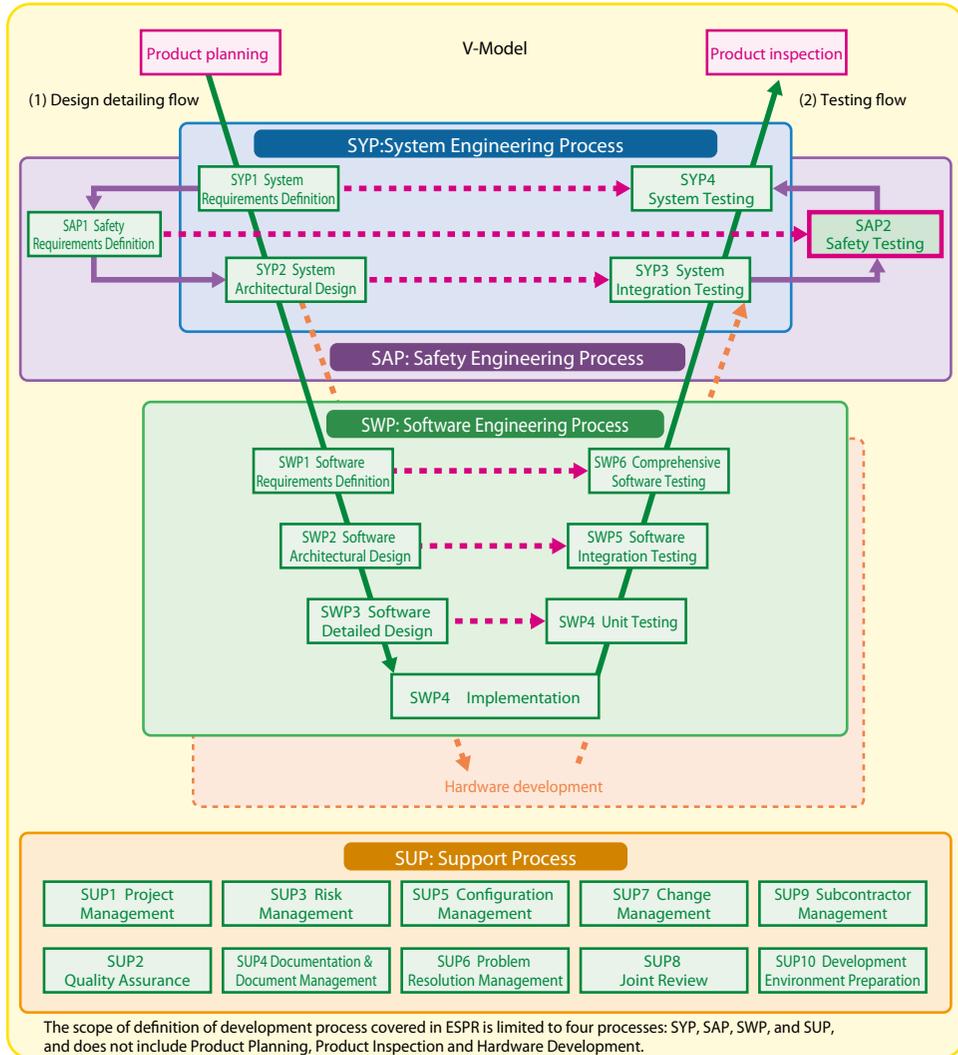


Figure 2.15 V-Model and Development Process (SAP2 Safety Testing)

SAP2.1 Preparing for Safety Test

Prepare for the Safety Test.

2.1.1 Preparing for Safety Test

Input

- (SA104) Safety Requirements Specifications
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Prepare for conducting the Safety Test to the system.

Output

- (SA201) Safety Test Specifications 
 - (SA202) Safety Test Data 
 - (SA203) Internal Confirmation Note (on Safety Test Specifications) 
-  Document template sample is available.

See also

Safety Test Specifications created in the past

Action

- (1) Make a list of items that need to be checked through the Safety Test, and create the test cases for the Safety Test based on this list.
- (2) Think of not only the test cases for testing the safety functions implemented in the system, but also the test cases for validating the methods and approaches taken to ensure safety of the system while it is used or in operation (such as, how the system administrator and the users must respond when the system is in abnormal state).
- (3) Create the test data used to conduct the above-mentioned test cases.
- (4) Also have the various test criteria defined beforehand, including the criteria for judging the test results, the criteria for evaluating the Safety Test on the overall, and the criteria for determining the satisfactory completion of the Safety Testing activity.
- (5) Examine the plan for reviewing and confirming the system safety.
- (6) Sort out the above-mentioned test cases and test data, and create a document called Safety Test Specifications to describe them in an orderly manner.
- (7) Prepare the test cases for verifying the modification (when modifications are implemented and need to be verified).
 - ▶ When a defect is detected during the Safety Testing activity and modification has been implemented to resolve it, prepare test cases to verify that the modification was successful in eliminating the defect, and that no other problems have derived from this modification.
 - ▶ Determine the scope of the modification verification test and select the appropriate test cases, based on the description of the defect.

■ Precaution

- ▶ Examine the test cases also for checking the possibility of disturbances to the system, input data errors and/or users' operation errors
- ▶ Be sure to review the description of the Safety Test Specifications.
- ▶ Examine and select beforehand the methods and techniques to be used in the Safety Testing activity.
- ▶ State the set of information explicitly in the safety test plan:
 - Who runs the tests;
 - When are the tests conducted;
 - What are tested;
 - Test strategies;
 - Test environment, etc.
- ▶ Points to consider when preparing the test for verifying the modification
 - Is the scope of implementation of this test

covering all the potentially affected areas?

- Basically, reuse the test cases that have already been created. Consider preparing new test cases when the modification was large-scaled, and extensive areas were affected.
- ▶ Points to keep in mind in documentation:
 - Attach the revision history and indicate clearly where have been revised;
 - Clearly indicate who or which organization is responsible of the created document;
 - Ensure that the created document is managed properly by performing configuration management and change management.

Related processes: SUP5 Configuration Management;
SUP7 Change Management

SAP2.2 Conducting the Safety Test

Conduct the Safety Test.

2.2.1 Conducting the Safety Test

Input

- Real machine
- (SA201) Safety Test Specifications
- (SA202) Safety Test Data
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Conduct the Safety Test, based on the Safety Test Specifications.

Output

- (SA204) Safety Test Results

Action

(1) Based on the test cases described in the Safety Test Specifications, check the operations and behaviors of the safety functions one by one.

- ▶ When the system is tested by using an alternative test case or data, keep records of the test results, along with the reason(s) stating explicitly why the alternative test case or data had to be used.
- ▶ When a prepared test case cannot be conducted, keep records of the event, along with the reason(s)

stating explicitly why it was not executable. Moreover, determine the reasonableness of the stated reason(s).

(2) Conduct the test for verifying the modification to:

- ▶ Check whether the defect has been eliminated or not by the modification;
- ▶ Check whether the modification to resolve the defect has led to any other defects or not.

Precaution

- ▶ For supplementing the tests for confirming safety, utilize simulation and other useful means whenever necessary.

- ▶ Before running the test for verifying the implemented modification, be sure to check that the latest modified version is tested.

2.2.2 Reviewing the Safety Test Results

Input

- (SA201) Safety Test Specifications System to be tested
- (SA204) Safety Test Results
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Outline

Review the results of the Safety Test, and judge whether the tested system has passed or failed this test.

Output

- (SA205) Safety Test Report 
- (SU601) Defect Management Ticket 

 Document template sample is available.

Action

- (1) Judge whether the result of the Safety Test was satisfactory or not, by referring to the criteria described in the Safety Test Specifications.
 - (2) When a defect has been detected while running the Safety Test, investigate the root cause of the defect (whether it was caused by or in the software or hardware, for example), and document the findings in Defect Management Ticket.
 - (3) In the Safety Test Report, state explicitly the following set of information, among others:
 - ▶ Test methods, test environment, tools and data that have been used;
 - ▶ Number of defects that have been detected, which of them are considered critical, result of the defect analysis;
 - ▶ Judgment of whether the Safety Test was completed successfully or not, and the grounds that support the stated judgment.
- Related processes: SUP6 Problem Resolution Management

Precaution

- ▶ When a defect has been detected while testing, first try reproducing the defect, and clarify the context and the state the system was in.

SAP2.3 Reviewing the Safety Test Results

Review the results of the Safety Test from the stand point of checking whether the requirements defined in the Safety Requirements Definition are realized correctly or not.

2.3.1 Reviewing the Safety Test Results Internally

Input

- (SA104) Safety Requirements Specifications
- (SA201) Safety Test Specifications
- (SA203) Internal Confirmation Note (on Safety Test Specifications)
- (SA205) Safety Test Report
- (SU601) Defect Management Ticket

Outline

Review the contents of the Safety Test Report, check whether or not there have been any issues that could not be solved, and document the findings orderly in the form of an Internal Review Report.

Output

- (SA206) Internal Review Report (on Safety Test) 

 Document template sample is available.

Action

(1) Review the results of the Safety Test from the following perspectives:

- ▶ When an unsolved issue is found:
 - Evaluate the severity (level of importance) of the issue;
 - When the issue is evaluated to be a critical problem that affects the functionality, reliability and/or safety of the entire system, carry out concrete measures by examining the following points, among others:
 - a. Return to system development processes (software, hardware);
 - b. Add restrictions to the conditions of using the system;

c. Reconsider the resource plan.

Related processes: SUP8 Joint Review; SUP1 Project Management

- ▶ Check whether there have been any test cases that were not carried out, and if there were any, investigate the reason(s) why they were not carried out, and examine the possible solutions.
- (2) Document the findings of the above check points orderly in the form of an Internal Review Report (SA206) where the issues raised in the internal review and the personnel in charge of handling these issues are stated explicitly, and distribute this report to the relevant members of the development project.

Precaution

- ▶ Check whether the number of defects that have been detected is acceptable or not, based on the quality criteria.
- ▶ When the test for verifying the modification has been conducted, check whether the scope of the test and the test environment have been appropriate or not.
- ▶ The review manager should make sure that the issues raised in the internal review meetings are included in the Internal Review Report preferably together with possible solutions and/or actions that address these issues (such as, holding joint review meetings).

SUP : Support Process

Support Process (SUP) consists of supportive activities performed across all stages of the development process (SYP, SWP and SAP) to help manage the respective activities (tasks and sub-tasks) defined in these engineering processes and smoothen the organizational execution of embedded software development. The table below shows the activities included in SUP.

Note that in this guidebook (ESPR Ver.2.0), the activities described in pink cells in this table are defined, while in ISO/IEC12207 (for software engineering) and ISO/IEC15288 (for system engineering), more detailed definitions of support process are given, by sub-dividing this process into life cycle process groups like organizational project-enabling processes and SW support processes.

ID	Activity	Outline of the activity	Comprising tasks
SUP1	Project Management	Define the project for developing embedded software and the tasks to proceed with this project smoothly.	SUP1.1 Creating the Project Plan Description SUP1.2 Understanding the Project Execution Status SUP1.3 Controlling the Project SUP1.4 Creating the Project Completion Report
SUP2	Quality Assurance	Define the tasks for enabling elaboration of quality in the course of software development to meet the requirements and market needs on the quality of embedded software currently under development.	SUP2.1 Defining the Quality Objectives SUP2.2 Establishing the Quality Assurance Method SUP2.3 Controlling the Quality Based on Quality Visualization
SUP3	Risk Management	Take measures to grasp, in the early stage of development, the potential risks that may arise in the course of embedded software development.	SUP3.1 Identifying and Understanding the Risks SUP3.2 Monitoring the Risks SUP3.2 Determining and Executing the Risk Treatments
SUP4	Documentation and Document Management	Document the outcome of the activities and tasks performed in SWP in an orderly manner, and define the tasks for managing the created documents.	SUP4.1 Creating and Reviewing the Documents SUP4.2 Distributing the Documents SUP4.3 Maintaining and Managing the Documents
SUP5	Configuration Management	Grasp and manage the individual units that structure the embedded system and the configuration of the design information pertaining to these units.	SUP5.1 Understanding the Objects of Configuration Management SUP5.2 Managing the Configuration Management / Change Management History
SUP6	Problem Resolution Management	Grasp the various problems and issues that arise in the course of development, and manage the solutions implemented to resolve them and the progress made to remedy the problematic situations.	SUP6.1 Recording the Problems and Analyzing the Causes SUP6.2 Analyzing the Impact and Devising the Acceptable Solution SUP6.3 Implementing the Acceptable Solution SUP6.4 Tracking the Implemented Solution
SUP7	Change Management	Manage the changes to the requirements and design that arise after commencing the development and the actions taken to respond to these changes.	SUP7.1 Recording the Information on Change Requests SUP7.2 Analyzing the Impact of Changes SUP7.3 Devising and Executing the Change Plan SUP7.4 Reviewing the Outcome of the Changes Made
SUP8	Joint Review	Check among the relevant stakeholders whether the outcome of current activities and tasks performed at various check gates of the development process have been appropriate or not from both the technical and administrative standpoints.	SUP8.1 Preparing for the Review SUP8.2 Carrying Out the Review SUP8.3 Acknowledging and Following Up on Matters That Have Been Reviewed
SUP9	Subcontractor Management	Define the tasks that would become necessary to outsource any portion of the process.	SUP9.1 Preparing for Order Placement and Entering into Contract SUP9.2 Monitoring the Outsourced Tasks
SUP10	Preparation of Development Environment	Build and manage the environments that would become necessary for carrying out various development process phases ranging from designing, creation of executable modules to testing (environment for implementation, testing, etc).	SUP10.1 Devising the Development Environment Preparation Plan SUP10.2 Building the Development Environment SUP10.3 Maintaining the Development Environment

SUP1 Project Management

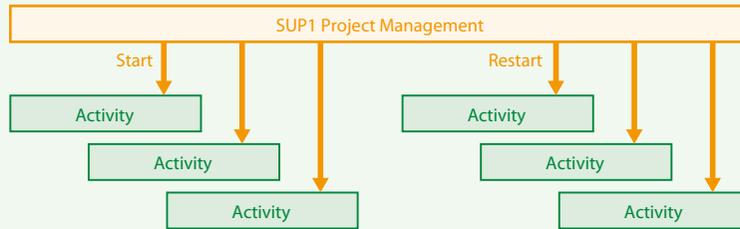
Define the project for developing embedded software and the tasks to proceed with this project smoothly.

Task	Input	Outline of the task	Output
SUP1.1 Creating the Project Plan Description	<ul style="list-style-type: none"> • Specifications of system requirements • Information on development project team members, etc • System resource plan <p>See also Project development plan guidebooks</p>	<p>Create the Project Plan Description (SU101) by carrying out the following sub-tasks:</p> <ol style="list-style-type: none"> (1) Grasp what is to be developed and the specifications of the system requirements; (2) Grasp the length of time that can be allocated for the development project and the constraints on resources, etc; (3) Clarify the objectives of the development project; (4) Examine the activities (tasks and sub-tasks) that need to be carried out in embedded software development, create the WBS (work breakdown structure), and design the development process; (5) Examine who (e.g.: engineers) should be assigned to each of these activities (tasks and sub-tasks). <p>*It is desirable to create the Project Plan Description gradually in a step-by-step manner whenever the information relevant to project plan becomes specific, starting from prior to development to after commencing the development (launching the project).</p>	<ul style="list-style-type: none"> • (SU101) Project Plan Description
SUP1.2 Understanding the Project Execution Status	<ul style="list-style-type: none"> • (SU101) Project Plan Description 	<p>Monitor the project to check whether or not the project is progressing according to the road map described in the Project Plan Description (SU101).</p> <p>Especially check whether or not the project is carried out according to the scope, budget, costs, resources and schedule of the project described in the above document.</p> <p>Also in case of embedded software development, give attention to the consistency with hardware development.</p>	<ul style="list-style-type: none"> • (SU102) Project Status Report

Task	Input	Outline of the task	Output
SUP1.3 Controlling the Project	<ul style="list-style-type: none"> • (SU102) Project Status Report • (SU603) Issue Management Ticket 	<p>When the information in the Project Status Report (SU102) apparently indicates that the project is not progressing smoothly as initially planned, analyze the cause(s) of the delay, and implement the countermeasures examined to be necessary.</p> <p>Such impeding problems and issues that arise while executing the project must be recorded in the Issue Management Ticket (SU603), which needs to be updated and managed without fail.</p> <p>Related processes: SUP6 Problem Resolution Management</p> <p>Moreover, the project plan should be re-examined to cope better with the given situation when deemed necessary.</p>	<ul style="list-style-type: none"> • (SU101) Project Plan Description (in case when the project plan is re-examined) • (SU603) Issue Management Ticket
SUP1.4 Creating the Project Completion Report	<ul style="list-style-type: none"> • (SW105) Software Requirements Specifications • (SW606) Comprehensive Software Test Report • (SU801) Joint Review Records (on Comprehensive Software Test) • (SY106) System Requirements Specifications • (SY406) System Test Report • (SU801) Joint Review Records (on System Test) • (SA104) Safety Requirements Specifications • (SA201) Safety Test Specifications • (SA206) Internal Review Report (on Safety Test) • (SU601) Defect Management Ticket • (SU603) Issue Management Ticket • (SU101) Project Plan Description • (SU102) Project Status Report 	<p>Based on the test results reported in Comprehensive Software Test Report and System Test Report, and the findings from the joint reviews, create a document named Project Completion Report that contains organized description on the final status and outcome of software / system development and other key information on, such as, quality that serves as the basis for determining the product release.</p> <p>Moreover, the issues concerning the development process and techniques applied in the development that came to knowledge while executing the project should also be described in Project Completion Report in an orderly manner, as points that can be improved in the next development project.</p> <p>Upon official issuance of the Project Completion Report, software / system development will leave the hands of the organizations (department, division, etc) in charge of development and will move on to subsequent processes handled by the organizations in charge of manufacturing and product release. Therefore, when preparing the set of information to be entered in the Project Completion Report, bear in mind that this document will later be transferred to the organizations in charge of these subsequent processes for use as one of their primary source materials.</p>	<ul style="list-style-type: none"> • Project Completion Report (on Software Development) • (SU104) Project Completion Report (on System Development) <p> Document template sample is available.</p>

Relationship between Project Management activity and other activities

Project Management activity controls the project by starting or restarting other activities according to the information in Project Plan Description (SU101).



■ Reference Information

In this guidebook, the descriptions on project management are limited to matters pertaining to creating the Project Plan Description, grasping the progress of the project and controlling the project. For general information on project management, Project Management Institute (PMI) has put together a comprehensive document known as PMBOK (Project Management Body of Knowledge) where basic actions in project management are laid out systematically. This document would be a useful reference material for carrying out specific sub-tasks in project management effectively, along with SEC’s booklet “Recommendations to Introduce Project Management”.

Furthermore, the Project Plan Description can be created efficiently by referring to the guidebook “ESMR Ver 1.0: Embedded System development Management Reference [Plan Description Edition]” organized by SEC and using the Project Plan Description template attached to this document.

■ <Reference> Techniques and Tools

- ▶ WBS (Work Breakdown Structure)
- ▶ PERT (Program Evaluation and Review Technique)

SUP2 Quality Assurance

Define the tasks for enabling elaboration of quality in the course of software development to meet the requirements and market needs on the quality of embedded software currently under development.

Task	Input	Outline of the task	Output
SUP2.1 Defining the Quality Objectives	<ul style="list-style-type: none"> • Product Plan Description • (SY106) System Requirements Specifications • (SW105) Software Requirements Specifications 	<p>In order to check how elaborately the quality is refined and predict how the quality can be further enhanced to reach the required level of high quality, set clear quality target levels by taking account of the users and the context in which the system under development is likely to be used. Examine and sort out what the quality target levels should be from the standpoint of functionality, reliability, usability, efficiency, maintainability, and portability, among others, and create the Quality Assurance Plan Description (SU201) based on the findings. The following are some examples of quality metrics to measure the product quality target levels:</p> <ol style="list-style-type: none"> (1) Defect detection rate in design phase and manufacturing phase (number of defects / scale of development) (2) Review rate in design phase and manufacturing phase (length of review time / scale of development) (3) Test density (Number of test cases / scale of development) (4) Failure detection rate in test phase (Number of failures / scale of development) (5) Failure convergence rate in test phase ((Cumulative number of failures that occurred during the test phase / targeted number of failures) x 100) <p>Build a consensus among the stakeholders beforehand on the quality objectives.</p>	<ul style="list-style-type: none"> • (SU201) Quality Assurance Plan Description
SUP2.2 Establishing the Quality Assurance	<ul style="list-style-type: none"> • (SU201) Quality Assurance Plan Description 	<p>Clarify the structure of the organizations that will be responsible of assuring quality, how these organizations will share the reasonability of quality assurance, and what kind of routine will be taken to assure quality. To build the mechanism of measuring the rate of achievement of the quality objectives, clearly define the following rules for administering quality assurance through the mechanism to:</p> <ol style="list-style-type: none"> (1) Gather quality-related information (who gathers the information when and how (by which means), and reports to whom); (2) Utilize quality-related information (who utilizes the information based on what, and for what purposes). <p>Clarify who and/or which organization(s) are responsible of promoting the quality assurance tasks, as well as who and/or which organization(s) are held responsible of quality-related matters.</p>	<ul style="list-style-type: none"> • (SU201) Quality Assurance Plan Description

Task	Input	Outline of the task	Output
SUP2.3 Controlling the Quality Based on Quality Visualization	<ul style="list-style-type: none"> • Various deliverables (specifications, plan descriptions, source code, real machine) • (SU201) Quality Assurance Plan Description 	<p>(1) In the key events for assuring the system quality (reviews and tests), measure the quality achievement rate of the deliverables based on the quality target levels defined in the Quality Assurance Plan Description.</p> <p>(2) When a large gap is identified between the quality of any of the reviewed deliverables and the quality target level, devise and implement one or more measures to bridge the gap. As for these measures, take necessary actions by coordinating with the engineers in charge of design and the project manager.</p>	<ul style="list-style-type: none"> • (SU201) Quality Assurance Plan Description

■ Reference Information

To develop a system and software as a product, there is a need to set clear targets on the level of quality to be achieved, and determine in advance the mechanism of assuring quality. In setting the quality target levels to be achieved by the system and software respectively, specify target values so that the quality level can be measured quantitatively, and define the quality assurance structure and mechanism to enable the developed system and software to achieve their pre-specified targets. Also, be sure to set the schedule and method of holding essential events to achieve the quality objectives, including reviews and tests. In addition, promote the quality assurance activity by taking account of the system requirements on safety.

