Chapter 3

System Engineering

Key Takeaway Points

- System engineering is a multidisciplinary approach to develop systems that involve hardware, software, and human components.
- System engineering defines the system requirements and constraints for the system. It allocates the requirements to the hardware, software, and human subsystems, and integrates these subsystems to form the system.
- Software engineering is a part of system engineering.

Many systems are embedded systems. An embedded system consists of hardware, software, and human components. These components interact with each other to accomplish the mission of the system. An example is an airport baggage handling system (ABHS). It is responsible for moving the baggage from one place to another within an airport. Its hardware includes conveyors and destination-coded vehicles (DCV) running on high-speed tracks. Its software handles luggage check-in and controls the hardware. The human workers operate the hardware and software. Other examples of embedded systems include mail handling systems, air traffic control systems, and manufacturing process control systems. System development of such systems must consider the total system rather than the software system alone. A system engineering approach is required.

Developing a software-only system may need a system engineering approach as well. Consider, for example, an enterprise resource planning (ERP) system. An ERP system is an integrated management information system for the whole organization. It provides integrated support to all functions of the organization including marketing, sales, manufacturing, project management, supply chain management, accounting, customer support, customer relationship management, access control, and more. It automates all these business functions and the workflows among these functions. System engineering is required to develop such a system for several reasons. The system involves many business and organizational functions. The functions involve different disciplines. A multidisciplinary development team is required. The system is large and complex. It may involve legacy systems that have been developed and used for years. How to design the system and integrate with existing databases and business processes is a big challenge: A system engineering approach is needed.

System engineering is a discipline of its own. It is impossible to cover the discipline in one chapter. Therefore, this chapter only serves as an introduction. It aims to provide the software engineer a basic understanding of the processes and activities of system engineering. After reading the chapter, you will understand the following:

- System engineering process.
- System modeling and design techniques.
- Allocation of system requirements to subsystems.
- System configuration management.

3.1 WHAT IS A SYSTEM?

A system consists of components that interact with each other to accomplish a purpose. A system can be big or small, complex or simple, and exist physically or only conceptually. For example, the universe is a very large system that has been in existence for millions of years. An ant is a very small system. These systems are natural systems. In contrast to natural systems, there are many man-made systems. Man-made systems may exist physically or only conceptually. Mathematical logic, number systems, measurement systems, and many classification systems are examples of conceptual systems. Sprinkler systems that water gardens and lawns and telephone systems that connect millions of families are examples of man-made systems that exist physically. Computer-based systems are man-made systems that include software as a subsystem or component. Examples of computer-based systems include telecommunication systems, email systems, library information systems, and process control systems. Some of these are software-only systems such as email systems. Others consist of both software, hardware, and human subsystems such as the ABHS. All systems share a set of properties or characteristics:

1. Each system consists of a set of interrelated and interacting subsystems, components, or elements. For example, the ABHS consists of several subsystems, which interact with each other to accomplish the mission of the system.
2. A system may be a subsystem of an even larger system, which in turn may be a subsystem of another system. For example, the conveyor system and the luggage check-in software system are subsystems of the ABHS. But the ABHS is a subsystem of the airport system. In this sense, the relationship between systems and subsystems is a recursive, whole-part relationship. This relationship forms a whole-part hierarchy. Each system or subsystem occupies a position in the hierarchy.
3. Each system exists in an environment and interacts with its environment. For example, the ABHS exists in the airport environment and interacts with the flight information system. The ABHS also works with the departing and arriving flights to load and unload luggage.
4. Systems are ever evolving due to internal or external causes. For example, if the economy is booming, then the number of business travelers as well as leisure

needs to expand. Technology advances may encourage the airport administration to replace bar code scanners with radio frequency identification (RFID) devices. Internal causes include detected design flaws, component defect, software bugs, and the like. All these require change to the system and cause the system to evolve.