■ <Reference> Techniques and Tools

- ▶ Reliability growth curve

SUP3 Risk Management

Define the tasks to implement the measures to grasp, in the early stage of development, the potential risks that may arise in the course of embedded software development.

Task	Input	Outline of the task	Output
SUP3.1 Identifying and Understanding the Risks	<ul style="list-style-type: none"> Information on case examples of troubles and risks in similar projects carried out in the past Product Plan Description Product Specifications 	<ol style="list-style-type: none"> Set the basic policies on risk management of the project and define the mechanism to put them into practice by taking the following points into account: <ul style="list-style-type: none"> Method of management (how to identify, analyze, prioritize, plan, monitor and solve the risks, among others); Responsibility and authority of risk management. Identify the potential risks that may arise in the course of the project. There are various ways of extracting the risks, including the ones below: <ul style="list-style-type: none"> Look into the case examples of troubles and risks that arose in similar projects in the past; Examine the potential risks through reviews of specifications, plan descriptions and other relevant documents; Identify the risks by using cause-and-effect diagrams and decision trees. For each risk that has been extracted, examine the probability of occurrence, triggers, level of impact when it arises (extent of impact, man-hours, etc), treatments and their order of priority, contingency plan, among others. Sort out the findings in the form of a risk table, for example, and create the Risk Management Plan Description (SU301). In case of embedded system / software development, give particular attention to risks that are attributable to, such as, hardware-related matters, operating environment, and business environment. 	<ul style="list-style-type: none"> (SU301) Risk Management Plan Description

Task	Input	Outline of the task	Output
SUP3.2 Monitoring the Risks	<ul style="list-style-type: none"> • (SU301) Risk Management Plan Description • Various deliverables (specifications, plan descriptions, source code, real machine) 	<p>(1) Monitor the risks during the key events (reviews, tests) in system development with an eye to manage the risks by checking if the risk treatments are put into practice as planned, and whether the pre-defined order of priority of the risks can be maintained or need to be adjusted in the given conditions.</p> <p>(2) Update the Risk Management Plan Description (SU301) when new risks are identified or when any of the risks have been found to be successfully eliminated and can therefore be omitted from the risk table described in the above document, as a result of monitoring.</p> <p>The risks to be monitored are not limited to initial risks identified in the early stage of the project. Risk monitoring must continue to be carried out periodically along with the progress of the project.</p>	<ul style="list-style-type: none"> • (SU301) Risk Management Plan Description
SUP3.3 Determining and Executing the Risk Treatments	<ul style="list-style-type: none"> • (SU301) Risk Management Plan Description 	<p>(1) Eliminate the risk factors before the risk arises.</p> <p>(2) When there are risks that cannot be eliminated, carry out the contingency plan to mitigate the probability of occurrence of such risks,</p> <p>(3) When a risk arises, execute the risk treatments according to the plan described in the Risk Management Plan Description (SU301).</p> <p>In executing the risk treatments, take necessary actions with the engineers in charge of design and the project manager.</p>	<ul style="list-style-type: none"> • (SU301) Risk Management Plan Description

■ Reference Information

In the course of system and software development, various problems are bound to occur and impact the efforts to achieve the quality objectives. However, many of these problems are often predictable. Therefore, there is usually a need to investigate the hidden risk factors and identify the potential risks both before and after commencing the project. And after the project has been launched, appropriate risk management must be carried to check whether or not there are any problems arising at the given moment, and also whether the treatments for foreseeable risks are put into practice to prevent those risks from actually occurring.

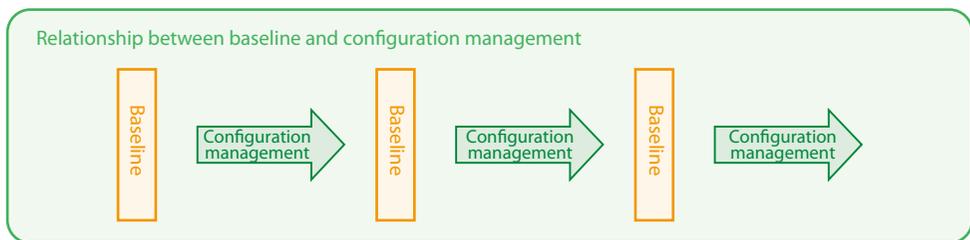
■ <Reference> Techniques and Tools

- ▶ Techniques applicable for risk analysis
 - FTA (Fault Tree Analysis)
 - FMEA (Failure Modes and Effects Analysis)

SUP5 Configuration Management

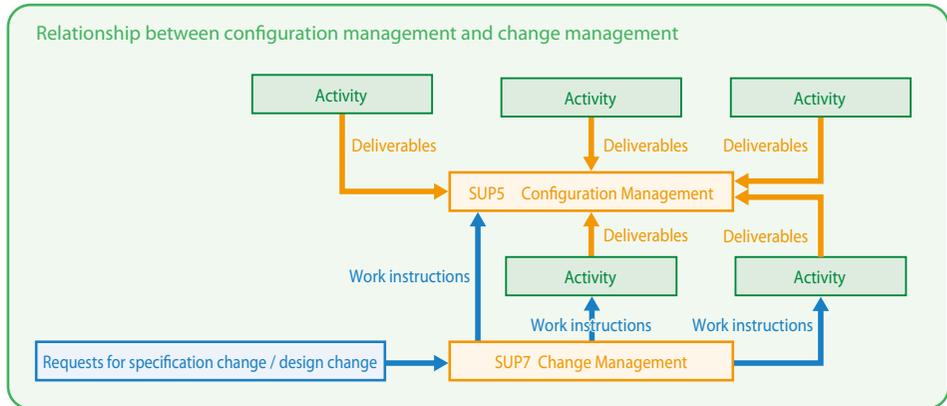
Define the tasks to sort out and manage the outcome of the activities performed in SYP, SWP and SAP (deliverables like design descriptions and source codes).

Task	Input	Outline of the task	Output
SUP5.1 Understanding the Objects of Configuration Management	<ul style="list-style-type: none"> • Objects of configuration management (Activity deliverables, development environment, integrated system, etc) 	<ol style="list-style-type: none"> (1) Clarify the policy and mechanism of configuration management. <ul style="list-style-type: none"> • Configuration management procedure and rules Use, for example, a table that lists the deliverables and shows their chronological associativity so that they can be traced. • Baseline schedule and audit policy, etc (2) Grasp and determine the objects of configuration management. These objects are outputs from activities, and include the following: <ul style="list-style-type: none"> • Documents (specifications, design descriptions, reports, etc) • Source code, integrated system, etc • Development environment 	<ul style="list-style-type: none"> • (SU501) Configuration Management Table
SUP5.2 Managing the Configuration Management / Change Management History	<ul style="list-style-type: none"> • Objects of configuration management (Activity deliverables, development environment, integrated system, etc) • (SU501) Configuration Management Table 	<ol style="list-style-type: none"> (1) When there are changes in the objects of configuration management, update the baseline and revision history, and manage the old and new versions. (2) Control the access to the baseline. (3) Approve the generation of deliverables from the baseline. (4) Distribute the configuration management information to relevant stakeholders. <p>In case of embedded software development, changes in specifications occur frequently in the course of development. Therefore, these changes need to be managed without fail (describe explicitly the contents of the changes as well as the reason(s) for the change).</p> <p>*Be careful with the handling of baseline and branch.</p>	<ul style="list-style-type: none"> • Objects of configuration management (Activity deliverables, development environment, integrated system, etc) • (SU501) Configuration Management Table



- Terminology

* Baseline management : A technique in change management where a configuration item at a specific point in time is set as the "baseline" (or the starting point), and used to manage the changes that occur from that point onward. The term "branch" refers to the item that derived from the baseline.



■ Reference Information

Recent system and software development projects are often planned to develop products in series, and tend to have multiple system and software versions developed concurrently. In such forms of development, version management plays a significant role, because appropriate management of information that specifically tells which functionalities have been implemented for which version is of vital importance.

Moreover, in such cases, a database to manage all the deliverables collectively is normally prepared. Following the “check-in / check-out” procedure is a desirable method to develop a new version derived from one of the deliverables stored in this database. Derivative versions must be properly managed in order to prevent the users from making mistakes like selecting the wrong files instead of the product release version files that they were actually looking for. Especially when a defect is found and a need to search for all the related versions subject to modification arises, confusion in version tracking may cause a serious impact on the progress of the development. Therefore, the relationship between individual versions must be clearly understood. If necessary, create a chart showing which versions are co-related to help understand the associativity of the versions more easily.

■ <Reference> Techniques and Tools

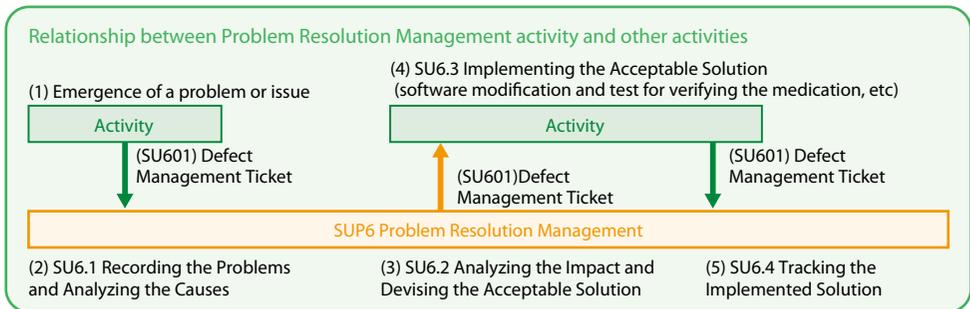
- ▶ Deliverables Management Database

SUP6 Problem Resolution Management

Grasp the various problems and issues that arise in the course of development, and manage the solutions implemented to resolve them and the progress made to remedy the problematic situations.

Task	Input	Outline of the task	Output
SUP6.1 Recording the Problems and Analyzing the Causes	<ul style="list-style-type: none"> • (SU601) Defect Management Ticket • (SU603) Issue Management Ticket 	<p>(1) Record in the Defect Management Ticket (SU601) all the defects found in the system or software while reviewing or testing the deliverables created in the course of development, assign an Item No. to each defect, and manage these itemized defects through the Defect Management Register (SU602).</p> <p>(2) Record in the Issue Management Ticket (SU603) all the issues that affect the execution of the project, assign an Item No. to each issue, and manage these itemized issues through the Issue Management Register (SU604).</p> <p>Related processes: SUP1 Project Management</p>	<ul style="list-style-type: none"> • (SU602) Defect Management Register  • (SU604) Issue Management Register
SUP6.2 Analyzing the Impact and Devising the Acceptable Solution	<ul style="list-style-type: none"> • (SU601) Defect Management Ticket • (SU602) Defect Management Register • (SU603) Issue Management Ticket • (SU604) Issue Management Register 	<p>(1) Analyze and examine the extent of impact of the defects and issues recorded in either the Defect Management Ticket (SU601) or Issue Management Ticket (SU603).</p> <p>Moreover, based on the result of the aforesaid analysis and examination, also analyze the level of severity and urgency of the problems caused by these defects and issues, and categorize them according to the findings from this subsequent analysis.</p> <p>(2) Investigate the root cause of the defects and issues, based on the result of the analysis and categorization carried out in (1).</p> <p>(3) Devise acceptable solutions to deal with the defects and issues, and record them in either the Defect Management Ticket (SU601) or the Issue Management Ticket (SU603).</p>	<ul style="list-style-type: none"> • (SU601) Defect Management Ticket  • (SU602) Defect Management Register  • (SU603) Issue Management Ticket • (SU604) Issue Management Register
SUP6.3 Implementing the Acceptable Solution	<ul style="list-style-type: none"> • (SU601) Defect Management Ticket • (SU602) Defect Management Register • (SU603) Issue Management Ticket • (SU604) Issue Management Register 	<p>Implement the acceptable solutions recorded in either the Defect Management Ticket (SU601) or the Issue Management Ticket (SU603).</p>	<ul style="list-style-type: none"> • (SU601) Defect Management Ticket  • (SU602) Defect Management Register  • (SU603) Issue Management Ticket • (SU604) Issue Management Register

Task	Input	Outline of the task	Output
SUP6.4 Tracking the Implemented Solution	<ul style="list-style-type: none"> • (SU601) Defect Management Ticket • (SU602) Defect Management Register • (SU603) Issue Management Ticket • (SU604) Issue Management Register 	Check whether or not the implemented solutions have been able to resolve the defects and issues.	<ul style="list-style-type: none"> • (SU601) Defect Management Ticket • (SU602) Defect Management Register • (SU603) Issue Management Ticket • (SU604) Issue Management Register • Document template sample is available.



Reference Information

The purpose of “SUP6 Problem Resolution Management” is to appropriately process the defects that are detected or indicated in reviews and tests conducted on interim and final deliverables created in the course of system or software development. Beside these defects, there are also various problems and issues that arise during the execution of the development project.

Whatever the potential issues and problems may be, regardless of whether they arise during the activities and tasks performed in the engineering process or in the support process, the important thing is to recognize and visualize each of them as a problem, determine the severity and urgency of each problem based on the extent of impact, and carry out concrete measures to resolve them.

“SUP6 Problem Resolution Management” should be considered as an activity that covers all the problems and issues that not only arise during the specific activities and tasks mentioned above, but are also intrinsic to the processes themselves. (In pointing out the problems and issues intrinsic to the processes, there may be a need to broadly interpret the defects described in (SU601) Defect Management Ticket.)

To implement the concrete measures that have been devised as acceptable solutions to the detected problems and issues, it is desirable to clearly define “who” these solutions are to be implemented “by when”, as well as the criteria for determining the completion of the implemented solution (the state in which the problem or issue has been completely resolved by the implemented solution).

<Reference> Techniques and Tools

- ▶ Program slicer
- ▶ Defect management tools

SUP7 Change Management

Manage the changes to the requirements and design that arise after commencing the development and the actions taken to respond to these changes.

Task	Input	Outline of the task	Output
SUP7.1 Recording the Information on Change Requests	<ul style="list-style-type: none"> Change request (from external sources) 	When external sources, including the customers and hardware division, request for a change in specifications, design or other matters after commencing the development, record these change requests in the Change Request Management Ticket (SU701) all after carefully reviewing the contents of each request. Assign an Item No. to each change request, and manage these itemized change requests through the Change Request Management Register (SU702).	<ul style="list-style-type: none"> (SU701) Change Request Management Ticket (SU702) Change Request Management Register
SUP7.2 Analyzing the Impact of Changes	<ul style="list-style-type: none"> (SU701) Change Request Management Ticket (SU702) Change Request Management Register 	<p>Closely analyze and evaluate each change request item described in the Change Request Management Ticket (SU701), mainly from the following perspectives.</p> <p>If the requested change is accepted:</p> <ul style="list-style-type: none"> Which part of the specifications and/or design will be affected? Which portion of the related documents will be affected? What kind of impact will there be on the work carried out by other divisions? 	<ul style="list-style-type: none"> (SU701) Change Request Management Ticket (SU702) Change Request Management Register
SUP7.3 Devising and Executing the Change Plan	<ul style="list-style-type: none"> (SU701) Change Request Management Ticket (SU702) Change Request Management Register 	Execute the requested change after determining the scope of the requested change and the timing to work on it, based on the contents described in the Change Request Management Ticket (SU701), and notifying all the related organizations about the changes that are going to be made. On change requests that affect the specifications, bear in mind the preliminary need to review and gain consensus among the relevant stakeholders and create a document on points that have been reviewed and agreed on. 	<ul style="list-style-type: none"> (SU701) Change Request Management Ticket (SU702) Change Request Management Register (SU703) Revised deliverables
SUP7.4 Reviewing the Outcome of the Changes Made	<ul style="list-style-type: none"> (SU701) Change Request Management Ticket (SU702) Change Request Management Register (SU703) Revised deliverables 	Check whether the changes that have been made fully meet the contents of the change request or not.	<ul style="list-style-type: none"> (SU701) Change Request Management Ticket (SU702) Change Request Management Register

 : Work related to safety

■ Reference Information

In embedded software development, various changes in specifications and design tend to occur rather frequently after commencing the development. When requests for such changes come in, there is a need to evaluate how they affect other parts of the software before implementing the requested changes. Close attention is necessary especially on requests for changing the internal components of the software that are highly interdependent or tightly binding with others.

Repeated changes to the software may complicate the internal structure of the software. Therefore, those evaluating the change requests that affect the software structurally should always be aware of the software architecture.

■ <Reference> Techniques and Tools

- ▶ Intersolv PVCS/VCS (Version Control System)

SUP8 Joint Review

Check among the relevant stakeholders whether the outcome of current activities and tasks performed at various check gates of the development process have been appropriate or not from both the technical and administrative standpoints.

Task	Input	Outline of the task	Output
SUP8.1 Preparing for the Review	• Deliverables to be reviewed	Prepare for the review by taking account of the project status, selecting the deliverables to be reviewed, and determining when to hold the review meeting(s), and who the participants of the review would be.	(Review meeting notice, etc)
SUP8.2 Carrying Out the Review	• Deliverables to be reviewed	Review the deliverables selected in the preceding task from both technical and administrative standpoints. Preferably, distribute in advance the deliverables to be reviewed to the participants of the review meetings, so that they can check the contents of the distributed deliverables beforehand. In the review meetings, focus mainly on detecting problems in the deliverables, and record the findings in the form of a document (SU801 Joint Review Records).	• (SU801) Joint Review Records 
SUP8.3 Acknowledging and Following Up on Matters That Have Been Reviewed	• (SU801) Joint Review Records	Distribute the Joint Review Records (SU801) in which the review points confirmed as matters examined in the review meetings are described in an orderly manner, not only to the participants of the review meetings, but also to other non-participating stakeholders. Moreover, continue tracking the subsequent efforts to check whether the solutions to the matters indicated in the review have been examined and implemented or not.	• (SU801) Joint Review Records   Document template sample is available.

■ Reference Information

Reviews can be regarded as one of the important opportunities in the course of software development to elaborate the software quality. It is desirable to review to deliverables created in the engineering processes at every check gate of the development process. Reviews can be held in various formats, ranging from internal reviews conducted within the development group (such as, peer reviews and cross-checks) to joint reviews described herein as one of the activities of SUP.

Joint reviews are opportunities not only for the engineers in charge of creating the deliverables to review them but also for other stakeholders involved in the product development to participate and check the deliverables from multiple perspectives. The review manager should appoint an individual to examine each issue or problem identified in these reviews, and agree with this appointee on the due date to record the findings in the Joint Review Records, so that follow-up actions to resolve the issues and problems can be taken smoothly.

■ <Reference> Techniques and Tools

- ▶ CBR (Checklist-based Reading)
- ▶ PBR (Perspective-based Reading)

SUP10 Preparation of Development Environment

Build and manage the environments that would become necessary for carrying out various development process phases ranging from designing, creation of executable modules to testing (environment for implementation, testing, etc).

Task	Input	Outline of the task	Output
SUP10.1 Devising the Development Environment Preparation Plan	<ul style="list-style-type: none"> • (SU101) Project Plan Description • (SW105) Software Requirements Specifications • (SW205) Software Architectural Design Description • (SW305) Software Detailed Design Description • (SW401) Unit Test Specifications • (SW501) Software Integration Test Specifications • (SW601) Comprehensive Software Test Specifications 	<ol style="list-style-type: none"> (1) Devise a plan for preparing the various development environments ranging from designing to creation of executable modules that are necessary to carry the software development forward. (2) Devise a plan for building the environment for debugging and testing (ranging from unit testing to comprehensive testing), by taking account of the hardware development plan, development period, budget, number of human resources and required skills, among others. (3) Devise a plan for preparing the administrative work environment (for progress management, deliverables management, quality assurance, specifications management, etc). 	<ul style="list-style-type: none"> • (SU1001) Development Environment Preparation Plan Description
SUP10.2 Building the Development Environment	<ul style="list-style-type: none"> • (SU1001) Development Environment Preparation Plan Description 	<ol style="list-style-type: none"> (1) Build the necessary environments according to the respective plan by the time they become necessary. (2) Provide training and education to the environment users by the time they start using their respective environments. (3) Confirm that the environments that have been built are fully usable. (4) Solve all problems concerning the delay in making the environments available, inferior quality, insufficient volume / capacity, among others. 	<ul style="list-style-type: none"> • (SU1002) Software Development Environment
SUP10.3 Maintaining the Development Environment	<ul style="list-style-type: none"> • (SU1002) Software Development Environment 	<ol style="list-style-type: none"> (1) Maintain the development environment constantly by updating it whenever the source code is revised, tools are upgraded (to newer versions), and other significant enhancements / renewals are in place. (2) Confirm that the development environment is fully usable after updating it. 	<ul style="list-style-type: none"> • (SU1002) Software Development Environment

Reference Information

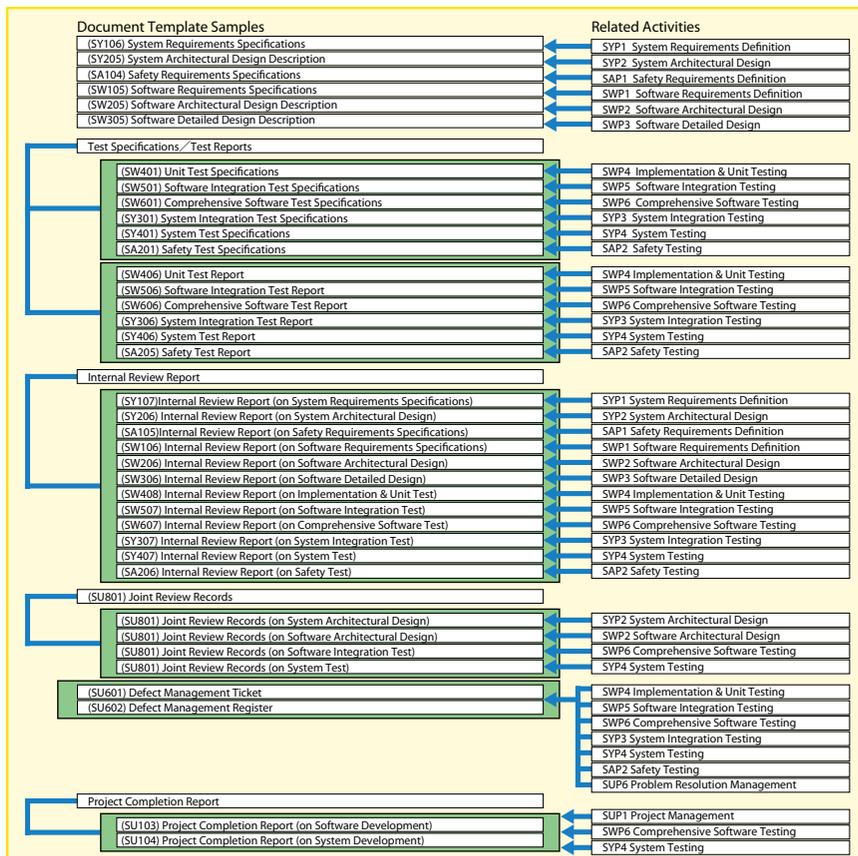
The preparation and building of development environment tend to be delayed. To proceed with the software development smoothly, devise the environment construction plan early and build the environment according to the planned schedule.

“Development environment” is a collective term that refers to various environments, including the environment for software development, environment for debugging and testing, and administrative work environments (for progress management, deliverables management, quality assurance, specifications management, etc).

2.3 Document Template Samples

Among the documents used for organizing the outcome of individual tasks and sub-tasks as discussed under “2.2 Process Definition Documents”, this guidebook provides template samples for some of the key documents to help promote the efficiency of documentation at actual development sites. (See chart below.)

The items, contents and examples of information to be entered in each document template sample are provided just for reference and do not imply any specific ways of using them. Therefore, for actual documentation, there is a need to arrange the items and terminology used in each document to those that best suit the specific interests of individual development projects.



(SY106) System Requirements Specifications

System Requirements Specifications is a document used to describe the functional and non-functional requirements that the system has to achieve, the constraints and various other matters that have been examined in the activity to define the system requirements.

■ Sub-tasks to create this document

- SYP1 System Requirements Definition
 - 1.1 Creating the System Requirements Specifications
 - 1.1.6 Creating the System Requirements Specifications

■ Information referenced to prepare this document

Product Plan Description	(SY103) List of Non-functional System Requirements
Product Specifications	(SY104) List of System Operational Constraints
(SY101) List of Functional System Requirements	(SY105) Prioritized List of System Requirements
(SY102) System Functionality-Operation Matrix	

■ Example of items to be described

- | | |
|-------------------------------------|--|
| 1. Overview | 6. Functional Details |
| 2. System Structure | 7. Detailed Non-functional Requirements on Performance, Quality, etc |
| 3. Functional Overview | 8. Others |
| 4. Constraints | |
| 5. Use Cases and Use Case Scenarios | |

■ Sub-tasks that make use of this document

SYP1 System Requirements Definition <ul style="list-style-type: none">1.2 Confirming the System Requirements Specifications<ul style="list-style-type: none">1.2.1 Reviewing the System Requirements Specifications Internally	2.3 Jointly Reviewing the System Architectural Design <ul style="list-style-type: none">2.3.1 Jointly Reviewing the System Architectural Design Description
SYP2 System Architectural Design <ul style="list-style-type: none">2.1 Creating the System Architectural Design Description<ul style="list-style-type: none">2.1.1 Confirming the Design Conditions2.1.2 Designing the System Structure2.1.3 Designing the Overall System Behaviors2.1.4 Designing the Interface2.2 Reviewing the System Architectural Design<ul style="list-style-type: none">2.2.1 Reviewing the System Architectural Design Description Internally	SWP1 Software Requirements Definition <ul style="list-style-type: none">1.1 Creating the Software Requirements Specifications<ul style="list-style-type: none">1.1.1 Identifying the Constraints1.1.2 Clarifying the Functional Software Requirements1.1.3 Clarifying the Non-functional Software Requirements1.2 Reviewing the Software Requirements Specifications

1.2.1 Reviewing the Software Requirements
Specifications Internally

SYP4 System Testing

4.1 Preparing for System Test

4.1.1 Creating the System Test Specifications

4.1.3 Reviewing the System Test Specifications
Internally

4.3 Reviewing the System Test Results

4.3.1 Reviewing the System Test Results
Internally

4.4 Confirming the Completion of System
Development

4.4.1 Confirming the Completion of System
Development

SUP1 Project Management

1.4 Creating the Project Completion Report

■ (SY106) System Requirements Specifications (Example)

Cover page

Name of document

Document No.

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				
			⋮	

Document No. Page No. Date of issue / Issued by

Provide a column to enter the revised information.

Table of Contents

Name of document

Table of Contents

1. Overview Page No.

2. System Structure Page No.

3. Functional Overview Page No.

4. Constraints Page No.

5. Use Cases and Use Case Scenarios Page No.

6. Functional Details Page No.

7. Detailed Non-functional Requirements
on Performance, Quality, etc. Page No.

8. Others Page No.

Document No. Page No. Date of issue / Issued by

1. Overview

- <Exemplary descriptions>
- Purpose of this document
 - Positioning of this document
 - Intended users
 - Scope of description, contents, etc
 - Referenced documents, etc
 - Definition (terminology, acronyms, etc)

Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.

2. System Structure

- <Exemplary descriptions>
- Overall system structure
 - Name / basic functionalities of each system component
 - Operating environment of the system and its external environment

Describe the structure of the system as a whole, including both the hardware and software, the relationship between the system components, and conditions to operate and use the system.

3. Functional Overview

- <Exemplary descriptions>
- List of functionalities achieved / provided by the system, along with their general description.

Provide the general description of the functionalities in the form of a list. Provide the detailed description of the functionalities separately later on. (See #6 below.)

4. Constraints

- <Exemplary descriptions>
- Length of time (development period) and budget (development costs) that can be allocated for product development
 - Serviceable life of the product (durable years), and product life cycle
 - Expected product quality (reliability, safety, usability, etc)
 - Other system(s) and hardware used as the prerequisites for using the product (peripheral system(s) and hardware)
 - Constraints on the continuity from the existing product specifications
 - Constraints on reusing the existing system
 - Constraints due to security and environmental issues
 - Constraints from laws and social conventions that correspond to system use
 - Association with intellectual properties and technologies of other owners

Describe all the constraints without leaving out any.

5. Use Cases and Use Case Scenarios

- <Exemplary descriptions>
- Use cases and use case scenarios written on the basis of situations and context in which the product is intended / assumed to be used

Take account of the interactions between the users and the system for each functionality constituting the system, and describe the flow of these interactions in chronological order.

6. Functional Details

- <Exemplary descriptions>
- Detailed description of functionalities for realizing the use cases described above in #5.

Describe each functionality in detail.

7. Detailed Non-functional Requirements on Performance, Quality, etc

<Exemplary descriptions>

- Requirements on reliability
 - System's abnormal processing methods
 - System's recovery procedures and methods from abnormal operating mode
- Requirements on usability
 - User-friendliness of the user interface
- Requirements on efficiency
 - System's execution performance (e.g.: processing speed, start-up time, response time, etc)
 - Degree of real-time processing
 - Resource efficiency (e.g.: memory capacity, data size)
- Requirements on maintainability
 - Maintenance methods (such as, remote maintenance) and techniques to realize them
- Requirements on portability
 - Independency of system functionalities
- Requirements on security
 - (Example) Data encryption, user authentication, anti-virus measures, etc

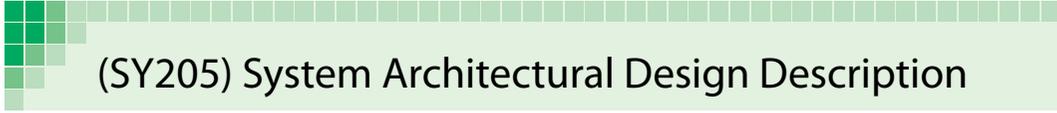
Describe the matters related to performance and quality that cannot be expressed as functional matters.

8. Others

<Exemplary descriptions>

- Interoperability (e.g.: communication protocol, etc)
- Requirements on external interfaces (e.g.: interface with peripheral systems, user interface, etc)

Describe miscellaneous matters that are worthy of special mention.



(SY205) System Architectural Design Description

System Architectural Design Description is a document used to describe how the system can achieve its requirements (software structure, control method, etc) that have been examined in the activity to design the system architecture.

■ Sub-tasks to create this document

SYP2 System Architectural Design

- 2.1 Creating the System Architectural Design Description
 - 2.1.5 Creating the System Architectural Design Description

■ Information referenced to prepare this document

(SY201) System Structural Diagram (Functional Block Diagram)
(SY202) List of Common Functionalities and Data

(SY203) System Behavioral Design Description
(SY204) System Interface Design Description

■ Example of items to be described

- | | |
|---|--|
| 1. Overview | 4.3 Performance Estimation |
| 2. System Structure | 5. Detailed Description of Functional Blocks |
| 3. General Description of Functional Blocks | 6. Data Handled by the System |
| 4. Control Method | 7. List of Exceptions |
| 4.1 Control Sequence | 8. Others |
| 4.2 Use Cases and Corresponding Functional Blocks | |

■ Sub-tasks that make use of this document

SYP2 System Architectural Design

- 2.2 Confirming the System Architectural Design
 - 2.2.1 Reviewing the System Architectural Design Description Internally
- 2.3 Jointly Reviewing the System Architectural Design
 - 2.3.1 Jointly Reviewing the System Architectural Design Description

SWP1 Software Requirements Definition

- 1.1 Creating the Software Requirements Specifications
 - 1.1.1 Identifying the Constraints
 - 1.1.2 Clarifying the Functional Software Requirements

1.1.3 Clarifying the Non-functional Software Requirements

- 1.2 Reviewing the Software Requirements Specifications
 - 1.2.1 Reviewing the Software Requirements Specifications Internally

SWP2 Software Architectural Design

- 2.1 Creating the Software Architectural Design Description
 - 2.1.1 Confirming the Design Conditions
 - 2.1.2 Designing the Software Structure
 - 2.1.3 Designing the Overall Software Behaviors
 - 2.1.4 Designing the Interface

SYP3 System Integration Testing

3.1 Preparing for System Integration Test

3.1.1 Preparing for System Integration

3.1.2 Preparing for System Integration Test

3.3 Reviewing the System Integration Test Results

3.3.1 Reviewing the System Integration Test Results Internally

■ (SY205) System Architectural Design Description (Example)

Cover page

Name of document

Document No.

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				

⋮

⋮

⋮

Document No. Page No. Date of issue / Issued by

Provide a column to enter the revised information.

Table of contents

Name of document

Table of Contents

1. Overview Page No.

2. System Structure Page No.

3. General Description of Functional Blocks Page No.

4. Control Method Page No.

 4.1 Operation Sequence Page No.

 4.2 Use Cases and Corresponding
 Functional Blocks Page No.

 4.3 Performance Estimation Page No.

5. Detailed Description of Functional Blocks Page No.

6. Data Handled by the System Page No.

7. List of Exceptions Page No.

8. Others Page No.

Document No. Page No. Date of issue / Issued by

Text (1)	Name of document	
<p>1. Overview</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Purpose of this document · Positioning of this document · Intended users · Scope of description, contents, etc · Referenced documents, etc · Definition (terminology, acronyms, etc) 	<p>Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.</p>	
<p>2. System Structure</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Overall system structure - Name / basic functionalities of each system component · Main functionalities (components) that structure the system, and the division of roles played respectively by the hardware and software <p>For example: Hardware platform (name of MPU, types of I/O connectors, hardware interrupt levels and I/O responses, memory types and size), software platform (OS, middleware, input / output data formats, various performance indicators), peripheral devices (types of external media, control methods, etc)</p>	<p>Describe the structure of the system as a whole, including the division of roles played respectively by the hardware and software, the names of system components, and basic functionalities, among others.</p>	
<p>3. General Description of Functional Blocks</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Names of functional blocks that structure the system, their basic functionalities, etc · Interface between functional blocks <p>For example: Show the static relationship between the functional blocks (in terms of control and data flow), and if necessary, also describe the OS, memory, external storage media, and hardware mechanism. Clarify the core system functionalities.</p>	<p>Provide the general description of the functional blocks realized and provided by the system. Provide the detailed description of the functional blocks later on(See #5 below).</p>	
<p>4. Control Method</p>	<p>Mainly from the standpoint of functional blocks, describe how the system as a whole (hardware, software, operators, external storage media, memory, etc fulfills the requirements specified in the System Requirements Specifications (SY106).</p>	
<p>4.1 Operation sequence</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Operation Sequence Diagrams 	<p>Describe how the functional blocks interact to realize the functional services of the system.</p>	
<p>4.2 Use Cases and Corresponding Functional Blocks</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Use cases, use case scenarios and corresponding functional blocks 	<p>Describe the use cases and the corresponding functional blocks based on the use cases examined in system requirements analysis.</p>	
<p>4.3 Performance Estimation</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Required performance / time: xx (ms) · Method of achieving the required level of performance: Given conditions and processing sequence · Estimate processing time: Estimate processing elements and the grounds for the estimation 	<p>Describe the estimate time needed for software and hardware operations by considering the non-functional system requirements and the time constraints on system operation (such as, the system's response time).</p>	
Document No.	Page No.	Date of issue / Issued by

5. Detailed Description of Functional Blocks

<Exemplary descriptions>

- Detailed description of XX functional blocks
- Structure, functionalities, input / output interface, processing methods, common functional areas, etc

Describe specifically or in detail, the structure / functionalities of each function block and the interface between the functional blocks, among others.

6. Data Handled by the System

<Exemplary descriptions>

- Common data that are exchanged between sensors, actuators and the like (data types, volume, precision, etc)

Describe the data handled commonly by the entire system, the purpose of using such data, their read / write sources, etc

7. List of Exceptions

<Exemplary descriptions>

- List of information on abnormal / exceptional cases
- Codes and messages activated in abnormal / exceptional cases, their definitions, countermeasures, etc

Describe the codes and messages activated in abnormal / exceptional cases, their definitions and the countermeasures to be taken at respective cases in the form of a table.

8. Others

<Exemplary descriptions>

- Details on the information areas that need to be commonly controlled
- Name, structure, format, size, initial value, access restrictions of information within the common areas, among others

Describe the types of information that need to be shared by all the members of the development team through the system as a whole.

(SA104) Safety Requirements Specifications

Safety Requirements Specifications is a document used to describe the outcome of the activity to define the safety requirements, including the safety integrity levels (SIL), that the system has to achieve,

■ Sub-tasks to create this document

SAP1 Safety Requirements Definition

- 1.1 Creating the Safety Requirements Specifications
 - 1.1.4 Creating the Safety Requirements Specifications

■ Information referenced to prepare this document

Product Plan Description	(SA102) List of System Safety Requirements
Product Specifications	(SA103) System Safety Integrity Level
(SA101) List of Potential System Failures	

■ Example of items to be described

- | | |
|----------------------------------|---------------------------------------|
| 1. Overview | 4. List of System Safety Requirements |
| 2. System Structure | 5. Others |
| 3. System Safety Integrity Level | |

■ Sub-tasks that make use of this document

- | | |
|--|---|
| SAP1 Safety Requirements Definition | 1.1.1 Creating the Software Requirements Specifications |
| 1.2 Reviewing the Safety Requirements Specifications | 1.1.2 Clarifying the Functional Software Requirements |
| 1.2.1 Reviewing the Safety Requirements Specifications Internally | 1.1.3 Clarifying the Non-functional Software Requirements |
| SYP2 System Architectural Design | 1.2 Reviewing the Software Requirements Specifications |
| 2.1 Creating the System Architectural Design Description | 1.2.1 Reviewing the Software Requirements Specifications Internally |
| 2.1.1 Confirming the Design Conditions | SAP2 Safety Testing |
| 2.1.2 Designing the System Structure | 2.1 Preparing for Safety Test |
| 2.1.3 Designing the Overall System Behaviors | 2.1.1 Preparing for Safety Test |
| 2.1.4 Designing the Interface | 2.3 Reviewing the Safety Test Results |
| 2.2 Reviewing the System Architectural Design | 2.3.1 Reviewing the Safety Test Results Internally |
| 2.2.1 Reviewing the System Architectural Design Description Internally | SYP4 Reviewing the Safety Test Results Internally |
| 2.3 Jointly Reviewing the System Architectural Design | 4.4 Confirming the Completion of System Development |
| 2.3.1 Jointly Reviewing the System Architectural Design Description | 4.4.1 Confirming the Completion of System Development |
| SWP1 Software Requirements Definition | SUP1 Project Management |
| 1.1 Creating the Software Requirements Specifications | 1.4 Creating the Project Completion Report |

■ (SA104) Safety Requirements Specifications (Example)

Cover page

Name of document

Document number

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				
			⋮	

Document No.
Page No.
Date of issue / Issued by

Provide a column to enter the revised information.

Table of contents

Name of document

Table of contents

1. Overview Page No.

2. System Structure..... Page No.

3. List of Potential System Failures Page No.

4. List of System Safety Requirements Page No.

5. List of System Safety Requirements Page No.

6. Others Page No.

Document No.
Page No.
Date of issue / Issued by

Text (1)	Name of document	
<p>1. Overview</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Purpose of this document · Positioning of this document · Intended users · Scope of description, contents, etc · Referenced documents, etc · Definition (terminology, acronyms, etc) 	<p>Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.</p>	
<p>2. System Structure</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Overall system structure - Name / basic functionalities of each system component 	<p>Describe the structure of the system as a whole, the positioning of software, and the relationship / conditions surrounding the software. Describe the requirements, conditions and other relevant matters in an orderly manner by referring to related documents like the System Requirements Specifications and Hardware Specifications.</p>	
<p>3. List of Potential System Failures</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Definition on the system's normally expected operating environments and operating modes · Potential circumstances in which the system may deviate from safe operations / behaviors, and the possibility of such circumstances arising · System hazard analysis (FTA, FMEA) 	<p>Describe the potential circumstances in which the system may deviate from safe operations / behaviors, and the result of hazard analysis on each of these potentially hazardous cases.</p>	
<p>4. List of System Safety Requirements</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Operations to avoid danger by correcting the system behavior when it deviates from the safe state (definitions of automatic (system) and manual (human) corrective operations) · Requirements on the functionalities to ensure safety when the system behavior falls into an abnormal state For example: Safety functions that correspond to each operating mode of the system (start-up, automatic, manual, semi-automatic, steady, non-steady, etc) · Requirements on the mechanism to avoid system failures For example: Safety mechanisms of the system like logic duplication, fail-safe and foolproof <ul style="list-style-type: none"> · Mechanism to maintain the safety of the system by minimizing the impact of possible safety-inhibiting factors including disturbances to the system, input data errors and users' operation errors · Mechanism to ensure safety of the system while it is used or in operation (such as, how the system administrator and the users must respond when the system is in abnormal state) 	<p>Describe the requirements for ensuring the safety of the system and its peripherals (including external systems and users).</p>	
<p>5. System Safety Integrity Level</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · The level of safety (SIL) that the system is required to meet for each safety function 	<p>Describe the level of safety (SIL: Safety Integrity Level) that the system is required to meet.</p>	
<p>6. Others</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Requirements on the safety check mechanism run during development. 	<p>Describe miscellaneous matters that are worthy of special mention.</p>	
Document No.	Page No.	Date of issue / Issued by

(SW105) Software Requirements Specifications

Software Requirements Specifications is a document used to describe the functional and non-functional requirements that the software has to achieve, the constraints and various other matters that have been examined in the activity to define the software requirements.