3.2 WHAT IS SYSTEM ENGINEERING?

System development for the ABHS must consider the total system rather than the software system alone. In addition, the ABHS involves multiple engineering disciplines including electrical and electronic engineering, mechanical engineering, civil engineering, and software engineering, among others. Engineers of these disciplines must work together to develop the system. In general, system development for embedded systems involves many engineering and nonengineering disciplines:

- *Electrical and electronic engineering.* Many systems use very large-scale integrated circuits (VLSIC), application-specific integrated circuits (ASIC), sensors, relays, switches, and power supply components, among many others. For example, the ABHS employs bar code printers to generate bag tags and bar code scanners at intersections of the conveyor network to read the bar codes. Electrical and electronic engineers perform analysis, design, integration, and testing of such components and subsystems.
- *Mechanical engineering.* Mechanical devices are used by many systems to perform physical work. For example, the ABHS uses conveyors to move the luggage and pushers at intersections to direct the luggage toward their destinations. The system also uses high-speed tracks to transport the luggage between the terminals. Therefore, the analysis, design, construction, testing, and installation of the ABHS require the participation of mechanical engineers and technicians.
- *Civil engineering.* Large, interconnected structures are needed by the ABHS to protect the equipment and the luggage. The conveyors and high-speed tracks must be mounted on concrete bases. The design, construction, and testing of such structures and bases, among others, are the focus of civil engineering.
- *Software engineering.* Software is responsible for performing the most critical and challenging tasks of the total system. It carries out information processing activities and controls the other equipment and devices of the total system. In the ABHS, the luggage check-in software subsystem lets an airline agent check in luggage for the passengers. The luggage bar code processing software directs the mechanical pushers at intersections of the conveyor network to dispatch the luggage toward their destinations. Software engineers are responsible for the analysis, design, implementation, testing, and deployment of the software subsystems and components of the total system.
- *Computer science.* Computer and information sciences are the foundations of software engineering and provide the technologies that are used in most systems. For example, the ABHS involves at least programming languages, algorithms and data structures, database systems, analysis and design of real-time embedded systems, computer networks, and computer security.

- *Business administration and economics.* Business aspects are important considerations for most system development projects. For example, system engineering for the ABHS needs to analyze the impact of local, national, and global economies on the business of the airport and the ABHS in particular. It needs to estimate the volume of luggage to be handled, future growth, revenues, costs, and return on investment. All these require knowledge in accounting, finance, management, and economics. The analysis, design, and implementation of the human resource subsystem for the ABHS require knowledge and experience in human resource management.

This is not an exhaustive list. Many systems need other engineering and nonengineering disciplines such as mathematics, natural sciences, social sciences, law, and humanities, to mention a few. In addition, system engineering must consider safety, security, reliability, and many other attributes. For example, the design of the ABHS must consider the safety of the workers working by the conveyors and high-speed tracks. The system must include equipment to check luggage for compliance of security rules. The reliability of the ABHS is critical because it affects hundreds of thousands of passengers.

In summary, system development is an interdisciplinary effort. It involves hardware, software, and human resource developments. These imply a number of challenges. For example, how do we identify and formulate system requirements for such systems? How do we design and construct such systems? And how do we manage the development activities? How do we measure the performance, reliability, safety, and other factors of such systems? How do we assess the impact of the system to the society and environment? System engineering is the engineering discipline that answers these questions. System engineering addresses these challenges. In particular, system engineering emphasizes the following elements:

1. *A system engineering process that covers the entire life cycle of the system.* The system development process defines the phases and the phase activities required to develop the system. The process addresses the complete system life-cycle activities beginning with the initial system concept to the maintenance and retirement of the system. This system-oriented approach encourages the engineering teams to focus on the customer's business needs and priorities, and develop the system to satisfy such needs and priorities.
2. *A top-down divide-and-conquer approach.* This approach decomposes the total system into a hierarchy of subsystems and components, and develops each of them separately. With this approach, the ABHS is decomposed into subsystems. Each subsystem is developed by a team of engineers. This divide-and-conquer strategy enables the engineering teams to develop very large, complex systems.
3. *An interdisciplinary approach to system development.* As discussed above, system development is an interdisciplinary effort. That is, interdisciplinary teams work with each other to carry out the system development activities.

Figure 3.1 illustrates the phases and workflows of a system engineering process. The boxes represent the phases or activities. The solid arrow lines represent work-

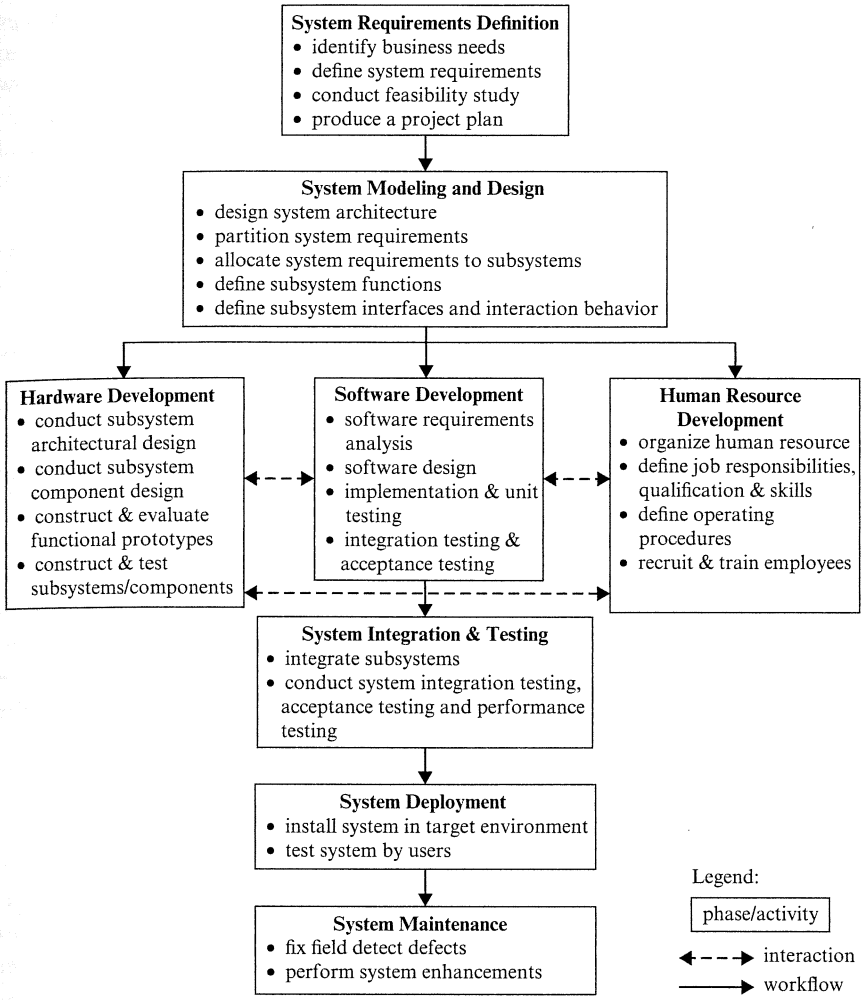


FIGURE 3.1 A system engineering process

hardware, software, and human resource development activities. The type of system influences the activities involved. For example, if the system is a software-only system, then the hardware development activities are not performed. The phases of the system engineering process are outlined below and detailed in the following sections:

1. *System requirements definition.* Systems are constructed to satisfy business needs. For example, an ABHS is developed to satisfy the needs of an airport. System requirements definition focuses on identifying business needs, and specifies the capabilities of the system to meet the business needs. The result is a system requirements specification and a project plan to design, implement, test, and deploy the system.