■ Sub-tasks to create this document

SWP1 Software Requirements Definition

- 1.1 Creating the Software Requirements Specifications
- 1.1.5 Creating the Software Requirements Specifications

■ Information referenced to prepare this document

(SW101) List of Constraints

(SW102) List of Functional Software Requirements

(SW103) List of Non-functional Software

Requirements

(SW104) Prioritized List of Software Requirements

■ Example of items to be described

- | | |
|-------------------------------------|--|
| 1. Overview | 6. Functional Details |
| 2. System Structure | 7. Interface Details |
| 3. Functional Overview | 8. Detailed Non-functional Requirements on Performance, Quality, etc |
| 4. Constraints | 9. Others |
| 5. Use Cases and Use Case Scenarios | |

■ Sub-tasks that make use of this document

SWP1 Software Requirements Definition

- 1.2 Reviewing the Software Requirements Specifications
 - 1.2.1 Reviewing the Software Requirements Specifications Internally

SWP2 Software Architectural Design

- 2.1 Creating the Software Architectural Design Description
 - 2.1.1 Confirming the Design Conditions
 - 2.1.2 Designing the Software Structure
 - 2.1.3 Designing the Overall Software Behaviors
 - 2.1.4 Designing the Interface
- 2.2 Reviewing the Software Architectural Design
 - 2.2.1 Reviewing the Software Architectural Design Description Internally
- 2.3 Jointly Reviewing the Software Architectural Design
 - 2.3.1 Jointly Reviewing the Software Architectural Design Description

SWP3 Software Detailed Design

- 3.1 Creating the Functional Unit Detailed Design Description
 - 3.1.1 Dividing into Program Units

SWP6 Comprehensive Software Testing

- 6.1 Preparing for Comprehensive Software Test
 - 6.1.1 Creating the Comprehensive Software Test Specifications
 - 6.1.2 Preparing for Comprehensive Software Test
 - 6.1.3 Reviewing the Comprehensive Software Test Specifications Internally
- 6.3 Reviewing the Comprehensive Software Test Results
 - 6.3.1 Reviewing the Comprehensive Software Test Results Internally

SUP1 Project Management

- 1.4 Creating the Project Completion Report

■ (SW105) Software Requirements Specifications (Example)

Cover page

Name of document

Document No.

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				
			⋮	
			⋮	
			⋮	

Document No. Page No. Date of issue / Issued by

Provide a column to enter the revised information.

Table of Contents

Name of document

Table of Contents

1. Overview Page No.

2. System Structure Page No.

3. Functional Overview Page No.

4. Constraints Page No.

5. Use Cases and Use Case Scenarios Page No.

6. Functional Details Page No.

7. Interface Details Page No.

8. Detailed Non-functional Requirements
on Performance, Quality, etc. Page No.

9. Others Page No.

Document No. Page No. Date of issue / Issued by

1. Overview

- <Exemplary descriptions>
- Purpose of this document
 - Positioning of this document
 - Intended users
 - Scope of description, contents, etc
 - Referenced documents, etc
 - Definition (terminology, acronyms, etc)

Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.

2. System Structure

- <Exemplary descriptions>
- Overall system structure
 - Name / basic functionalities of each system component
 - Specifications on software requirements

Describe the structure of the system as a whole, the positioning of software, and the relationship / conditions surrounding the software. Describe the requirements, conditions and other relevant matters in an orderly manner by referring to related documents like the System Requirements Specifications and Hardware Specifications.

3. Functional Overview

- <Exemplary descriptions>
- List of functionalities extracted from the functionalities that the system has to realize / provide as the ones to be realized by the software, along with their overview

Provide the general description of the functionalities in the form of a list. Provide the detailed description of the functionalities separately later on. (See #6 below.)

4. Constraints

- <Exemplary descriptions>
- Hardware configuration and its constraints
 - Constraints from OS and middleware that are going to be used, among others

Describe all the constraints without leaving out any.

5. Use Cases and Use Case Scenarios

- <Exemplary descriptions>
- Use cases and use case scenarios written on the basis of situations and context in which the product is intended / assumed to be used

Take account of the interactions between the users and the system for each functional block constituting the system, and describe the flow of these interactions in chronological order.

6. Functional Details

- <Exemplary descriptions>
- Detailed description of functionalities for realizing the use cases described above in #5.

Describe each functionality in detail.

7. Interface Details

- <Exemplary descriptions>
- Interface used, such as, for operating the product in conjunction with the peripheral software, system and hardware, among others.

Describe each relevant interface in detail.

8. Detailed Non-functional Requirements on Performance, Quality, etc

- <Exemplary descriptions>
- Requirements on reliability
 - System's abnormal processing methods
 - System's recovery procedures and methods from abnormal operating mode
 - Requirements on usability
 - Required usability achieved by the software
 - Interface with the hardware portion that is used to achieve the required usability
 - Requirements on efficiency
 - System's execution performance (e.g.: processing speed, start-up time, response time, etc)
 - Resource efficiency (e.g.: memory capacity, data size)
 - Requirements on maintainability
 - Maintenance methods (such as, remote maintenance) and techniques to realize them
 - Requirements on portability
 - Dependency of software units
 - Requirements on security
 - Data encryption, user authentication, anti-virus measures, etc

Describe the matters related to performance and quality that cannot be expressed as functional matters.

9. Others

- <Exemplary descriptions>
- Requirements on security (e.g.: data encryption, user authentication, anti-virus measures, etc)
 - Interoperability (e.g.: communication protocol, etc)
 - Requirements on external interfaces (e.g.: function interface with linked software, communication protocol, user interface, etc)

Describe miscellaneous matters that are worthy of special mention.

(SW205) Software Architectural Design Description

Software Architectural Design Description is a document used to describe how the software can achieve its requirements (software structure, control method, etc) that have been examined in the activity to design the software architecture.

■ Sub-tasks to create this document

SWP2 Software Architectural Design

- 2.1 Creating the Software Architectural Design Description
 - 2.1.6 Creating the Software Architectural Design Description

■ Information referenced to prepare this document

(SW201) Software Structure Design Description
(SW202) Functional Unit Design Description
(SW203) Software Behavioral Design Description
(SW204) Software Interface Design Description

(Materials for estimating the performance)
(Materials for estimating the amount of memory used)

■ Example of items to be described

- | | |
|-----------------------|--------------------------------|
| 1. Overview | 4.2 Software Control Method |
| 2. System Structure | 4.3 Performance Estimation |
| 3. Software Structure | 5. Details on Functional Units |
| 4. Control Method | 6. Others |
| 4.1 Memory layout | |

■ Sub-tasks that make use of this document

SWP2 Software Architectural Design

- 2.2 Reviewing the Software Architectural Design
 - 2.2.1 Internally Reviewing the Software Architectural Design Description
- 2.3 Jointly Reviewing the Software Architectural Design
 - 2.3.1 Jointly Reviewing the Software Architectural Design Description

SWP3 Software Detailed Design

- 3.1 Creating the Functional Unit Detailed Design Description
 - 3.1.1 Dividing into Program Units
 - 3.1.2 Designing the Program Units
 - 3.1.3 Defining the Interface in Detail
 - 3.1.4 Estimating the Amount of Memory Used
- 3.2 Reviewing the Software Detailed Design

3.2.1 Internally Reviewing the Software Detailed Design Description

3.3 Checking the Consistency with Hardware Specifications

3.3.1 Checking the Consistency with Hardware Specifications

SWP5 Software Integration Testing

5.1 Preparing for Software Integration Test

5.1.1 Preparing for Software Integration

5.1.2 Preparing for Software Integration Test

5.2 Conducting the Software Integration Test

5.2.1 Software Integration

5.3 Reviewing the Software Integration Test Results

5.3.1 Internally Reviewing the Software Integration Test Results

■ (SW205) Software Architectural Design Description (Example)

Cover page

Name of document

Document No.

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				

⋮
⋮
⋮

Document No. Page No. Date of issue / Issued by

Provide a column to enter the revised information.

Table of Contents

Name of document

Table of Contents

1. Overview Page No.

2. System Structure Page No.

3. Software Structure Page No.

4. Control Method Page No.

 4.1 Memory Configuration / Layout Page No.

 4.2 Software Control Method Page No.

 4.3 Performance Estimation Page No.

5. Details on Functional Units Page No.

6. Data Handled by the System Page No.

7. List of Exceptions Page No.

8. Others Page No.

Document No. Page No. Date of issue / Issued by

Text (1)	Name of document	
<p>1. Overview</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Purpose of this document · Positioning of this document · Intended users · Scope of description, contents, etc · Referenced documents, etc · Definition (terminology, acronyms, etc) 	<p>Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.</p>	
<p>2. System Structure</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Overall system structure - Name / basic functionalities of each system component - Basic specifications of external elements related to the software · Main software components that constitute the system <p>For example: Name of MPU, types of I/O connectors, hardware interrupt levels and I/O responses, memory types and size, communication protocol, input/output data formats, various performance indicators, types of external media, control methods, etc</p>	<p>Describe the structure of the system as a whole, the positioning of software, and the relationship / conditions surrounding the software. Describe the requirements, conditions and other relevant matters in an orderly manner by referring to related documents like the System Requirements Specifications and Hardware Specifications.</p>	
<p>3. Software Structure</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Overall Software Structure - Name / basic functionalities / operating modes of functional units that structure the software (Operating modes include task/non-task, task priority, operation in system space/user space, resident/non-resident common functions, maintenance, etc) - Interface between software functional units <p>For example: Show the static relationship between the functional blocks (in terms of control and data flow), and if necessary, also describe the OS, memory, external storage media, and hardware mechanism.</p>	<p>Describe the structure and basis relationship of the software that is going to be developed.</p>	
<p>4. Control Method</p>	<p>Describe mainly about how the software functional units realize the requirements defined in the Software Requirements Specifications (SW105) through the entire system (hardware, operator, external storage media, memory, etc).</p>	
<p>4.1 Memory layout</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Types of ROM/RAM and segment allocated for SRAM/cache, memory area for software, location of hardware register area, among others, based on memory address space · Layout of memory area for software (code segment, data segment, segment used for OS) · Layout of memory area for software used to control common information, linkage with table/buffer, etc 	<p>Clarify the layout, types and size of memory used, based on System Architectural Design Description or hardware specifications, among other relevant documents, and describe how the memory area for software is used. (However, in case of OS and SoC with high-performance 32 bit MPU and VM functionality, either the memory area is fixed or virtual memory is used. In such case, it would be sufficient to clarify only the portions required for the software.)</p>	
<p>4.2 Software Control Method</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Recovery method: Flow of control from when the error occurred to restart · Performance realization method: Concurrent processing control to realize the required equipment performance · Language support: Method of switching from one language to another for multiple languages · Method of updating the software after shipping: Method of refreshing the software after shipping it to the field · Method of acquiring troubleshooting data: Flow from storing trace data to retrieving / using it for analysis 	<p>By using the diagram that shows the processing sequence between functional units, describe how the signals, commands and data are detected, controlled, converted, processed, notified and stored in each use case (scenario). Moreover, it is also desirable to describe important matters that are not stated explicitly in the requirements but should be shared among the developers (for example, OS operation, internal state transition, etc).</p>	
<p>4.3 Performance Estimation</p> <p><Exemplary descriptions></p> <ul style="list-style-type: none"> · Required performance / xx(ms) · Performance realization method: Conditions and processing sequence · Estimation of each processing time: Estimate of each processing element and the grounds for the calculation 	<p>Estimate the performance quantitatively.</p>	
Document No.	Page No.	Date of issue / Issued by

Note : The term “software” used here refers to both the software that is currently developed and the existing software that is already available for use.

5. Details on Functional Units

- <Exemplary descriptions>
- Details on XX functional units
 - Structure, functionalities, input/output interface, processing methods, common functional areas, etc

Describe the structure / functionalities of each functional unit and the interface used between the functional units in detail that is specific enough to create the detailed design.

As to how low-leveled the detailed design should be, describe the intended detail level by taking account of the scale of development and the number of human resources available to the project.

The description should suffice if it covers all the information that the designers need to create the detailed design.

6. Data Handled by the System

- <Exemplary descriptions>
- Common data that requires exclusive processing, etc

Describe the data handled commonly by the entire system, the purpose of using such data, their read/write sources, etc

7. List of Exceptions

- <Exemplary descriptions>
- List of information on abnormal / exceptional cases
 - Codes and messages activated in abnormal /exceptional cases, their definitions, countermeasures, etc

Describe the codes and messages activated in abnormal /exceptional cases, their definitions and the countermeasures to be taken at respective cases in the form of a table.

8. Others

- <Exemplary descriptions>
- List of errors / information on abnormalities
 - Error / abnormal codes, messages, meaning, countermeasures, etc
 - Details of the memory area for software used to control common information
 - Name, structure, type, size, default values, access restrictions, and other attributes of each information table in the memory area for common use

Describe the information on matters related to all aspects of the software that should be shared among all the members of the development group.



(SW305) Software Detailed Design Description

Software Detailed Design Description is a document used to describe the structure of program units, the details of what they process, and the interfaces between the program units that have all been examined and defined to be implementable in the course of detailed designing of the software.

■ Sub-tasks to create this document

SWP3 Software Detailed Design

- 3.1 Creating the Functional Unit Detailed Design Description
- 3.1.5 Creating the Software Detailed Design Description

■ Information referenced to prepare this document

(SW301) Program Unit Functional / Structural Design Description
(SW302) Program Unit Design Description
(SW303) Program Unit Interface Design Description

(SW304) Amount of Memory Used (Notes)
(SW309) Report on Hardware Specifications Consistency Check Result (after the indicated matters are reflected)

■ Example of items to be described

- | | |
|--|--|
| 1. Overview | 3.3 Resource Definition |
| 2. Program Unit Functional / Structural Design Description | 3.4 Hardware Control Method |
| 2.1 List of Program Units | 3.5 System Initialization |
| 2.2 Structural Diagram of Program Units | 3.6 Common Definitions |
| 3. Program Unit Design Description | 4. Program Unit Interface Design Description |
| 3.1 Detailed Processing by Program Units | 4.1 Sequence Diagram |
| 3.2 State Management | 4.2 Interface Details |
| | 5. Amount of Memory Used |

■ Sub-tasks that make use of this document

SWP3 Software Detailed Design

- 3.2 Reviewing the Software Detailed Design
 - 3.2.1 Internally Reviewing the Software Detailed Design Description
- 3.3 Checking the Consistency with Hardware Specifications
 - 3.3.1 Checking the Consistency with Hardware Specifications

SWP4 Implementation & Unit Testing

- 4.1 Preparing for Implementation and Unit Test
 - 4.1.2 Preparing for Unit Test
- 4.2 Conducting the Implementation and Unit Test
 - 4.2.1 Implementing the Program Units
- 4.3 Reviewing the Implementation and Unit Test Results
 - 4.3.1 Reviewing the Source Code
 - 4.3.2 Reviewing the Unit Test Results Internally

■ (SW305) Software Detailed Design Description (Example)

Cover page

Name of document

Document No.

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				
		:		
		:		
		:		

Document No. Page No. Date of issue / Issued by

Provide a column to enter the revised information.

Table of Contents

Name of document

Table of Contents

1. Overview Page No.

2. Program Unit Functional / Structural Design Description Page No.

 2.1 List of Program Units Page No.

 2.2 Structural Diagram of Program Units Page No.

3. Program Unit Design Description Page No.

 3.1 Detailed Processing by Program Units Page No.

 3.2 State Management Page No.

 3.3 Resource Definition Page No.

 3.4 Hardware Control Method Page No.

 3.5 System Initialization Page No.

 3.6 Common Definitions Page No.

4. Program Unit Interface Design Description Page No.

 4.1 Sequence Diagram Page No.

 4.2 Interface Details Page No.

5. Amount of Memory Used Page No.

Document No. Page No. Date of issue / Issued by

1. Overview

- <Exemplary descriptions>
- Purpose of this document
 - Positioning of this document
 - Intended users
 - Scope of description, contents, etc
 - Referenced documents, etc
 - Definition (terminology, acronyms, etc)

Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.

2. Program Unit Functional / Structural Design Description**2.1 List of Program Units**

- <Exemplary descriptions>
- Name of program unit
 - Functional Overview

Create a list showing all the program units with their names and functional overviews.

2.2 Structural Diagram of Program Units

- <Exemplary descriptions>
- Name of program unit
 - Interrelationship between program units (start method, general description of the interface, etc)

Create a diagram illustrating the structure of the program units and their interrelationship.

3. Program Unit Design Description**3.1 Detailed Processing by Program Units**

- <Exemplary descriptions>
- Name of program unit
 - Argument
 - Return value
 - Processed contents (hardware control method, timing, error processing, OS to be used, system calls, argument values of generic libraries, conditions and techniques for high-speed processing when needed, etc)
 - Constraints (processing time, interrupt, etc)
 - Remarks (on referenced design descriptions, etc)

Describe what each program unit is designed to process. Include the functionality used for analyzing defects also in the description of what each program unit is designed to process. Desirably, the description should cover all the information necessary for implementation.

3.2 State Management

<Exemplary descriptions>

State Transition Table				
	State A	State B	State C	State D
Event A				
Event B		Enter the name of		
Event C		the transitioned state.		

Describe the defined states of the software, and how the state transitions to a different state when an event occurs. (State Transition Table)

3.3 Resource Definition

- <Exemplary descriptions>
- Name of resource
 - Usage
 - Structure
 - Values and their respective meanings
 - Capacity (size)
 - Allocation / release

Describe the commonly used resources (memory, databases, etc) in details.

3.4 Hardware Control Method

<Exemplary descriptions>
 · Method of writing to / reading from the hardware
 Set values
 Address . Port No.
 Constraints (access sequence, restrictions, etc)

Describe the specific methods of controlling each hardware.

3.5 System Initialization

<Exemplary descriptions>
 · Initialization sequence
 · Initial value of the hardware
 · Vector address / interrupt
 · Initial values of the resources (memory, database, etc)
 · Initialization of OS
 · Initialization of generic libraries
 · Initialization of external devices

Describe the method of initializing the hardware and software.

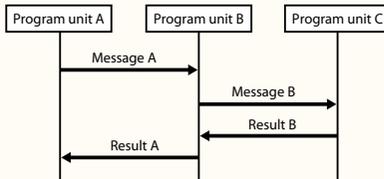
3.6 Common Definitions

<Exemplary descriptions>
 · Error values
 · Conditions for compiling
 · Resource capacity (memory, database, etc)

Describe the values that are used commonly throughout the software.

4. Program Unit Interface Design Description**4.1 Sequence Diagram**

<Exemplary descriptions>



This diagram shows the interaction between the program units in chronological order. The vertical axis stands for the course of time that moves forward from top to bottom.

4.2 Interface Details

<Exemplary descriptions>
 · Call method (values, meaning)
 · Result (values, meaning)
 · Data format (structure, values, meaning, size)
 · Constraints

Describe the interfaces in detail.

5. Amount of Memory Used

<Exemplary descriptions>
 · ROM (list of the unit names + amount of memory used by each unit)
 · RAM (list of the resource names + amount of memory used by each resource)
 · Stack area (list of the unit names + amount of memory used by each unit)

Describe the amount of memory used by each type of memory. Desirably, the description should indicate the amount of memory used respectively by each unit and resource so that the total memory usage can be easily broken down for analysis.

(SW401, SW501, SW601, SY301, SY401, SA201) Test Specifications (SW406, SW506, SW606, SY306, SY406, SA205) Test Reports

Test Specifications is a document used for describing the test cases and test data.

Test Report is a document used for describing how the test results have been evaluated. The description of the evaluation of test results normally include the information on the issues captured through the evaluation, the countermeasures to deal with these issues, the number of inquiries on uncertain matters regarding the test results, the responses to these inquiries, the number of defects that have been detected, and the judgment on whether there is a need to revise the software or not, among others.