2. *System architectural design.* The system requirements specify the capabilities of the system needed to solve the business problems. They do not state how to provide the capabilities. System architectural design drafts the solution or how to provide the capabilities. In particular, this activity defines the system architecture or overall structure of the system. That is, it depicts the subsystems and the relationships between the subsystems. This activity also assigns the system requirements to the subsystems.
3. *Specify subsystem functions and interfaces.* Based on the system requirements assigned to the subsystems, the subsystems' functions and interfaces are specified. This activity also specifies how the subsystems interact with each other.
4. *Subsystems development.* After system modeling and design, the subsystems are developed by separate engineering teams simultaneously. The engineering teams refine the system requirements allocated to the separate subsystems, and design and implement the subsystems to satisfy the requirements. For example, the software engineering team refines the software requirements, and design, implement and test the software subsystems according to the software requirements.
5. *System integration, testing, deployment, and maintenance.* The subsystems are integrated. Integration testing is performed to ensure that the subsystems work with each other. System acceptance testing is performed to ensure that the system indeed delivers the capabilities specified in the system requirements specification. The system is then installed in the target environment and tested by the users. Errors, defects, and user feedback are addressed. The system enters the maintenance phase.

3.3 SYSTEM REQUIREMENTS DEFINITION

System requirements definition identifies the business needs and specifies the system requirements. It begins with an initial system concept, and expands and refines the concept. During this process, a set of capabilities that the system must deliver is identified. These capabilities are formulated as system requirements. The system requirements include functional requirements, quality requirements, performance requirements, and other system-specific requirements. This section describes the system requirements definition activity.

3.3.1 Identifying Business Needs

Identifying business needs begins with an information collection activity. That is, information about the business goals and the current business situation is collected. The team identifies the gap between the current situation and the business goals, and derives the business needs. Consider, for example, a small town airport that wants to install an ABHS. To identify the business needs, the team collects information about the current airport and business goals of the new airport. Information about the new airport may include the population and the nature of the businesses of the small town, number of

and other factors. The information collection activity answers the following questions:

1. What is the business that the system will automate?
2. What is the system's environment or context?
3. What are the business goals or product goals?
4. What is the current business situation, and how does it operate?
5. What are the existing business processes, and how do they relate to each other?
6. What are the problems with the current system?
7. Who are the users of the current system and the future system, respectively?
8. What do the customer and users want from the future system, and what are their business priorities?
9. What are the quality, performance, and security considerations?

The information collection activity uses several information collection techniques:

1. *Customer presentation.* Customer presentation is an effective approach to gathering information. It takes place at the very beginning of the project. A management or senior personnel designated by the customer presents an overview of the current business, known problems, what the customer and users expect the system to accomplish, and what their business priorities are. The list of questions presented above may serve as the focus of the presentation. The presentation should be limited to no more than two hours including questions and answers.
2. *Study of current business operations.* The team may pay a visit to the business environment. The team may request a guided tour of the customer's business operations where they may collect paperwork forms, descriptions of operating procedures, policies, and other relevant materials. The team should study these materials carefully. These activities help the team understand the customer's business entities, business processes, and workflows. The workflows may include information flows and material flows.
3. *User survey.* User surveys are useful for acquiring users' opinions about the current system and their expectations of the new system. The survey questionnaire should be brief and focus on important issues. Use different survey questionnaires for different groups of users. Issues that require clarification are identified and addressed during user interviews.
4. *User interview.* User interviews are useful for acquiring information that is difficult to obtain through user surveys. They are also useful for clarifying issues that are not clear in the user surveys. The interviews are conducted with selected users, either jointly or individually. Each interview session should last about one hour. A list of items to be discussed during the interviews should be prepared beforehand. The list may be shared with the users prior to the interviews.
5. *Literature survey.* Literature survey provides additional information to help the development team understand the application domain. Literature survey should focus on similar projects, domain knowledge, business processes, government

3.3.2 Defining System Requirements

The next step is deriving system requirements from the business needs identified. For example, the capabilities of the ABHS are derived to satisfy the needs of the ABHS. However, not all needs are to be satisfied due to budget, delivery schedule technology, and political constraints as well as cost-effectiveness considerations. A feasibility study is sometimes performed to ensure that the team can deliver the capabilities with the budget and schedule constraints. The capabilities that the system must deliver are formulated as system requirements. In illustration, the ABHS project may identify many requirements to satisfy the needs. Some of them are stated below to serve as examples. The requirements are numbered to facilitate reference.

R1. ABHS shall check in and transport luggage to departure gates and baggage claim areas according to the destinations of the passengers.

R2. ABHS shall allow airline agents to inquire about luggage status and to locate luggage.

R3. ABHS shall check all baggage and detect items that are prohibited.

R4. ABHS shall be able to serve 20,000 passengers per day.

An end product of this phase is a system requirements specification. The specification may include constraints that restrict the solution space. For example, the ABHS may include constraints on the available space to build the conveyor and high-speed track network. It may include constraints on the types of equipment that must be used. Another product of this phase is a project plan. The project plan outlines the milestones of the project, schedules the development activities, and allocates resources to the activities. A system test plan may be produced. The test plan specifies the system test objectives, test procedures, and needed resources.

3.4 SYSTEM ARCHITECTURAL DESIGN

After the system requirements are identified, the next logical step is to design the system to satisfy the system requirements. Ideally, the system should be designed and implemented by engineers who are experts in all the engineering disciplines involved. Unfortunately, such engineers are hard to find and expensive to hire. Therefore, systems are usually decomposed into a hierarchy of subsystems, which can be developed by engineers of separate disciplines. For example, electrical subsystems are developed by electrical engineers. Mechanical subsystems are developed by mechanical engineers and software subsystems are developed by software engineers. These subsystems are then integrated during the system integration and testing phase. This approach makes it easier to find qualified engineers. It also reduces the system development complexity and costs. System architectural design performs the following interrelated activities:

1. *Decompose the system into a hierarchy of subsystems.* This step decomposes

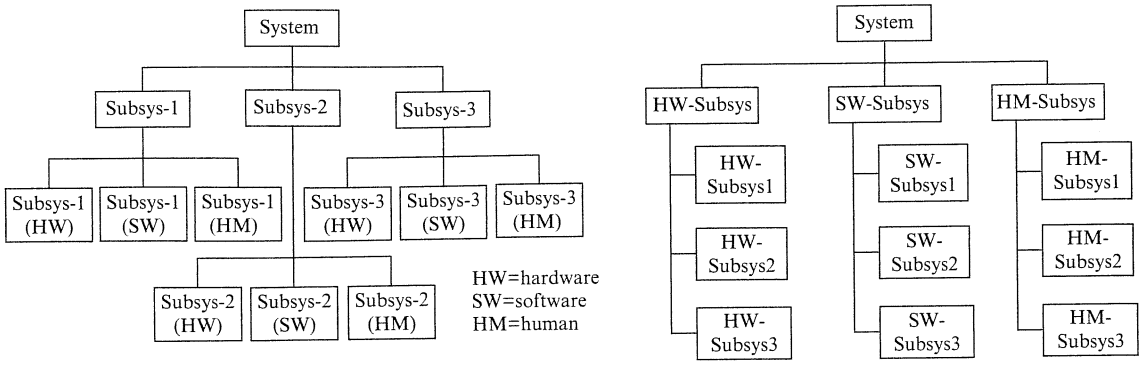
reduces the complexity and costs of system development because the subsystems are easier to design and implement.

2. *Allocate system requirements to subsystems.* This activity assigns the system requirements to the subsystems. It aims to reduce the number of requirements that are shared among the subsystems. In this way, the responsibilities of the subsystems are clearly defined. In addition, the requirements allocated to a subsystem should be functionally related. The allocation is shown in a requirement–subsystem traceability matrix.
3. *Visualize the system architecture.* This activity shows the architectural design as a diagram that consists of the subsystems and the relationships between the subsystems.