■ Sub-tasks to create this document

SWP4 Implementation & Unit Testing

- 4.1 Preparing for Implementation and Unit Test
 - 4.1.2 Preparing for Unit Test
- 4.2 Conducting the Implementation and Unit Test
 - 4.2.3 Reviewing the Unit Test Results

SWP5 Software Integration Testing

- 5.1 Preparing for Software Integration Test
 - 5.1.2 Preparing for Software Integration Test
- 5.2 Conducting the Software Integration Test
 - 5.2.3 Reviewing the Software Integration Test Results

SWP6 Comprehensive Software Testing

- 6.1 Preparing for Comprehensive Software Test
 - 6.1.1 Creating the Comprehensive Software Test Specifications
- 6.2 Conducting the Comprehensive Software Test
 - 6.2.2 Reviewing the Comprehensive Software Test Results

SYP3 System Integration Testing

- 3.1 Preparing for System Integration Test
 - 3.1.2 Preparing for System Integration Test
- 3.2 Conducting the System Integration Test
 - 3.2.2 Reviewing the System Integration Test Results

SYP4 System Testing

- 4.1 Preparing for System Test
 - 4.1.1 Creating the System Test Specifications
- 4.2 Conducting the System Test
 - 4.2.2 Reviewing the System Test Results

SAP2 Safety Testing

- 2.1 Preparing for Safety Test
 - 2.1.1 Preparing for Safety Test
- 2.2 Conducting the Safety Test
 - 2.2.2 Reviewing the Safety Test Results

■ Information referenced to prepare this document

Unit Testing

- (SW305) Software Detailed Design Description
- (SW405) Unit Test Results (notes)
- (SU1002) Software Development Environment
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Software Integration Testing

- (SW205) Software Architectural Design Description
- (SU1002) Software Development Environment
- (SW505) Software Integration Test Results
- (SU601) Defect Management Ticket (when modifications are implemented and need to be

verified)

Comprehensive Software Testing

- (SW105) Software Requirements Specifications
- Information necessary to create the test specifications (deliverables)

See also

- Comprehensive Software Test Specifications created in the past
- (SW504) Final executable source code
- (SW605) Comprehensive Software Test Results
- (SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

System Integration Testing
(SY205) System Architectural Design Description
(SY305) System Integration Test Results
(SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

System Testing
(SY106) System Requirements Specifications
Product manuals
Information necessary to create the test specifications (deliverables)

See also
System Test Specifications created in the past

System to be tested
(SY405) System Test Results
(SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

Safety Testing
(SA104) Safety Requirements Specifications

See also
Safety Test Specifications created in the past
System to be tested

(SA204) Safety Test Results
(SU601) Defect Management Ticket (when modifications are implemented and need to be verified)

■ Example of items to be described

1. Overview
2. Test Overview
 - 2.1 Test Policy
 - 2.2 Development Phase & Test Period
 - 2.3 Scope of Test
3. Evaluation Facilities & Environment
 - 3.1 System Structure
 - 3.2 Test Environment & Facilities

4. Test Cases
5. Test Data
6. Test Procedure
7. Evaluation Result*
 - 7.1 General Description of the Test Results*
 - 7.2 Detailed Description of the Test Results*

* Items described in the Test Report

■ Sub-tasks that make use of this document

- SWP4 Implementation & Unit Testing
- 4.2 Conducting the Implementation and Unit Test
 - 4.2.2 Conducting the Unit Test
 - 4.2.3 Reviewing the Unit Test Results
 - 4.3 Reviewing the Implementation and Unit Test Results
 - 4.3.2 Reviewing the Unit Test Results Internally
- SWP5 Software Integration Testing
- 5.2 Conducting the Software Integration Test
 - 5.2.2 Conducting the Software Integration Test
 - 5.2.3 Reviewing the Software Integration Test Results
 - 5.3 Reviewing the Software Integration Test Results
 - 5.3.1 Reviewing the Software Integration Test Results Internally
- SWP6 Comprehensive Software Testing
- 6.1 Preparing for Comprehensive Software Test
 - 6.1.2 Preparing for Comprehensive Software Test
 - 6.1.3 Reviewing the Comprehensive Software Test Specifications Internally
 - 6.2 Conducting the Comprehensive Software Test

- 6.2.1 Conducting the Comprehensive Software Test
 - 6.2.2 Reviewing the Comprehensive Software Test Results
- 6.3 Reviewing the Comprehensive Software Test Results
- 6.3.1 Reviewing the Comprehensive Software Test Results Internally
- 6.4 Confirming the Completion of Software Development
- 6.4.1 Confirming the Completion of Software Development
- SYP3 System Integration Testing
- 3.2 Conducting the System Integration Test
 - 3.2.1 Conducting the System Integration Test
 - 3.2.2 Reviewing the System Integration Test Results
 - 3.3 Reviewing the System Integration Test Results
 - 3.3.1 Reviewing the System Integration Test Results Internally

SYP4 System Testing

- 4.1 Preparing for System Test
 - 4.1.2 Preparing for System Test
 - 4.1.3 Reviewing the System Test Specifications Internally
- 4.2 Conducting the System Test
 - 4.2.1 Conducting the System Test
 - 4.2.2 Reviewing the System Test Results
- 4.3 Reviewing the System Test Results
 - 4.3.1 Reviewing the System Test Results Internally
- 4.4 Confirming the Completion of System Development

- 4.4.1 Confirming the Completion of System Development

SAP2 Safety Testing

- 2.2 Conducting the Safety Test
 - 2.2.1 Conducting the Safety Test
 - 2.2.2 Reviewing the Safety Test Results
- 2.3 Reviewing the Safety Test Results
 - 2.3.1 Reviewing the Safety Test Results Internally

SUP1 Project Management

- 1.4 Creating the Project Completion Report

SA201, SA205) Test Specifications/ Test Reports

Cover page

Name of document

Document No.

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Include a column to which responsibility and confirmation of creation and approval information can be added.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				
			⋮	
			⋮	
			⋮	

⋮

⋮

Document No.
Page No.
Date of issue / Issued by

Provide a column to enter the revised information.

Table of Contents

Name of document

Table of Contents

1. Overview Page No.

2. Test Overview Page No.

 2.1 Test Policy Page No.

 2.2 Development Phase & Test Period Page No.

 2.3 Scope of Test Page No.

3. Evaluation Facilities & Environment Page No.

 3.1. System Structure Page No.

 3.2 Test Environment & Facilities Page No.

4. Overview of the Test Results Page No.

5. Attachments Page No.

 5.1 Detailed Description of Test Cases Page No.

 5.2 Bug Curve Page No.

 5.3. Others Page No.

Document No.
Page No.
Date of issue / Issued by

1. Overview

<Exemplary descriptions>

- Purpose of this document
- Positioning of this document
- Intended users
- Scope of description, contents, etc
- Referenced documents, etc
- Definition (terminology, acronyms, etc)

Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.

2. Test Overview**2.1 Test Policy**

Describe the basic concept of the test.

2.2 Development Phase & Test Period

<Exemplary descriptions>

Development phase	SW Version	HW Version	Inspection start date	Inspection end date	Remarks

Describe the software development phase in which the test is conducted, the version of the software and hardware that are tested, and the period taken to complete the test.

2.3 Scope of Test

<Exemplary descriptions>

Software Requirements Specifications	Software Architectural Design Description	Functionality to be inspected	Conditions for inspection

Describe the scope of the software test (software functionality that is tested) conducted according to the Test Specifications (Match each test case with the corresponding check points described in the Software Requirements Specifications or the Software Architectural Design Description) .

⋮
⋮

3. Evaluation Facilities & Environment

3.1. System Structure

Describe the system structure and the relationship with the test object (software) in an easy-to-understand manner by using a diagram or other means.

3.2 Test Environment & Facilities

<Exemplary descriptions>

Equipment	Manufacturer	Model No.	Serial No.	Quantity	Usage

⋮

Describe the environment (including the method of connecting the test tools) used to carry out the test (hardware environment in case of Comprehensive Software Test) in an easy-to-understand manner by using a diagram or other means.

In case of Comprehensive Software Test, also provide additional information about the test equipment (such as, their version) and the purpose of using them.

4. Overview of the Test Results

<Exemplary descriptions>

Classification ID	Classification	Number of test cases that have been tested	Inspection start date	Inspection end date	Number of test cases that are not tested yet	Number of defects	Number of defects that are not solved yet	Remarks

⋮

Describe the test contents and the expected results. Number each test case for administrative purpose, enter the test dates, and describe the test results.

5. Attachments

5.1 Detailed Description of Test Cases

<Exemplary descriptions>
(Continued to next page)

Describe the test data, expected output and results, and other relevant information regarding each test case.

5.2 Bug Curve

<Exemplary descriptions>



Show the transition in the number of bugs detected over time during the testing phase by using a graph where the vertical axis stands for the total bug count and horizontal axis stands for the course of time. <Related Information> Related Techniques (P73)

5.3 Others

<Exemplary descriptions>

- Criteria for classifying the test cases (normal value inputs, abnormal value inputs, past defects, load test, durability test, etc)
- Criteria for judging the acceptance or rejection of the test results (OK, NG, OB, No Check)
- Attachments that can be used as objective evidences of the test results (waveform data, etc)

Describe the information on software related matters that should be shared among all the members of the development group, including the test trails.

(SY107, SY206, SA105, SW106, SW206, SW306, SW408, SW507, SW607, SY307, SY404, SY407, SA206) Internal Review Report

Internal Review Report is a document used to describe the issues found while reviewing the specifications, design descriptions and reports on test results internally, the measures to address these issues, the personnel in charge of carrying out these measures, and the outcome of these actions, including the results of the tests for verifying the implemented modifications.

■ Sub-tasks to create this document

SYP1 System Requirements Definition

1.2 Reviewing the System Requirements Specifications

1.2.1 Reviewing the System Requirements Specifications Internally

SYP2 System Architectural Design

2.2 Reviewing the System Architectural Design

2.2.1 Reviewing the System Architectural Design Description Internally

SAP1 Safety Requirements Definition

1.2 Reviewing the Safety Requirements Specifications

1.2.1 Reviewing the Safety Requirements Specifications Internally

SWP1 Software Requirements Definition

1.2 Reviewing the Software Requirements Specifications

1.2.1 Reviewing the Software Requirements Specifications Internally

SWP2 Software Architectural Design

2.2 Reviewing the Software Architectural Design

2.2.1 Reviewing the Software Architectural Design Description Internally

SWP3 Software Detailed Design

3.2 Reviewing the Software Detailed Design

3.2.1 Reviewing the Software Detailed Design Description Internally

SWP4 Implementation & Unit Testing

4.3 Reviewing the Implementation and Unit Test Results

4.3.2 Reviewing the Unit Test Results Internally

SWP5 Software Integration Testing

5.3 Reviewing the Software Integration Test Results

5.3.1 Reviewing the Software Integration Test Results Internally

SWP6 Comprehensive Software Testing

6.1 Preparing for Comprehensive Software Test

6.1.3 Reviewing the Comprehensive Software Test Specifications Internally

6.3 Reviewing the Comprehensive Software Test Results

6.3.1 Reviewing the Comprehensive Software Test Results Internally

SYP3 System Integration Testing

3.3 Reviewing the System Integration Test Results

3.3.1 Reviewing the System Integration Test Results Internally

SYP4 System Testing

4.1 Preparing for System Test

4.1.3 Reviewing the System Test Specifications Internally

4.3 Reviewing the System Test Results

4.3.1 Reviewing the System Test Results Internally

SAP2 Safety Testing

2.3 Reviewing the Safety Test Results

2.3.1 Reviewing the Safety Test Results Internally

■ Information referenced to prepare this document

System Requirements Definition	(SW407) Internal Confirmation Note (on Source Code)
(SY106) System Requirements Specifications	(SU601) Defect Management Ticket
System Architectural Design	Software Integration Testing
(SY106) System Requirements Specifications	(SW501) Software Integration Test Specifications
(SA104) Safety Requirements Specifications	(SW503) Internal Confirmation Note (on Software Integration Test Specifications)
(SY205) System Architectural Design Description	(SW506) Software Integration Test Report
(SY201) System Structural Diagram (Functional Block Diagram)	(SU601) Defect Management Ticket
(SY203) System Behavioral Design Description	Comprehensive Software Testing
(SY204) System Interface Design Description	(SW601) Comprehensive Software Test Specifications
Software Requirements Definition	(SW604) Internal Confirmation Note (on Comprehensive Software Test Specifications)
Product Plan Description	(SW606) Comprehensive Software Test Report
(SY106) System Requirements Specifications	(SU601) Defect Management Ticket
(SY205) System Architectural Design Description	System Integration Testing
(SA104) Safety Requirements Specifications	(SY301) System Integration Test Specifications
(SW105) Software Requirements Specifications	(SY303) Internal Confirmation Note (on System Integration Test Specifications)
Software Architectural Design	(SY306) System Integration Test Report
(SW105) Software Requirements Specifications	(SU601) Defect Management Ticket
(SW205) Software Architectural Design Description	System Testing
(SW201) Software Structure Design Description	(SY401) System Test Specifications
(SW202) Functional Unit Design Description	(SY404) Internal Confirmation Notes (on System Test Specifications)
(SW203) Software Behavioral Design Description	(SY406) System Test Report
(SW204) Software Interface Design Description	(SU601) Defect Management Ticket
Software Detailed Design	Safety Testing
(SW205) Software Architectural Design Description	(SA201) Safety Test Specifications
(SW305) Software Detailed Design Description	(SA203) Internal Confirmation Note (on Safety Test Specifications)
Implementation & Unit Testing	(SA205) Safety Test Report
(SW401) Unit Test Specifications	(SU601) Defect Management Ticket
(SW403) Internal Confirmation Note (on Unit Test Specifications)	
(SW406) Unit Test Report	

■ Example of items to be described

- ▶ Project name
- ▶ Object to be reviewed
- ▶ Date & time; Location
- ▶ Attendees
- ▶ Reviewed documents
- ▶ Issues; Countermeasures; Person in charge; Due date; Modification verified

■ Sub-tasks that make use of this document

SYP2 System Architectural Design

2.3 Jointly Reviewing the System Architectural Design

2.3.1 Jointly Reviewing the System Architectural Design Description

SWP2 Software Architectural Design

2.3 Jointly Reviewing the Software Architectural Design

2.3.1 Jointly Reviewing the Software Architectural Design Description

SWP6 Comprehensive Software Testing

6.4 Confirming the Completion of Software Development

6.4.1 Confirming the Completion of Software Development

SYP4 System Testing

4.4 Confirming the Completion of System Development

4.4.1 Confirming the Completion of System Development

SUP1 Project Management

1.4 Creating the Project Completion Report

■ (SY107, SY206, SA105, SW106, SW206, SW306, SW408, SW507, SW607, SY307, SY404, SY407, SA206) Internal Review Report

Internal Review Report					
Project name			Management No.		
Object to be reviewed			Organization		
Date & time	/ / () : - :		Approved by	Checked by	Created by
Location					
Attendees			/ /	/ /	/ /
Reviewed documents					
No	Issue	Countermeasures	Person in charge Due date	Modification verified	
			/ /	/ /	

Enter the project name, date & time, location, attendees and other administrative information relevant to the internal review.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of the documents reviewed internally.

Describe the issues raised in the internal review, the countermeasures that can be possible solutions, personnel in charge of executing the countermeasures (including modifications), due date by when the corrective actions need to be completed, and whether the modification has been verified as successful in solving the issue or not.

Related Standards

ISO/IEC 15288 Systems engineering - System life cycle processes (JIS X 0170)

Year of establishment / revision :	ISO/IEC 2002, JIS 2004
Purpose :	Standardization of the concepts of (IT) system life cycle processes
Scope :	(IT) System development process
URL :	http://www.iso.org/
Source in Japan :	Japanese Standards Association

Overview:

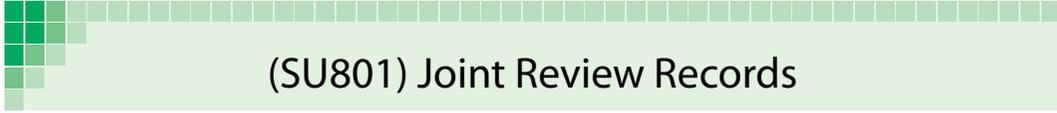
Standard that defines the processes to develop a system with multiple software and hardware that work together. It is positioned as the standard that extended the scope of "ISO/IEC 12207 Software life cycle processes" to the system level.

In this standard, the processes ranging from the initial planning stage to the final retirement stage when the users stop using the system are sorted out orderly according to the life cycle of the system, in addition to grouping the processes and defining the terminology related to respective processes.

Moreover, in this standard, the processes are described separately by "purpose(s)", "outcome" and "actions necessary to fulfill the defined purpose(s)".

Process description

- Process name
- Purpose of the process
- Outcome of the process
- Actions necessary to fulfill the purpose(s).



(SU801) Joint Review Records

Joint Review Records is a document used to describe the matters indicated in the joint reviews, modifications to resolve the indicated matters, personnel in charge of implementing the modifications, and the result of verification of the modifications, among others.

■ Sub-tasks to create this document

- SYP2 System Architectural Design
 - 2.3 Jointly Reviewing the System Architectural Design
 - 2.3.1 Jointly Reviewing the System Architectural Design Description
- SWP2 Software Architectural Design
 - 2.3 Jointly Reviewing the Software Architectural Design
 - 2.3.1 Jointly Reviewing the Software Architectural Design Description
- SWP6 Comprehensive Software Testing
 - 6.4 Confirming the Completion of Software Development
 - 6.4.1 Confirming the Completion of Software Development
- SYP4 System Testing
 - 4.4 Confirming the Completion of System Development
 - 4.4.1 Confirming the Completion of System Development
- SUP8 Joint Review
 - 8.2 Carrying Out the Review
 - 8.3 Acknowledging and Following Up on Matters That Have Been Reviewed

■ Information referenced to prepare this document

- | | |
|---|---|
| System Architectural Design | (SW606) Comprehensive Software Test Report |
| (SY205) System Architectural Design Description | (SW607) Internal Review Report (on Comprehensive Software Test) |
| (SY106) System Requirements Specifications | (SU101) Project Plan Description |
| (SA104) Safety Requirements Specifications | (SU601) Defect Management Ticket |
| Software Architectural Design | System Testing |
| (SW205) Software Architectural Design Description | (SY106) System Requirements Specifications |
| (SW206) Internal Review Report (on Software Architectural Design) | (SY401) System Test Specifications |
| | (SY406) System Test Report |
| See also | (SY407) Internal Review Report (on System Test) |
| (SY106) System Requirements Specifications | (SA104) Safety Requirements Specifications |
| (SY205) System Architectural Design Description | (SA201) Safety Test Specifications |
| (SW105) Software Requirements Specifications | (SA206) Internal Review Report (on Safety Test) |
| Comprehensive Software Testing | (SU101) Project Plan Description |
| (SW106) Software Requirements Specifications | (SU601) Defect Management Ticket |
| (SW601) Comprehensive Software Test Specifications | |

■ Example of items to be described

- ▶ Project name
- ▶ Object to be reviewed
- ▶ Date & time; Location
- ▶ Attendees
- ▶ Reviewed documents
- ▶ Issues; Countermeasures; Person in charge; Due date; Modification verified

■ Sub-tasks that make use of this document

SUP1 Project Management

1.4 Creating the Project Completion Report

SUP8 Joint Review

8.3 Acknowledging and Following Up on
Matters That Have Been Reviewed

■ (SU801) Joint Review Records (Example)

Joint Review Records						
Project name			Management No.			
Object to be reviewed			Organization			
Date & time / / () : - :			Approved by		Checked by	
Location					Created by	
Attendees	Review manager					
	Reviewers					
	Reviewees		/ /		/ /	
Reviewed documents						
Review outcomes			<input type="checkbox"/> Review closed <input type="checkbox"/> Re-review			
Summary						
No	Issue	Countermeasure	Person in charge Due date	Modification verified		
			/ /	/ /		
Confirmation of the result of implemented corrective actions		Review manager's comments			Issue closed confirmed	
					/ /	

Describe the project name, date & time, location, attendees, and administrative information.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of the documents that are reviewed.

Summarize the outcome of the review, and also indicate whether the review can be closed or there is a need to review again.

Describe the issue(s) raised in the joint review, the countermeasures that can be possible solutions, personnel in charge of executing the countermeasures (including modifications), due date by when the corrective actions need to be completed, and whether the modification has been verified as successful in solving the issue or not.

Enter the review manager's comments.



(SU601) Defect Management Ticket

Defect Management Ticket is a form used to record the defective condition detected in a test and the result of the investigation of its cause(s) and impact analysis, as well as to describe the countermeasures and/or policy of corrective actions to be taken to resolve the defect.

Each Defect Management Ticket is managed by Defect Management Register (SU602).

■ Sub-tasks to create this document

SWP4 Implementation & Unit Testing

4.2 Conducting the Implementation and Unit Test

4.2.3 Reviewing the Unit Test Results

SWP5 Software Integration Testing

5.2 Conducting the Software Integration Test

5.2.3 Reviewing the Software Integration Test Results

SWP6 Comprehensive Software Testing

6.2 Conducting the Comprehensive Software Test

6.2.2 Reviewing the Comprehensive Software Test Results

SAP2 Safety Testing

2.2 Conducting the Safety Test

2.2.2 Reviewing the Safety Test Results

SYP3 System Integration Testing

3.2 Conducting the System Integration Test

3.2.2 Reviewing the System Integration Test Results

SYP4 System Testing

4.2 Conducting the System Test

4.2.2 Reviewing the System Test Results

SUP6 Problem Resolution Management

6.1 Recording the Problems and Analyzing the Causes

6.2 Analyzing the Impact and Devising the Acceptable Solution

6.3 Implementing the Acceptable Solution

6.4 Tracking the Implemented Solution

■ Information referenced to prepare this document

Unit Testing

(SW401) Unit Test Specifications

(SW405) Unit Test Results

Software Integration Testing

(SW501) Software Integration Test Specifications

(SW505) Software Integration Test Results

Comprehensive Software Testing

(SW601) Comprehensive Software Test Specifications

(SW605) Comprehensive Software Test Results

Safety Testing

(SA201) Safety Test Specifications

(SA204) Safety Test Results

System Integration Test

(SY301) System Integration Test Specifications

(SY305) System Integration Test Results

System Testing

(SY401) System Test Specifications

(SY405) System Test Results

■ Example of items to be described

- ▶ Project name
- ▶ Management No.
- ▶ Circumstances (When (date & time), where (location) and in which phase the defect was detected, phenomenon, etc)
- ▶ Cause(s) / Impact
- ▶ Description of the treatment
- ▶ Response policy (Priority level, due date, etc)

■ Sub-tasks that make use of this document

SWP4 Implementation & Unit Testing

- 4.1 Preparing for Implementation and Unit Test
 - 4.1.2 Preparing for Unit Test
- 4.2 Conducting the Implementation and Unit Test
 - 4.2.1 Implementing the Program Units
 - 4.2.2 Conducting the Unit Test
 - 4.2.3 Reviewing the Unit Test Results
- 4.3 Reviewing the Implementation and Unit Test Results
 - 4.3.2 Reviewing the Unit Test Results Internally

SWP5 Software Integration Testing

- 5.1 Preparing for Software Integration Test
 - 5.1.2 Preparing for Software Integration Test
- 5.2 Conducting the Software Integration Test
 - 5.2.2 Conducting the Software Integration Test
 - 5.2.3 Reviewing the Software Integration Test Results
- 5.3 Reviewing the Software Integration Test Results
 - 5.3.1 Reviewing the Software Integration Test Results Internally

SWP6 Comprehensive Software Testing

- 6.1 Preparing for Comprehensive Software Test
 - 6.1.2 Preparing for Comprehensive Software Test
- 6.2 Conducting the Comprehensive Software Test
 - 6.2.2 Reviewing the Comprehensive Software Test Results
- 6.3 Reviewing the Comprehensive Software Test Results
 - 6.3.1 Reviewing the Comprehensive Software Test Results Internally
- 6.4 Confirming the Completion of Software Development

6.4.1 Confirming the Completion of Software Development

SAP2 Safety Testing

- 2.1 Preparing for Safety Test
 - 2.1.1 Preparing for Safety Test
- 2.2 Conducting the Safety Test
 - 2.2.1 Conducting the Safety Test
 - 2.2.2 Reviewing the Safety Test Results
- 2.3 Reviewing the Safety Test Results
 - 2.3.1 Reviewing the Safety Test Results Internally

SYP3 System Integration Testing

- 3.1 Preparing for System Integration Test
 - 3.1.2 Preparing for System Integration Test
- 3.2 Conducting the System Integration Test
 - 3.2.1 Conducting the System Integration Test
 - 3.2.2 Reviewing the System Integration Test Results
- 3.3 Reviewing the System Integration Test Results
 - 3.3.1 Reviewing the System Integration Test Results Internally

SYP4 System Testing

- 4.1 Preparing for System Test
 - 4.1.2 Preparing for System Test
- 4.2 Conducting the System Test
 - 4.2.2 Reviewing the System Test Results
- 4.3 Reviewing the System Test Results
 - 4.3.1 Reviewing the System Test Results Internally
- 4.4 Confirming the Completion of System Development
 - 4.4.1 Confirming the Completion of System Development

SUP1 Project Management

1.4 Creating the Project Completion Report

SUP6 Problem Resolution Management

6.1 Recording the Problems and Analyzing the
Causes

6.2 Analyzing the Impact and Devising the
Acceptable Solution

6.3 Implementing the Acceptable Solution

6.4 Tracking the Implemented Solution

■ (SU601) Defect Management Ticket (Example)

Defect Management Ticket					
Project name			Management No.		
Circumstances	Detection date / / ,		Detector		Function name
	Title				Test case ID
	[Describe the contents of the defect / procedure to reproduce the defect / incidence rate, etc]				Software version
					Development phase when the defect was detected
			Attachment: <input type="checkbox"/> Yes <input type="checkbox"/> No		
Cause(s)/Impact	Date of investigation		Team	Person in charge	Main cause [For classification of the cause]
	Cause(s)				
	Impact [Extent of impact / (Also describe if necessary) the impact when no corrective actions were taken]				Development phase when the defect was implanted
Attachment: <input type="checkbox"/> Yes <input type="checkbox"/> No					
Description of the treatment	Treatment fix date		Team	Person in charge	Estimate man-hours
	Date of completion		Team	Person in charge	
Attachment: <input type="checkbox"/> Yes <input type="checkbox"/> No					
Response policy	Action / no action	Priority level	Due date	Remarks	
*Confirmation columns					
Treatment approved		Verification completed			
/ /		/ /			

Enter the project name and administrative information.

Enter the date and time, and the circumstances when the defect was detected.

Also describe where and when the defect was detected.

Describe the cause (s) and the extent of impact of the defect.

Also describe when the defect was implanted.

Describe how the defect is treated, who / which team is in charge of implementing the treatment, and the estimate man-hours required to complete the treatment.

Describe the response policy (whether to take the corrective action or not), the priority of implementing the action, and the due date when the corrective action must be completed by.

Prepare columns to enter the status of defect management (verification and approval of the treatment).

Related Standards

ISO 9241 Ergonomic requirements for office work with visual display terminals (VDTs)

Part 10: Dialogue principles (JIS Z 8520)

Part 11: Guidance on usability (JIS Z 8521)

Year of establishment / revision :	ISO 1996 (9241-10), 1998 (9241-11), JIS 1999 (JIS Z 8520, 8521)
Purpose :	Defining the levels of user satisfaction of office systems that make use of visual display terminals (VDTs)
Scope :	Organizations and institutions that design, manufacture and/or evaluate visual display terminals (VDTs) used in office environment
URL :	http://www.iso.org/
Source in Japan :	Japanese Standards Association

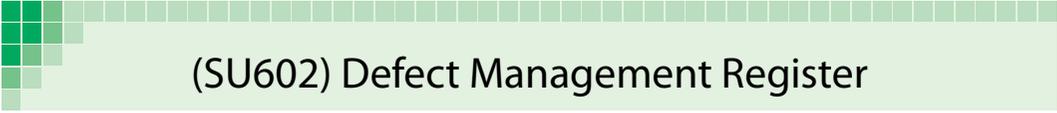
Overview:

This standard sorts out the matters pertaining to office work using visual display terminals (VDTs) from the standpoint of ergonomics. ISO 9241 is comprised of seventeen parts in all, of which Part 10 and Part 11 are the two sections of this standard that deal with usability.

“ISO 9241-10: Dialogue principles” provides the perspectives for examining the desirable forms of interaction between users and computers that can be applied when designing or evaluating software. “ISO 9241-11: Guidance on usability” defines usability in more concrete terms, and provides a set of information that should be taken into account when establishing or assessing the levels of usability. Other international standards that discuss about usability in the context of software quality include ISO 9126.

Usability :

In general, this is a term used to express the level of ease in using software or hardware. In Japanese, there are various alternative expressions that refer to the concept of usability, including “user-friendliness”, “easiness to use”, “availability”, “applicability”, and “quality use”. In the context of embedded system, the growing trend is to position usability as one of the characteristics of quality perceived by the users.



(SU602) Defect Management Register

Defect Management Register is a document that displays a table listing all the reported defects along with related information excerpted from the Defect Management Ticket (SU601) prepared for each. This document is used as the basis for managing defects.

■ Sub-tasks to create this document

SUP6 Problem Resolution Management

- 6.1 Recording the Problems and Analyzing the Causes
- 6.2 Analyzing the Impact and Devising the Acceptable Solution
- 6.3 Implementing the Acceptable Solution
- 6.4 Tracking the Implemented Solution

■ Information referenced to prepare this document

(SU601) Defect Management Ticket

■ Example of items to be described

The table consists of the following information related to defects excerpted from the Defect Management Ticket (SU601).

- ▶ Project name
- ▶ Management No.
- ▶ Circumstances
- ▶ Date when the investigation of the cause(s) and impact have been completed
- ▶ Treatment
- ▶ Date when verification was completed

■ Sub-tasks that make use of this document

SUP6 Problem Resolution Management

- 6.2 Analyzing the Impact and Devising the Acceptable Solution
- 6.3 Implementing the Acceptable Solution
- 6.4 Tracking the Implemented Solution



(SU103, SU104) Project Completion Report

Project Completion Report is a document used to report the final status and outcome of system development based on the information gained from System Test Report and Joint Review Report, provide quality-related information that will be used as the basis for determining whether to head on to product release or not, and describe the points that could be improved in the next development project.

■ Sub-tasks to create this document

SUP1 Project Management

1.4 Creating the Project Completion Report

■ Information referenced to prepare this document

(SW105) Software Requirements Specifications
(SW606) Comprehensive Software Test Report
(SU801) Joint Review Records (on Software Integration Test)
(SY106) System Requirements Specifications
(SY406) System Test Report
(SU801) Joint Review Records (on System Test)

(SA104) Safety Requirements Specifications
(SA201) Safety Test Specifications
(SA206) Internal Review Report (on Safety Test)
(SU601) Defect Management Ticket
(SU101) Project Plan Description
(SU102) Project Status Report

■ Example of items to be described

- | | |
|--|--|
| 1. Overview | 3.3 Actual Quality |
| 2. Project Overview | 3.4 Actual Productivity |
| 2.1 Project Profile | 3.5 Difference Between the Estimated and Actual Effort |
| 2.2 System Structure | 4. Constraints in Operating / Using the System |
| 3. Analysis and Feedback | 5. Orderly Disposition of Issues |
| 3.1 Engineering on the Overall | 6. Attachments |
| 3.2 Planned vs. Actual Distribution of Effort and Duration | |

■ Sub-tasks that make use of this document

(Product inspection, similar next development projects, etc)

■ (SU103, SU104) Project Completion Report (Example)

Cover page

Name of document

Document No.

Approved by	Created by
Name of approver	Name of creator
Date of approval	Date of creation

Date of issue
Issued by

Enter the name of document.

Enter the information that makes the document identifiable.

Provide a column to enter who is responsible of creating / approving the document, and when it was created / approved.

Enter the name of organization that issued the document and the date of issue.

Revision history

Name of document

Revision History

Item No.	Date	Version	Revised contents	Remarks
1				
2				
3				
4				
		⋮		
		⋮		
		⋮		

⋮

⋮

⋮

Document No.
Page No.
Date of issue / Issued by

Provide a column to enter the revised information.

Table of Contents

Name of document

Table of contents

1. Overview Page No.

2. Project Overview Page No.

 2.1 Project Profile Page No.

 2.2. System Structure..... Page No.

3. Analysis and Feedback..... Page No.

 3.1 Engineering on the Overall Page No.

 3.2 Planned vs. Actual Distribution
 of Effort and Duration Page No.

 3.3 Actual Quality Page No.

 3.4 Actual Productivity Page No.

 3.5 Difference Between the Estimated
 and Actual Effort Page No.

4. Constraints in Operating / Using the System Page No.

5. Orderly Disposition of Issues Page No.

6. Attachments Page No.

Document No.
Page No.
Date of issue / Issued by

1. Overview

- <Exemplary descriptions>
- Purpose of this document
 - Positioning of this document
 - Intended users
 - Scope of description, contents, etc
 - Referenced documents, etc
 - Definition (terminology, acronyms, etc)

Provide the general description of this document (its purpose, positioning, contents, etc) and the names of referenced documents, among others.

2. Project Overview**2.1 Project Profile**

- <Exemplary descriptions>
- Duration, actual effort, constraints, etc

Describe the basic project matters.

2.2. System Structure

- <Exemplary descriptions>
- Overall system structure
 - Name / basic functionalities of each system component
 - Operating environment of the system and its external environment

Describe the structure of the entire system including both the hardware and software.

3. Analysis and Feedback

3.1 Engineering on the Overall

<Exemplary descriptions>

- Measures to realize the product vision and embody the concept
- Measures to meet the system requirements
- Measures to deal with the constraints in realizing the product (delivery due date, quality, costs, reuse, knowledge / experience of development members, etc).

3.2 Planned vs. Actual Distribution of Effort and Duration

<Exemplary descriptions>

- Distribution of effort per phase (planned)
- Distribution of effort per phase (actual)
- Evaluation of the distribution of effort
- Future issues

3.3 Actual Quality

<Exemplary descriptions>

- Targeted level of quality (quantitative indicators)
- Actual level of quality (quantitative indicators)
- Mechanism of assuring quality
- Evaluation on quality
- Future issues

<Examples of quantitative indicators to measure quality against the targeted levels>

- Functionality, reliability, usability, efficiency, maintainability, portability
- Defect detection rate and review rate in requirement definition, designing and implementation phases
- Test density, failure detection rate, and failure convergence rate in test phase

3.4 Actual Productivity

<Exemplary descriptions>

- Targeted level of productivity
- Actual level of productivity
- Evaluation on productivity
- Future issues

3.5 Difference Between the Estimated and Actual Effort

<Exemplary descriptions>

- Difference between the estimated and actual effort
- Difference between the estimated and actual costs
- Evaluation of the accuracy between the estimated and actual results
- Future issues

Provide a concise description of the inventive approaches and measures that have been taken to realize the product vision, embody the concept and deal with constraints, as well as the effectiveness and the level of achievement of these measures.

Compare the effort actually taken to complete the activities with the planned effort, and enter the result of evaluation of the current level of achievement.

If the difference between the actual and planned effort is large, or if the actual effort of the current project differs greatly from the result of similar projects in the past, analyze what led to this big difference and organize the findings as issues to be addressed in the future.

Compare the actual level of quality achieved with the targeted level of quality, and enter the result of evaluation of the current level of achievement.

If the difference between the actual and targeted level of quality is large, analyze what led to this big difference and organize the findings as issues to be addressed in the future.

Compare the actual level of productivity achieved with the targeted level of productivity, and enter the result of evaluation of the current level of achievement.

If the difference between the actual and targeted level of productivity is large, or if the actual productivity of the current project differs greatly from the result of similar projects in the past, analyze what led to this big difference and organize the findings as issues to be addressed in the future.

Compare the estimated effort and costs with the actual effort and costs, and enter the result of evaluation of the current level of achievement.

Analyze what led to the difference between the estimated and actual results, and organize the findings as issues to be addressed in the future.

4. Constraints in Operating / Using the System

Describe the constraints in operating or using the system that became necessary as a result of treatment of the defects found in the system.

5. Orderly Disposition of Issues

Sort out the issues in development process and techniques that have been identified in the course of development, and describe them as points to be improved in the next development project.

6. Attachments

- <Exemplary descriptions>
- List of deliverables

Sort out the issues in development process and techniques that have been identified in the course of development, and describe them as points to be improved in the next development project.

Part 3

Practical Section

3.1 Procedure for Practical Use	218
3.2 Tailoring the Development Process for the Organization / Department	221
3.3 Designing the Process Phases of the Development Project	222
3.4 Planning the Development Project Management (Designing the development process phases in detail)	227

3.1 Procedure for Practical Use

Tailoring the Process

By using the process definition provided in Part 2 of this guidebook, the development process can be easily tailored to best suit the characteristics of each organization or project.

The development process defined in Part 2 of this guidebook consists of processes, activities, tasks and sub-tasks that are considered to be the standard types of work required in embedded software development. However actually, in order to move the product development forward with high efficiency in the real world, there is a need to make necessary arrangements to build a most suitable development process by taking account of the distinctive features of the targeted product itself as well the characteristics of the organization or department in charge of development.

Described here in Part 3 are various practical methods of utilizing the development process defined in this guidebook.

The practical methods described below are general examples intended to provide the readers of this guidebook a set of useful references to help customize their own development process.

Utilizing the Development Process

The development process described in Part 2 of this guidebook is defined as the standard development process that consists of various clusters of work deemed necessary to proceed with the development.

It is, however, easily foreseeable that the required set of work for actual product development or the level of importance placed on each work may vary from the process definition provided in this guidebook, depending on the specific characteristics of the targeted product.

Therefore, in order to effectively utilize the standard development process defined in this guidebook for the actual product development, the following three types of work have to be carried out (See Fig. 3.1):

- (1) Tailoring the Development Process for the Organization / Department;

(2) Designing the Process Phase of the Development Project;

(3) Planning the Development Project Management

(1) Tailoring the Development Process for the Organization / Department

The development process defined in this guidebook should, in practice, be tailored or tuned to meet the characteristics of the targeted product and/or the organization in charge of development.

(2) Designing the Process Phases of the Development Project

In actual product development, there are constraints imposed on each development project, such as, on the delivery deadline, quality and costs, that must be taken into account.

Based on the development process tailored according to product and organizational characteristics, the description of required activities (that may be tasks or sub-tasks in some cases) and the weight of each activity must be all examined. After considering these points, there is a need to place these activities along the actual timeline of the overall development project. Upon completion of this work placement process, referred here to as “process phase designing”, the activities placed on the project timeline will form the project’s “development process phases”.

(3) Planning the Development Project Management (Designing the Development Process Phases in Detail)

Then comes the decision-making process to determine specifically how, when and by whom the various types of work grouped in development process phases should be carried out. Questions, such as, when to kick off the development, when the development must be completed, which department, section or team within the organization in charge of development is responsible of specific activities, tasks or sub-tasks, and who (which member or members belonging to these departments, sections or teams) are assigned to each of these development-related jobs need to be answered and clarified at this stage. Through this clarification process, the development process phases are designed in detail, and the decisions made in this process will serve as the basis for project scheduling and resource allocation.

The process of designing the development process phases in detail is referred to as “project management planning” and the outcome of this project management planning process is called the “development project plan”.

Please note that the description on project management planning provided here is limited to general information, since it is categorized as a part of project management, which is not the main subject matter of this guidebook.

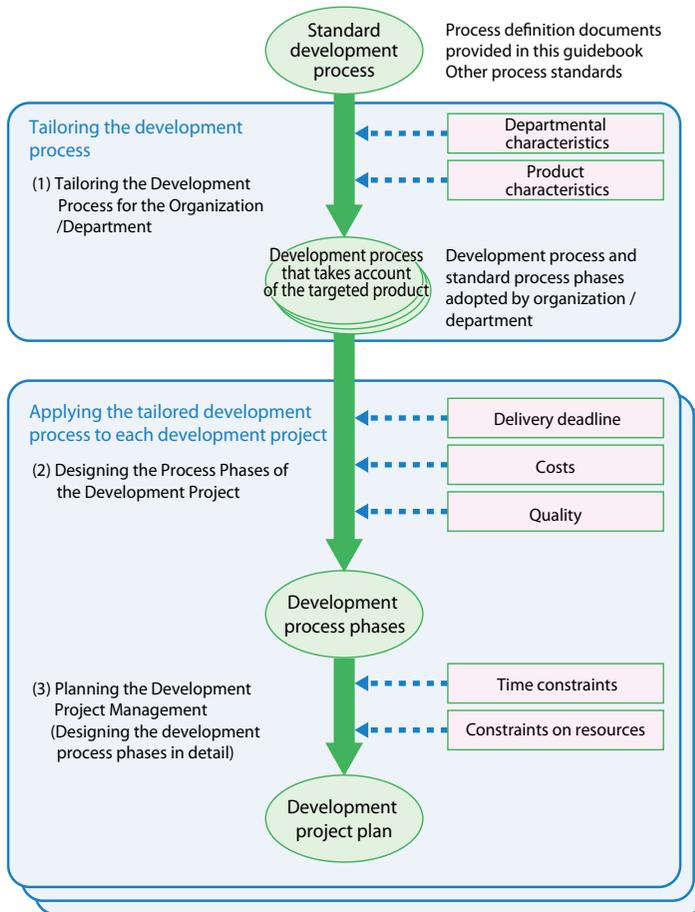


Figure 3.1 Example of How to Utilize the Development Process

3.2 Tailoring the Development Process for the Organization / Department

Tailoring the Development Process According to the Characteristics of the Organization / Department

As earlier mentioned, the development process described in Part 2 of this guidebook is defined as the standard development process consisting of an organized set of work deemed necessary to

proceed with the development.

This standard development process, however, often cannot be adopted as it is, since the targeted products to which the software are planned to be embedded come in wide variety, requiring the development process to be tailored according to the characteristics of the organization / department in charge of development and the targeted product.

Tailoring the Development Process

Tailoring of the development process includes the renaming of the general names given to processes, activities, tasks, sub-tasks and deliverables in Part 2 of this guidebook to the names that are actually used in or familiar to the organization / department(s) in charge of development, and adjusting and/or adding extra steps / items to sub-task procedures and precautions according to the characteristics of the targeted product.

Preparing the Organization / Department-specific Standard Process Phases

Organization- or department-specific standard development process phases (shortened to “standard process phases” hereafter) have to be prepared, so that the development process becomes applicable and usable in actual development projects.

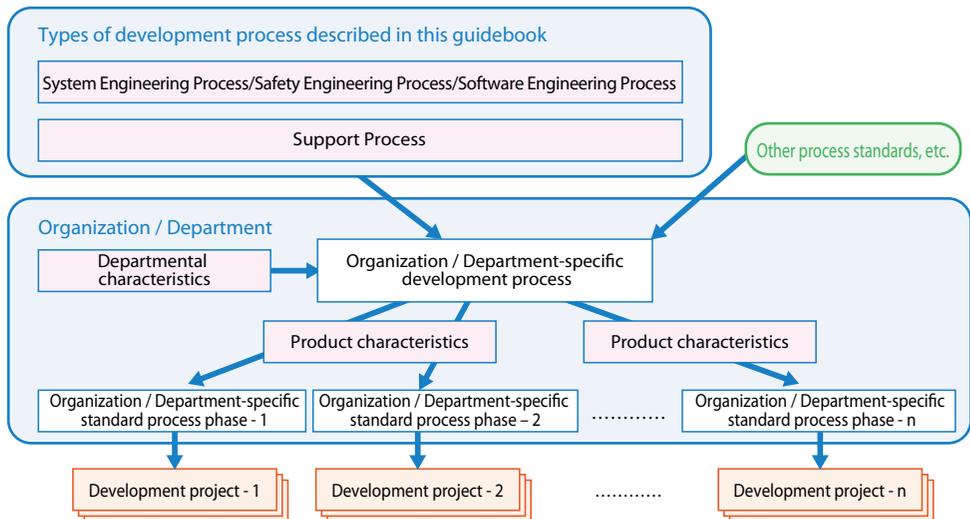


Figure 3.2 Tailoring the Development Process for the Organization / Department

3.3 Designing the Process Phases of the Development Project

■ Designing the Process Phases of Embedded Software Development Project

The objectives of designing the process phases of the development project are to examine and decide which work is executed at which level of the organization, by taking account of the product requirements (both functional and non-functional) and the structural levels of the organization (department and project team levels) in charge of development, and identifying the different types of work required for development.