3.4.1 System Decomposition

One important task of system architectural design is identifying the subsystems of the system. A top-down, divide-and-conquer approach is often used. In particular, the approach decomposes the system into a hierarchy of subsystems. This approach reduces the complexity of system development because each subsystem is easier to design and implement. There are different ways to decompose a system. Therefore, the result is not unique. System decomposition aims at accomplishing the following goals:

1. *The result should enable separate engineering teams to develop the subsystems.* This reduces the system development costs because it reduces the number of engineers who are specialized in multiple engineering disciplines. It also simplifies the communication between the teams and reduces the communication overhead.
2. *The result should facilitate the use of commercial off-the-shelf (COTS) parts.* COTS are third-party products that can be purchased or acquired. The use of COTS increases productivity and quality while it reduces cost and time to market. For example, the design of the ABHS should consider using COTS for the conveyor systems, high-speed tracks, bar code readers, and luggage check-in software rather than developing these from scratch.
3. *The result should partition or nearly partition the system requirements.* That is, few subsystems share requirements with other subsystems. This reduces the possibility that some requirements are not fulfilled adequately due to a misunderstanding between the teams that share the requirements.
4. *Each subsystem should have a well-defined functionality.* This makes the subsystems easy to understand. For example, the luggage check-in subsystem deals with luggage check-in. The conveyor subsystem is responsible for moving the luggage within a terminal.
5. *The subsystems should be relatively independent.* That is, changing a subsystem does not affect the other subsystems. Such a modular design facilitates system maintenance. For example, the luggage check-in subsystem and the conveyor subsystem are two independent subsystems. Either of them can be replaced without



(a) Partition according to major functionality

(b) Partition according to hardware, software, and human subsystems

FIGURE 3.2 Partitioning a system into a hierarchy of subsystems

6. *The subsystems should be easy to integrate.* That is, the subsystems should have simple interfaces and interaction behavior. This simplifies the communication between the subsystems. It facilitates subsystem integration, integration testing, and system maintenance.

System decomposition applies a number of strategies, which may affect each other. Therefore, they should be applied iteratively until a satisfactory result is obtained. The strategies are:

1. Decompose the system according to system functions.
2. Decompose the system according to disciplines.
3. Decompose the system according to existing architecture.
4. Decompose the system according to the functional units of the organization.
5. Decompose the system according to models of the application.

Decomposing the system according to system functions is a frequently used strategy. It identifies the functions of the systems from the system requirements. That is, it partitions the system requirements into nearly disjoint functional clusters. It then identifies the functional subsystems according to the functional clusters. If there are three functional clusters, then the system is decomposed into three functional subsystems. Figure 3.2(*a*) shows that the system is decomposed into Subsys-1, Subsys-2, and Subsys-3. The subsystems are then decomposed into domain-specific subsystems, that is, hardware, software, and human resource subsystems. This approach is used if there are development teams to develop the functional subsystems, or the functional subsystems can be ordered from a third party. Sometimes engineers need to decompose the system requirements into lower-level requirements to make the subsystems relatively independent. To illustrate, suppose that the ABHS has the following requirement:

R1. ABHS shall check in and transport luggage to departure gates and baggage

This requirement includes several functions such as luggage check-in and luggage transportation. If the airport has more than one terminal, then the luggage transportation function involves conveyors and high-speed tracks. Therefore, the requirement should be decomposed. Requirements decomposition should ensure that the low-level requirements are equivalent to the high-level requirements. That is, the high-level requirement and the resulting low-level requirements state the same capabilities. Assume that each flight's check-in area and departure gate are located in the same terminal. So are the arrival gate and the baggage claim area. The requirement R1 can be decomposed into the following:

R1.1. ABHS shall allow airline agents to check in luggage.

R1.2. ABHS shall transport luggage to their destinations within the airport.

R1.2.1. ABHS shall transport luggage from check-in areas to departure gates.

R1.2.2. ABHS shall transport luggage from arrival gates to baggage claim areas.

R1.2.3. ABHS shall transport luggage from arrival gates to departure gates for transfer passengers.

R1.2.4. ABHS shall transport luggage within a terminal using conveyors.

R1.2.5. ABHS shall transport luggage between terminals using DCVs running on high-speed tracks.

R1.3. ABSH shall control the transportation of the luggage within and between the terminals.