Points That Should Be Considered

In designing the process phases of software development project, the following matters should be taken into consideration, especially when the project is to develop embedded software:

- (1) Be aware of the development of hardware and gain a clear understanding of its relationship with the software development project. Also, clarify the interfaces between hardware and software development areas;
- (2) Clarify the inputs, procedures and outputs (IPO) of each process (at various work levels);
- (3) Clarify the conditions for starting / ending each process (at various work levels);
- (4) Clarify the deliverables of each process (at various work levels);
- (5) Clarify the dependency relationship of information referenced in each process (at various work levels);
- (6) Group the various types of work into units that can be handled;
- (7) Gain a clear understanding on how milestones for checking the progress of the development should be set.

How to Use This Guidebook for Process Phase Designing

Process phase designing can be facilitated by using the process definition documents provided in this guidebook and following the steps described below:

- Step 1: Grasp the characteristics of the software to be developed and the organization in charge of its development.
- Step 2: Gain a rough idea of the necessity and importance of the processes and activities included in the standard development process.

Step 3: By referring to the relevant information provided in Part 2 of this guidebook under “2.2 Process Definition Documents”, examine the procedure for more specific (lower-leveled) types of development work (tasks and sub-tasks), including how much weight (focus) should be given to each work.

Step 4 : Examine what kind of deliverables should be expected as the outcome of each development effort. If necessary, also use the information described in Part 2 of this guidebook under “2.3 Document Template Samples” as your reference.

This guidebook has defined two engineering processes, System Engineering Process (SYP) and Software Engineering Process (SWP) that are directly related to development, and two other processes, Support Process (SUP) and Safety Engineering Process (SAP), that support the engineering processes. In selecting the particular types of work required in actual development, there is a need to clarify which work should be carried out in which of these processes, and also examine the combination that best works for the given development scheme.

Examining the Overall Process Phase Design

One of the actions performed in process phase designing is the examination and formulation of the overall development schedule for carrying out the activities required for the development, based on the organization / department-specific development process. In this guidebook, the act of creating the overall development schedule is called “overall process phase designing”.

The purpose of overall process phase designing is to clarify the project milestones and the interfaces between the activities, and the output (deliverables) resulting from these activities (See Fig. 3.4). These clarifications as well as the sharing of information on the formulated overall process phase design among all the members of the development project team are important in proceeding with the development as planned “on schedule”.

This guidebook assumes that, in practice, the organization or department in charge of actual development will go through the process of selecting, revising, reshuffling and/or recursively iterating the processes, activities, tasks and sub-tasks included in the development process tailored according to organizational / departmental characteristics and the organization / department-specific standard phases to meet the individual development needs.

Matters That Should Be Clarified

Through overall process phase designing, the following matters should be clarified:

- (1) Development process phases of the development project
- (2) Output (deliverables)
- (3) External interfaces (for coordination and correspondence with external organizations)

Development Process Phases of Development Project

Development process phase refers to a group of activities lined in chronological order (and allocated on a timeline). Select the activities to be included in the development process phase and clarify the order to carry them out, by taking account of the development model that is applied (waterfall development model, spiral development model, etc), the scope of work (the range of activities responsible of), whether the project is a new development or additional development of an existing product, and any other points of consideration (See Fig. 3.3).

(1) In case of small-scaled development project



(2) In case of large-scaled development project where multiple functionalities are developed concurrently

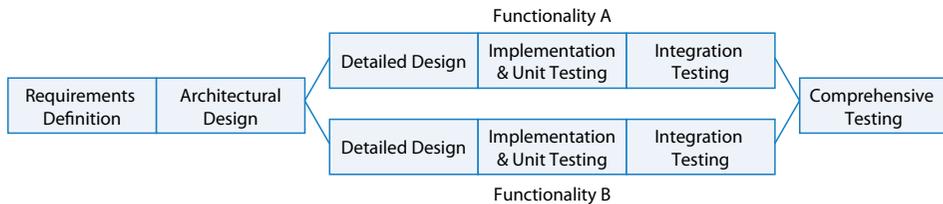


Figure 3.3 Example of Development Process Phases

Output (Deliverables)

Output refers to main deliverables produced in each activity.

Clarify the output (deliverables) by taking account of the timeframe when deliverables from external sources can be received, and the inputs and outputs between activities.

External Interfaces (for Coordination and Correspondence with External Organizations)

External interfaces refer to inputs from external sources, outputs to external destinations and collaboration with external partners, among others.

Clarify the development milestones by taking account of concurrent development of hardware, among others.

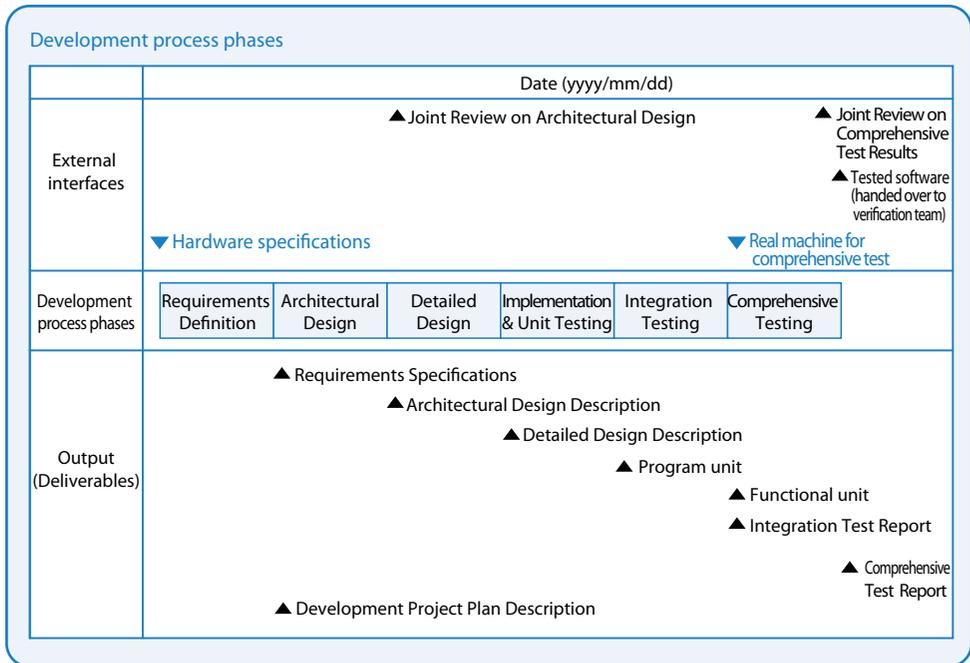


Figure 3.4 Example of Overall Process Phase Planning

Case Examples of Development Process Phases

Described below are some case examples of development process phases:

Case Example That Places Testing as High Priority (Conducting Tests in Advance)

In development projects that place importance on the development of user interfaces, the task to create the test specifications is often scheduled to be carried out at an early stage (normally during the requirements definition phase and architectural design phase). This schedule (see Fig. 3.5 below) reflects the consideration that many issues in the user interface (points that need to be improved) are normally found through trial use (i.e.: through testing).



Figure 3.5 Example of Development Process Phases (Testing as High Priority)

Case Example of Concurrent Development

In software development projects where new hardware has to be developed concurrently, more importance is placed on holding joint review meetings between software and hardware developers in the course of development, updating and sharing the latest hardware specifications with software developers without delay, and assuring the quality of the real machine used for testing. In such concurrent development projects (see Fig. 3.6 below.), consideration should also be given on clarifying the timing to interface externally with the hardware team on matters including:

- (1) Collaboration with the hardware team (holding joint reviews, joint tests, joint problem analysis, etc);
- (2) Provisions to the hardware team (software for debugging hardware, user manual, etc);
- (3) Receipt from the hardware team (updated hardware specifications, real machine for debugging / testing, etc).

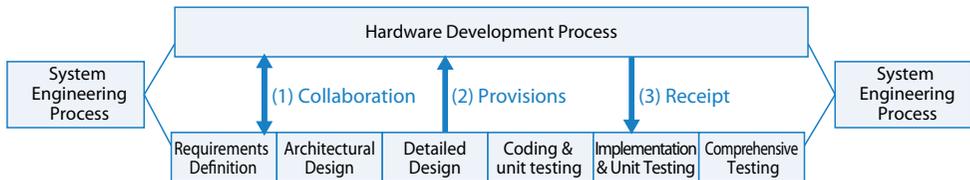


Figure 3.6 Example of Development Process Phases (Concurrent Development)

Case Example of Outsourcing

In development projects that are partially outsourced, the scheduling of activities and tasks pertaining to subcontractor management, including order placement duties, acceptance tests, progress management, quality assurance, and deliverables management are of critical importance.

Especially, the synchronization of internal processes with the processes adopted by the subcontractors is a key matter that requires careful coordination during the process phase stage designing (see Fig. 3.7) designing

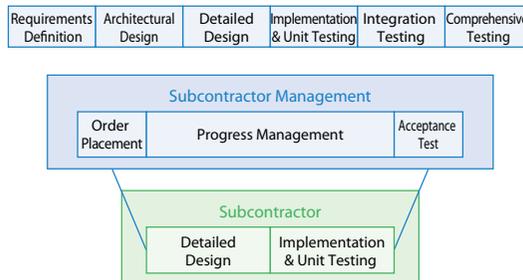


Figure 3.7 Example of Development Process Phases (Outsourcing)

3.4 Planning the Development Project Management (Designing the development process phases in detail)

■ Planning the Embedded Software Development Project Management (Designing the development process phases in detail)

The project management planning for embedded software development must basically not only take account of software development elements but also include the development elements of hardware that constitutes a part of the system to be developed for the targeted product. In this context, project management planning refers to the act of breaking down the high-level (activity-level) work units examined and deemed necessary in advance in overall process phase designing into lower-level work units, placing them on the actual timeline of the development project, allocating them to the organization (department, section, or team), and assigning them respectively to individual engineers. The outcome of all these actions is called the development project plan.

Please note that the description on project management planning provided here is limited to general information, since it is categorized as a part of project management, which is not the main subject matter of this guidebook. For more detailed information on project management planning, please refer to “ESMR Ver 1.0: Embedded System development Management Reference [Plan Description Edition]”.

Points That Should Be Taken Into Account in Embedded Software Development Project Management Planning

During embedded software development project management planning, particular attention should be given to the following matters:

- (1) Consistency with the processes adopted in hardware development and incorporation of the points (timing) to interface with the counterparts at the hardware side;
- (2) Arrangements to carry out multiple work units concurrently where possible by considering the sequential relationship of these work units;
- (3) Distribution and allocation of work units to individual engineers by bearing their skill levels in mind.

Basic Procedure for Project Management Planning

The basic procedure for embedded software development project management planning is as follows (see Fig. 3.8).

Step 1: Determine in which order to carry out each work (decide on the working sequence).

- (1) WBS (Work Breakdown Structure): Estimate the workload of each work unit.
- (2) PERT (Program Evaluation and Review Technique): Clarify the relationship between work units, and determine in which order to carry out each work.
- (3) Set the milestones of the development project.

Step 2: Divide the work and responsibilities.

- (1) Decide which organization and person is going to be responsible of each work.
- (2) Decide who to assign each work to, by taking account of resource constraints.

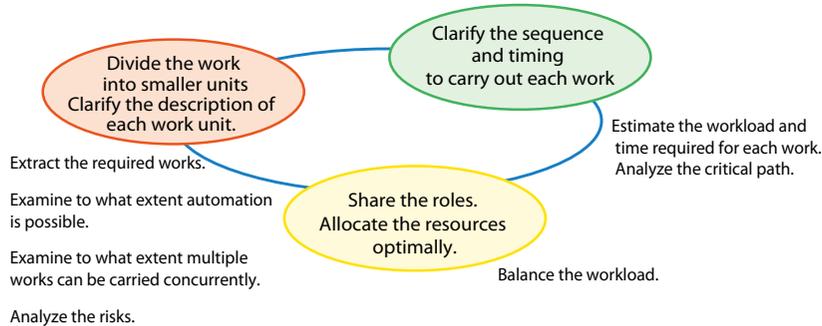


Figure 3.8 Points to Keep in Mind in Development Project Management Planning

Deciding on the Working Sequence

While the main task in overall process phase designing is to clarify the execution level of each work and the logical sequential relationship between the work units, the main task in project management planning is to place these work units on the actual timeline within the specific development period.

Estimating the Workload

The most fundamental information gained during development project management planning is the estimate workload of each work. The workload of each work is estimated by considering the rough amount of man-hours needed to complete each work or the rough volume of the deliverable(s) expected to be outputted from each work. For workload estimation, past data showing the amount of workload required for each work in similar system development projects is often a

useful reference source.

To gain a clearer picture and better understanding of the estimated workloads and how they compare with each other, it is desirable to tabulate them in the form of WBS (see Fig. 3.9).

Deciding on the Working Sequence

The actual order in which each work is carried out is determined, based on the workload of each work and the logical sequential relationship between the work units. Below are some important points that need to be considered in deciding on the working sequence:

- (1) Be aware of the pre-agreed delivery deadline, and see if all the required works can fit in the given timeframe.
- (2) Assuming that there are some types of work that can be handled by multiple departments or engineers, consider the possibility of engaging in concurrent development wherever feasible.
- (3) Give attention to how the development process phases of the software development project are aligned with the development process phases adopted by the developers of hardware and peripherals.
- (4) Also analyze and evaluate the critical path, and identify the risks that may become the bottleneck of the workflow.

In working on the above points of consideration, PERT can serve as an effective tool (See Fig. 3.9).

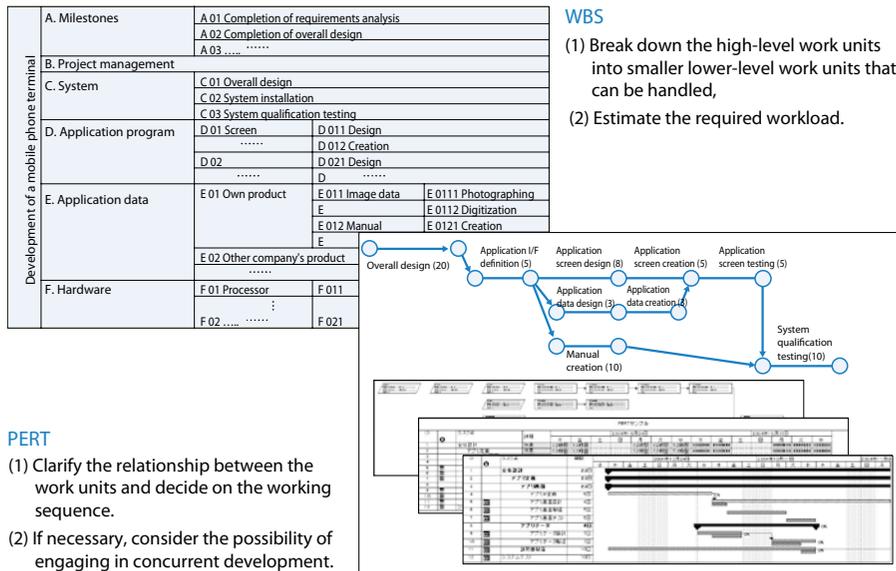


Figure 3.9 WBS and PERT

Deciding on the Work Distribution

To determine which organization (department, team, or subcontractor) should be responsible of which work, or who to assign the individual work, the skill level and workload of engineers available in or out of the organization must be taken into account.

Determining the Organization / Person Responsible of Individual Work

Before assigning the individual engineers what work to take up, there is a need to determine which organization (department, team, or subcontractor) should be in charge of which group of works and who should be responsible of each of them. In deciding these matters, careful consideration must be given to the description of each work to prevent mismatches from occurring at both the organizational level and leadership level.

Allocating the Resources

In large-scaled development projects that require multiple engineers to be engaged in each activity, there is a need to assign them respectively the right amount of work and utilize the available human resources as efficiently as possible.

Another point to think about is the different levels of skill required at different stages of the development process phases, and the adequate number of resources with the right skill level in each

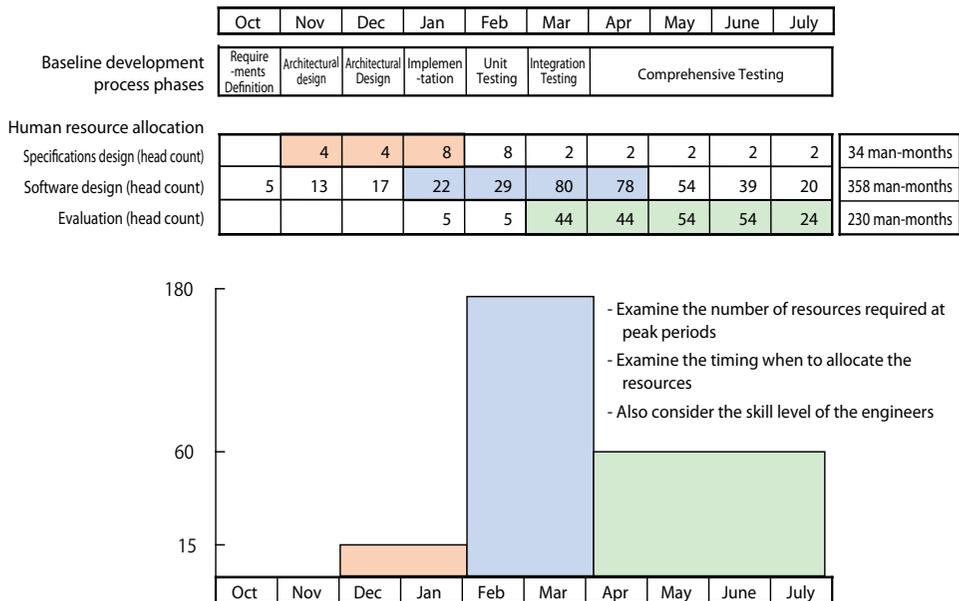


Figure 3.10 Example of Work Distribution

stage. Resource allocation must be handled very carefully especially during the high-demand periods in development when the overall workload reaches the peak, and likewise, at the initial stage of development.

In general, many of the engineers who are already busy engaging in various assignments for other projects are often not available right away. Therefore, in practice, it is normally very important to have a well-managed resource allocation plan that takes full account of the latest workload and work schedule of engineers expected to be called to the project that is being planned (see Fig. 3.10).

Conceptualizing the Planning the Development Project Management

Below are some simple case examples to help understand how development project management planning should be thought out.

Example 1: Small-scaled Development (New)

In case of developing an optical mouse newly

Software characteristics:

Scale of software development: Approximately several hundred lines (newly)
User interface: About three buttons
Number of engineers in charge of development: One or two engineers

In this case, the scale of new development is small. Therefore, it may be reasonable to consider the following adjustments to lighten the overall workload:

- SWP1.2 (1.2.1 Reviewing the Software Requirements Specifications Internally)

Conduct a simplified review process within the development group rather than carrying out the internal review rigidly in an official style.

- SWP4.1 (4.1.1 Preparing for Implementation)

In a completely new development project, there is no need to consider about reusable program units because they do not exist.

Moreover, in a project like this case where the targeted product for development is a mouse, the quality characteristics of the mouse as a functional device (i.e.: performance-related attributes like response time) tend to be weighed relatively more heavily than other aspects. Therefore, in the following tasks:

- SWP2.1 (2.1.1 Confirming the Design Conditions)

Focus particularly on examining the methods of meeting the non-functional requirements on, such as, the operating environment and performance.

•SWP3.3 (3.3.1 Checking the Consistency with Hardware Specifications)

Focus on aligning the software specifications and the hardware specifications so that adequate level of performance of the optical hardware components that control the main functionalities of the mouse can be achieved.

In short, determine the intensity and contents of activities, tasks and sub-tasks to be carried out in the project according to their respective level of importance placed on the basis of various considerations (like those above) that reflect the particular characteristics of the targeted product.

Example 2: Large-scaled Development (Additional)

In case of developing a new car navigation system with additional new functionalities, based on the existing car navigation system

Software characteristics:

Scale of software development: Large scale (several million lines)
User interface: Complex interfaces mainly related to display
Number of engineers in charge of development: 100 engineers or more

In this case, the scale of development is extremely large. Therefore, the following activities should be carried out intensively:

- SUP1 Project Management
- SUP8 Joint Review

Moreover, by considering the fact that the targeted software to be developed is large-scaled and that the software development will be based on existing software with various assets that can be carried over:

• SWP2.1 (2.1.2 Designing the Software Structure)

Design the software structure to be easily adaptable for reuse in the future by aligning it as closely as possible to the structure of the entire system and deliberately separating the core components, among others.

Furthermore, to reflect the complexity and usability of the user interface:

- SWP1.1 (1.1.2 Clarifying the Functional Software Requirements)
(1.1.3 Clarifying the Non-functional Software Requirements)

Considerations on system usability and human-centered design process should also be included.

Appendices

Appendix 1 Terminology	234
Appendix 2 Standards Correspondence Table (Between This Guidebook and X0160)	238
Appendix 3 List of Activities / Tasks / Sub-tasks	239

Appendix 1 Terminology

A

Activity

Activity is a high-level work unit that constitutes a part of the group of specific actions that are performed to complete a process.