Requirement R4 is a performance requirement and relates to several subsystems. It should be decomposed as well so that the lower-level requirements can be assigned to separate subsystems. For example, the lower-level requirements would specify the required speed and volume of luggage for the subsystems. This must consider the number of terminals, check-in areas, and conveyor systems per terminal, hours of operation, and the like. The decomposition must also take into account a certain percentage of redundancy to cope with extremely high demand during the holiday season. The following are examples of the lower-level performance requirements:

R4.1. Each check-in area shall handle 1,150 check-in baggages per day.

R4.2. Each check-in agent shall check in an average three passengers per minute.

R4.3. Each conveyor hardware shall scan and transport 500 check-in pieces of luggage per hour.

R4.4. ABHS control software shall process 2,300 check-in bags per day and 1,000 bar code scan requests per hour.

Next, the system requirements are partitioned into functional clusters. That is, requirements that share the same functionality are grouped together to form a functional cluster. The functional clusters are used to derive the functional subsystems. Figure 3.3 shows the functional clusters and subsystems derived from the system requirements.

Functional Cluster	Functional Description	System Requirements	Functional Subsystem Identified
Luggage check-in	This functional cluster processes luggage check-in.	R1.1, R4.1, R4.2	Luggage check-in subsystem
Conveyor	This functional cluster is responsible for moving luggage within a terminal.	R1.2.1, R1.2.2, R1.2.3, and R1.2.4, R4.3	Conveyor subsystem
High-speed track	This functional cluster transports luggage between terminals.	R1.2.3 and R1.2.5	High-speed track subsystem
Software control	This functional cluster controls the hardware to transport luggage within and between the terminals.	R1.3, R4.4	Software control subsystem

FIGURE 3.3 Functional clusters and functional subsystems

Another strategy is to decompose the system according to the engineering disciplines. It results in subsystems that are named after the disciplines such as electrical subsystem, electronic subsystem, mechanical subsystem, and software subsystem. In Figure 2.2(*b*), the total system is decomposed into hardware, software, and human subsystems. These are decomposed into functional subsystems. This approach is used if the hardware, software, and human subsystems are developed by hardware, software, and human engineering teams or departments.

Decomposing the system according to existing architecture is applied when the goal of the project is to extend or enhance an existing system. The approach could reduce the development costs and avoid the potential risks due to modifying the existing system architecture. The system can also be decomposed according to the organization's functional units. For example, the manually operated baggage handling system of the small town airport may consist of luggage check-in, transport, and baggage claim departments. These may suggest three major subsystems for the new system.

If models of the application or system are constructed to help understand the application or existing system, then these models may be used to guide the decomposition. For example, models may be constructed to understand the workflow of the existing airport luggage handling system. The models may show that baggages are manually checked in and transported to the departure gates. Moreover, bags are unloaded from arriving flights and transported to the baggage claim areas and transfer gates. From the models, subsystems may be identified and used as the initial decomposition of the system. The initial decomposition is then refined and evolved into a hierarchy of subsystems.

3.4.2 Requirements Allocation

Once the system is decomposed into a hierarchy of subsystems, the system requirements are assigned to the subsystems. The allocation considers several factors including developmental and operational costs, performance, quality of service, cost-effectiveness, ease of change, among others. For example, some system functions may be implemented by either hardware or software. Generally speaking, application specific integrated circuits provide better performance. If performance is not an issue, then some of the system requirements may be assigned to the software subsys-

	Luggage Check-In Subsystem	Conveyor Subsystem	High-Speed Track Subsystem	Software Control Subsystem
R1				
R1.1	x			
R1.2				
R1.2.1		x		
R1.2.2		x		
R1.2.3		x	x	
R1.2.4		x		
R1.2.5			x	
R1.3				x

FIGURE 3.4 Requirements to subsystems traceability matrix

depends on the results of the previous steps. If the subsystems are derived from the functional clusters, then the mapping from the functional clusters to the subsystems is the allocation. This is the case for the ABHS example, as Figure 3.3 shows.