For example, the activities to be performed to complete the software engineering process (SWP) are defined in this guidebook to be as follows:

- Software Requirements Definition
- Software Architectural Design
- Software Detailed Design
- Implementation & Unit Testing
- Software Integration Testing
- Comprehensive Software Testing

F

Functional requirements

Functional requirements refer to capabilities like “Can input the data named xxx” and “Can send emails” that the software is required to provide as functionalities to meet the needs of the targeted product and intended users.

Related terminology Non-functional requirements

I

Internal review

Internal review is a form of review held within a group of members in the development project, including the engineers in charge of creating the deliverables, to check from time to time whether appropriate outcome has been produced or not in the course of development. Internal reviews are typically more oriented towards checking the technical aspects, and held without the participation of non-technical stakeholders involved in the product development.

Related terminology Joint review

J

Joint review

Joint review is a form of review held by the members of the development project to check whether appropriate outcome has been produced or not at each milestone of the development process from both the technical and administrative standpoints. This is an opportunity not only for the engineers in charge of creating the deliverables to cross-check them, but also for other stakeholders involved in the product development to participate and check the deliverables from multiple perspectives.

Related terminology Internal review

■ N

Non-functional requirements

Non-functional requirements refer to requirements on, such as, efficiency (eg: “Complete processing within nn seconds”), usability (eg: “Enable the users to operate without relying on manuals”) and portability (eg: “Can be reused”) that are required by the software.

Related terminology Functional requirements

■ P

Process phase designing

Process phase designing is a part of the planning process to place the individual work required for development on the actual timeline of the development project. Upon completion of this work placement process, the activities placed on the project timeline will form the project’s “development process phases”.

The decisions on work placement made in process phase designing are based on the examination of the sequential relationship between each work, the feasibility of concurrent development, and the availability of resources who can be assigned to each work.

PERT (Program Evaluation and Review Technique)

PERT is a technique used to draw up a plan to perform the various types of work identified as required development work through WBS (Work Breakdown Structure) in the most efficient combination to minimize the time required to complete the development project. Through PERT, the project schedule is set on the basis of the best order to carry out the required set of work (analyzed by determining which can be carried out concurrently with others and which must be executed sequentially).

By creating a PERT chart, the project’s critical path can be defined. Critical path is a series of work required to complete a project that altogether determines the total length of time needed to complete the given project. If any of the work on the critical path is delayed, the period required to complete the project will be delayed.

Defining this critical path and managing all the work on the critical path to be executed on time are decisive factors for successful project management.

Process

Generally in any type of work, various tasks need to be performed in the course of meeting the purposes and objectives aimed at achieving through that work. In addressing “what kind of tasks need to be performed”, various inputs and outputs, and the overall contents of these tasks are defined and organized in an orderly manner, which are collectively referred to as the “process”.

In this guidebook, the various types of work that need to be performed in order to proceed with the development of embedded software have been largely divided into the following four processes (or work clusters), which are collectively referred to as the “software development process”:

(1) System Engineering Process

This process mainly consists of the various types of work for developing the embedded system built on the basis of the software that would be embedded.

(2) Software Engineering Process

This process mainly consists of the various types of work related to the actual software development.

(3) Safety Engineering Process

This process consists of the various types of work that should be carried out to develop an embedded system that can be used safely and without anxiety.

(4) Support Process

This process consists of the various types of work for providing a wide range of support that becomes necessary in the course of the development (e.g.: documentation).

■ S

Safety Engineering Process (SAP: SAFety engineering Process)

Safety Engineering Process is a process that consists of an orderly set of work that should be carried out to develop an embedded system that can be used safely and without anxiety.

The following activities are included in this process:

SAP1 Safety Requirements Definition

SAP2 Safety Testing

Software Engineering Process (SWP: SoftWare engineering Process)

Among the various types of work pertaining to embedded software development, the activities and tasks ranging from Software Requirements Definition directly related to software production to Comprehensive Software Testing are defined in this process.

The following activities are included in this process:

SWP1 Software Requirements Definition

SWP4 Implementation & Unit Testing

SWP2 Software Architectural Design

SWP5 Software Integration Testing

SWP3 Software Detailed Design

SWP6 Comprehensive Software Testing

Stakeholder

Stakeholders are individuals with interest in the product, ranging from those belonging to corporate entities to end users.

Stub

Stub is a placeholder for substituting the lower-level module that is called by the tested component (program unit).

Related terminology Test driver

Support Process (SUP: SUPport Process)

Support Process (SUP) consists of supportive activities performed across all stages of the development process (SYP, SWP and SAP) to help manage the respective activities (tasks and sub-tasks) defined in these engineering processes and smoothen the organizational execution of embedded software development.

While in ISO/IEC12207 (for software engineering) and ISO/IEC15288 (for system engineering), more detailed definitions of support process are given by sub-dividing this process into life cycle process groups like organizational project-enabling processes and SW support processes, this guidebook (ESPR Ver.2.0) has focused on defining the following support activities: SUP1, SUP6, SUP7, SUP8, and SUP10.

The following activities are included in this process:

SUP1 Project Management

SUP5 Configuration Management

SUP2 Quality Assurance

SUP6 Problem Resolution Management

SUP3 Risk Management

SUP7 Change Management

SUP4 Documentation & Document Management

SUP8 Joint Review

SUP9 Subcontractor Management

SUP10 Preparation of Development Environment

System Engineering Process (SYP: SYstem engineering Process)

System Engineering Process mainly consists of the activities for defining the system requirements for the embedded system that operates by the workings of the software that is embedded, and for verifying the behaviors as the system.

The following activities are included in this process:

SYP1 System Requirements Definition

SYP3 System Integration Testing

SYP2 System Architectural Design

SYP4 System Testing

T

Task

Task is a lower-level work unit that constitutes a part of the cluster of specific actions grouped by activity that are required to execute the given activity and achieve its pre-defined objectives.

For example, the activity named “Software Requirements Definition” is defined in this guidebook to be composed of the following two tasks:

- Creating the Software Requirements Specifications;
- Reviewing the Software Requirements Specifications.

Test driver

Test driver is a placeholder for substituting the higher-level module that transmits the test data to the tested component (program unit).

[Related terminology](#) Stub

W

WBS (Work Breakdown Structure)

WBS is a technique to decompose the entire project into specific types of work. It is used to break down the project goal (product to be developed) into small pieces of deliverables that are arranged systematically in hierarchical order, and to allocate the necessary work to develop each of these partial deliverables.

Building the WBS involves the sub-dividing of deliverables as detailed as possible to define concrete outputs that can be visualized as direct outcome in response to requirements of particular importance to the project. By building the WBS, the detailed set of work required to achieve the objectives of the project can be identified. Moreover, once the WBS for the given project is established, it will be a useful tool for the stakeholders of the project (sponsors, personnel in charge of marketing, project members, etc) to gain clear preliminary knowledge about the scope of their development project, and help them prevent the project plan from leaving out any elements of required development that may lead to delays in the project schedule due to inevitable later additions of extra work to make up for the missing elements.

Furthermore, by following the WBS closely throughout the project, the extent of impact on the ongoing project caused by the changes in requirement that become necessary after commencing the development can be well-controlled.

Appendix 2 Standards Correspondence Table (Between This Guidebook and X0160)

ESPR (This guidebook)	X 0160-1996 (ISO/IEC12207: 1995) Software life cycle processes
–	Acquisition Process
–	Supply Process
–	Development Process Process Implementation
SYP : System Engineering Process SYP1 System Requirements Definition SYP2 System Architectural Design	System Requirements Analysis System Architectural Design
SWP : Software Engineering Process SWP1 Software Requirements Definition SWP2 Software Architectural Design SWP3 Software Detailed Design SWP4 Implementation & Unit Testing SWP5 Software Integration Testing SWP6 Comprehensive Software Testing	Software Requirements Analysis Software Architectural Design Software Detailed Design Software Coding and Testing Software Integration Software Qualification Testing
SYP : System Engineering Process SYP3 System Integration Testing SYP4 System Testing	System Integration System Qualification Testing Software Installation Software Acceptance Support
–	Operation Process
–	Maintenance Process
SUP : Support Process SUP4 Documentation & Document Management	Documentation Process
SUP : Support Process SUP5 Configuration Management	Configuration Management Process
SUP : Support Process SUP7 Change Management	–
SUP : Support Process SUP2 Quality Assurance	Quality Assurance Process
–	Verification Process
–	Validation Process
SUP : Support Process SUP8 Joint Review	Joint Review Process
–	Audit Process
SUP : Support Process SUP6 Problem Resolution Management	Problem Resolution Process
SUP : Support Process SUP1 Project Management	Management Process
SUP : Support Process SUP3 Risk Management	–
SUP : Support Process SUP10 Preparation of Development Environment	Infrastructure Process
–	Improvement Process
–	Training Process
–	Tailoring Process
SUP : Support Process SUP9 Subcontractor Management	–
SAP : Safety Engineering Process SAP1 Safety Requirements Definition SAP2 Safety Testing	–

□ : Processes defined in this guidebook

Appendix 3 List of Activities / Tasks / Sub-tasks

■ SYP : System Engineering Process

SYP1 System Requirements Definition22	SYP3 System Integration Testing.....43
1.1 Creating the System Requirements Specifications	3.1 Preparing for System Integration Test
1.1.1 Understanding the Product Plan Description and Product Specifications	3.1.1 Preparing for System Integration Test
1.1.2 Analyzing and Sorting the Functional System Requirements	3.1.2 Preparing for System Integration Test
1.1.3 Analyzing and Sorting the Non-functional System Requirements	3.2 Conducting the System Integration Test
1.1.4 Clarifying the System Operational Constraints	3.2.1 Conducting the System Integration Test
1.1.5 Prioritizing the System Requirements	3.2.2 Reviewing the System Integration Test Results
1.1.6 Creating the System Requirements Specifications	3.3 Reviewing the System Integration Test Results
1.2 Reviewing the System Requirements Specifications	3.3.1 Reviewing the System Integration Test Results Internally
1.2.1 Reviewing the System Requirements Specifications Internally	SYP4 System Testing51
SYP2 System Architectural Design32	4.1 Preparing for System Test
2.1 Creating the System Architectural Design Description	4.1.1 Creating the System Test Specifications
2.1.1 Confirming the Design Conditions	4.1.2 Preparing for System Test
2.1.2 Designing the System Structure	4.1.3 Reviewing the System Test Specifications Internally
2.1.3 Designing the Overall System Behaviors	4.2 Conducting the System Test
2.1.4 Designing the Interface	4.2.1 Conducting the System Test
2.1.5 Creating the System Architectural Design Description	4.2.2 Reviewing the System Test Results
2.2 Reviewing the System Architectural Design	4.3 Reviewing the System Test Results
2.2.1 Reviewing the System Architectural Design Description Internally	4.3.1 Reviewing the System Test Results Internally
2.3 Jointly Reviewing the System Architectural Design Description	4.4 Confirming the Completion of System Development
2.3.1 Jointly Reviewing the System Architectural Design Description	4.4.1 Confirming the Completion of System Development

SWP1 Software Requirements Definition	63	3.3.1 Checking the Consistency with Hardware Specifications	
1.1 Creating the Software Requirements Specifications		SWP4 Implementation & Unit Testing	99
1.1.1 Identifying the Constraints		4.1 Preparing for Implementation and Unit Test	
1.1.2 Clarifying the Functional Software Requirements		4.1.1 Preparing for Implementation	
1.1.3 Clarifying the Non-functional Software Requirements		4.1.2 Preparing for Unit Test	
1.1.4 Prioritizing the Requirements		4.2 Conducting the Implementation and Unit Test	
1.1.5 Creating the Software Requirements Specifications		4.2.1 Implementing the Program Units	
1.2 Reviewing the Software Requirements Specifications		4.2.2 Conducting the Unit Test	
1.2.1 Reviewing the Software Requirements Specifications Internally		4.2.3 Reviewing the Unit Test Results	
SWP2 Software Architectural Design	76	4.3 Reviewing the Implementation and Unit Test Results	
2.1 Creating the Software Architectural Design Description		4.3.1 Reviewing the Source Code	
2.1.1 Confirming the Design Conditions		4.3.2 Reviewing the Unit Test Results Internally	
2.1.2 Designing the Software Structure		SWP5 Software Integration Testing	109
2.1.3 Designing the Overall Software Behaviors		5.1 Preparing for Software Integration Test	
2.1.4 Designing the Interface		5.1.1 Preparing for Software Integration	
2.1.5 Estimating the Performance / Amount of Memory Used		5.1.2 Preparing for Software Integration Test	
2.1.6 Creating the Software Architectural Design Description		5.2 Conducting the Software Integration Test	
2.2 Reviewing the Software Architectural Design		5.2.1 Software Integration	
2.2.1 Reviewing the Software Architectural Design Description Internally		5.2.2 Conducting the Software Integration Test	
2.3 Jointly Reviewing the Software Architectural Design		5.2.3 Reviewing the Software Integration Test Results	
2.3.1 Jointly Reviewing the Software Architectural Design Description		5.3 Reviewing the Software Integration Test Results	
SWP3 Software Detailed Design	89	5.3.1 Reviewing the Software Integration Test Results Internally	
3.1 Creating the Functional Unit Detailed Design Description		SWP6 Comprehensive Software Testing	120
3.1.1 Dividing into Program Units		6.1 Preparing for Comprehensive Software Test	
3.1.2 Designing the Program Units		6.1.1 Creating the Comprehensive Software Test Specifications	
3.1.3 Defining the Interface in Detail		6.1.2 Preparing for Comprehensive Software Test	
3.1.4 Estimating the Amount of Memory Used		6.1.3 Reviewing the Comprehensive Software Test Specifications Internally	
3.1.5 Creating the Software Detailed Design Description		6.2 Conducting the Comprehensive Software Test	
3.2 Reviewing the Software Detailed Design		6.2.1 Conducting the Comprehensive Software Test	
3.2.1 Reviewing the Software Detailed Design Description Internally		6.2.2 Reviewing the Comprehensive Software Test Results	
3.3 Checking the Consistency with Hardware Specifications		6.3 Reviewing the Comprehensive Software Test Results	
		6.3.1 Reviewing the Comprehensive Software Test Results Internally	
		6.4 Confirming the Completion of Software Development	
		6.4.1 Confirming the Completion of Software Development	

■ SAP : Safety Engineering Process

SAP1 Safety Requirements Definition	132	SAP2 Safety Testing.....	141
1.1 Creating the Safety Requirements Specifications		2.1 Preparing for Safety Test	
1.1.1 Understanding the Product Plan Description and Product Specifications		2.1.1 Preparing for Safety Test	
1.1.2 Examining the Potential System Failures		2.2 Conducting the Safety Test	
1.1.3 Examining the Requirements to Achieve the Required Safety		2.2.1 Conducting the Safety Test	
1.1.4 Creating the Safety Requirements Specifications		2.2.2 Reviewing the Safety Test Results	
1.2 Reviewing the Safety Requirements Specifications		2.3 Reviewing the Safety Test Results	
1.2.1 Reviewing the Safety Requirements Specifications Internally		2.3.1 Reviewing the Safety Test Results Internally	

■ SUP : Support Process

SUP1 Project Management	149	6.2 Analyzing the Impact and Devising the Acceptable Solution	
1.1 Creating the Project Plan Description		6.3 Implementing the Acceptable Solution	
1.2 Understanding the Project Execution Status		6.4 Tracking the Implemented Solution	
1.3 Controlling the Project		SUP7 Change Management	160
1.4 Creating the Project Completion Report		7.1 Recording the Information on Change Requests	
SUP2 Quality Assurance	152	7.2 Analyzing the Impact of Changes	
2.1 Defining the Quality Objectives		7.3 Devising and Executing the Change Plan	
2.2 Establishing the Quality Assurance Method		7.4 Reviewing the Outcome of the Changes Made	
2.3 Controlling the Quality Based on Quality Visualization		SUP8 Joint Review	161
SUP3 Risk Management	154	8.1 Preparing for the Review	
3.1 Identifying and Understanding the Risks		8.2 Carrying Out the Review	
3.2 Monitoring the Risks		8.3 Acknowledging and Following Up on Matters That Have Been Reviewed	
3.3 Determining and Executing the Risk Treatments		SUP9 Subcontractor Management	
SUP4 Documentation & Document Management		9.1 Preparing for Order Placement and Entering into Contract	
4.1 Creating and Reviewing the Documents		9.2 Monitoring the Outsourced Tasks	
4.2 Distributing the Documents		SUP10 Preparation of Development Environment.....	162
4.3 Maintaining and Managing the Documents		10.1 Devising the Development Environment Preparation Plan	
SUP5 Configuration Management	156	10.2 Building the Development Environment	
5.1 Understanding the Objects of Configuration Management		10.3 Maintaining the Development Environment	
5.2 Managing the Configuration Management / Change Management History			
SUP6 Problem Resolution Management.....	158		
6.1 Recording the Problems and Analyzing the Causes			

Afterword

In order to complete the development of software as a product, various types of work need to be performed in layers. The question is, “What exactly are the types of work that need to be performed to fully develop a software as a product, especially when the product is embedded software?” As one of the responses to this question, the members of Development Process Technical Working Group of Embedded Software Development Improvement and Promotion Committee under Ministry of Economy, Trade and Industry (METI) took over two years to study and define the specific set of work required in embedded software development, and organized their findings in the form of a document titled “ESPR: Embedded System development Process Reference – Development Process Guide for Embedded Software”, that was released in the fall of 2006.

This 2006 fall version, commonly known as ESPR Ver.1.0, focused on defining the mandatory set of work directly related to software engineering, and on describing the procedures and precautions for each of these works in an orderly manner, using expressions that can be well comprehended by the engineers at the development sites.

Although needless to say, successful software development project involves more than just carrying out the set of work directly related to software engineering, and various other types of work related to management and support are also equally important. In light of this recognition, the contents of ESPR Ver.1.0 have been enriched by including descriptions on three additional processes – System Engineering Process, Support Process, and Safety Engineering Process – and issued this revised version as ESPR Ver.2.0. It is the sincere hope of everyone who contributed to the making of this new document to see ESPR Ver.2.0 being used as widely as or even more than its preceding version 1.0.

Lastly but not least, the names of all the contributors of this publication have been listed on the final page as a sign of our profound gratitude to them all.

November 2007

Embedded Software Development Improvement and Promotion Committee

Authors and editors

ABE Koji	CSK SYSTEMS CORPORATION
ASAI Makio	NIPPON ELECTRIC CONTROL EQUIPMENT INDUSTRIES ASSOCIATION
CHENG Zixue	THE UNIVERSITY OF AIZU
FUJIMURA Hiroshi	NEC Communication Systems, Ltd.
HIRAO Yuji	Nagaoka University of Technology
HIRAYAMA Masayuki	IPA/SEC(TOSHIBA CORPORATION)
IGARI Hideo	IPA/SEC(Yokogawa Digital Computer Corporation)
IWAHASHI Masami	Mitsubishi Electric Mechatronics Software Corporation
KANEDA Mitsunori	TOSHIBA SYSTEM TECHNOLOGY CORPORATION
KANEMOTO Shigeru	THE UNIVERSITY OF AIZU
MIURA Kunihiko	YAZAKI Corporation
MIZUGUCHI Daichi	National Institute of Advanced Industrial Science and Technology
MUKAIDONO Masao	MEIJI UNIVERSITY
MURAMATSU Akio	FUJITSU LIMITED
MURO Shuji	IPA/SEC(Yokogawa Digital Computer Corporation)
NAGATOMO Yuji	VeriServe Corporation
NAKAGAWA Masamichi	Matsushita Electric Industrial Co., Ltd.
NONAKA Makoto	PA/SEC(TOYO UNIVERSITY)
OHNO Katsumi	TOYOTA TECHNICAL DEVELOPMENT CORPORATION
OHTA Takashi	NEC Corporation
SATO Yoshinobu	Tokyo University of Marine Science and Technology
SUGIYAMA Hidetoshi	Canon Inc.
SUNAZUKA Toshihiko	Sunazuka Consulting Service, Inc.
SUZUKI Toshihiko	VeriServe Corporation
TAKANO Taiko	Hitachi, Ltd.
TAMARU Kiichiro	IPA/SEC(TOSHIBA CORPORATION)
TANABE Yasuo	Japan Functional Safty Inc.
TATSUNO Yukio	Oki Information Systems Co., Ltd.
TOMOBE Masaaki	Yokogawa Electric Corporation
TSAI Kuangchih	Business Cube & Partners, Inc.
WATANABE Masato	CSK SYSTEMS CORPORATION
YAMAZAKI Taro	IPA/SEC(Nihon Unisys, Ltd.)
YOSHIOKA Ritsuo	Japan Functional Safety Laboratory

(Affiliations are as of the publication of Japanese edition)

Contributors to English translation version

SHIMIZU Tatsuo	IPA/SEC
MIURA Atsuko	IPA/SEC
MATSUDA Mitsuhiro	IPA/SEC
MIHARA Yukihiro	IPA/SEC

ESPR

**Embedded System development Process Reference guide
Ver.2.0**

October 31, 2012
Written and edited by Software Engineering Center,
Technology Headquarters,
Information-technology Promotion Agency, Japan

<http://www.ipa.go.jp/english/sec/>

Copyright © 2012, IPA/SEC