The allocation of the system requirements to the subsystems is visualized in a traceability matrix as shown in Figure 3.4. The rows show the requirements while the columns show the subsystems. The entries indicate the allocation of the requirements to the subsystems. For a real-world project, the traceability matrix is large because it involves hundreds of requirements and many subsystems. Therefore, Figure 3.4 is only an illustration. The matrix has a number of advantages:

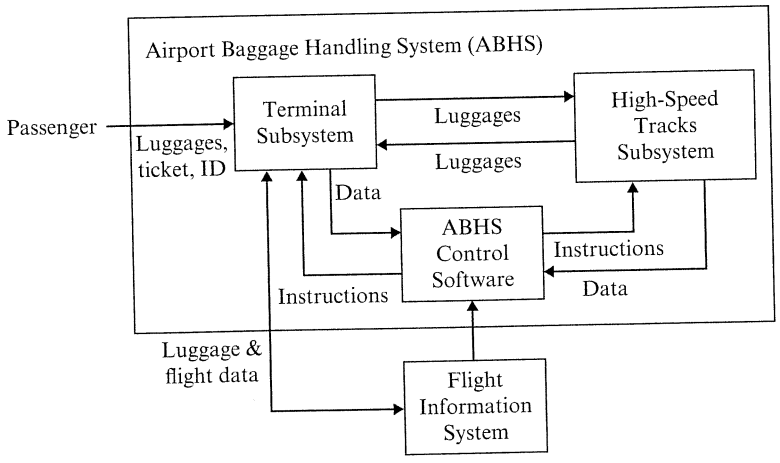
1. *It can check that every requirement is assigned to a subsystem.* That is, each leaf-level requirement row shows at least one cross ('x'). If a leaf-level row does not contain a cross, then the system requirement is not assigned to any subsystem.
2. *It can check if the allocation is appropriate.* Assigning a system requirement to several subsystems should be avoided because it is difficult to divide the functionality among the subsystems. The traceability matrix can detect such cases by counting the number of crosses on each row. If a row contains many crosses, then the requirement is assigned to many subsystems. In this case, decomposing the requirement is desired.
3. *It shows which requirements are assigned to a subsystem.* That is, the crosses shown in each column indicate the requirements that are assigned to the subsystem represented by the column. The subsystems must satisfy the requirements, respectively.
4. *It can check if a subsystem is assigned too many responsibilities.* If a column contains many crosses, then the subsystem is assigned many responsibilities. The subsystem may be decomposed to distribute the requirements among the lower-level subsystems.
5. *It can check the functional cohesion of the subsystems.* The requirements that are assigned to a subsystem should be related functionally. If the requirements indicated by the crosses on a column are not related, then the subsystem represented by the column has low functional cohesion. The requirements should be partitioned into several functional clusters. The subsystem should be decomposed into functional subsystems corresponding to the functional clusters.

3.4.3 Architectural Design Diagrams

It is a common practice to construct application models and system models during system design. These models help the team understand and analyze the application domain, business processes, and workflows to identify problems, and develop and evaluate design solutions. Various diagrams are used to depict different aspects of the application and the system. Block diagrams, Unified Modeling Language (UML), and its extension, System Modeling Language (SysML), and data flow diagrams are widely used during system modeling and design.

Block diagrams use rectangles to represent subsystems and components and directed edges to denote workflows between the subsystems and components. As an example, Figure 3.5 shows a high-level block diagram for the ABHS. The block diagram indicates that an ABHS consists of terminal subsystems, connected to a high-speed track subsystem. Passengers check in luggage with the terminal subsystem. The high-speed track subsystem transports luggage between the terminals. Both of these subsystems interact with the ABHS control software, which interacts with the flight information system. Figure 3.6 shows a refinement of the terminal subsystem. In the figure, hardware, software, and human subsystems are displayed.

UML was initially created for modeling software applications and systems. Its generality makes it a useful tool for system modeling as well. This is achieved by using the stereotype mechanism provided by UML. For example, Figure 3.7 shows a component diagram that models the hardware and software of a radio communication system (RCS). It uses the stereotype "<<subsystem>>" to extend the component modeling construct to model a subsystem. The diagram shows that an RCS consists of three subsystems: a base station subsystem, an account management subsystem, and a mobile units subsystem. The working of an RCS is similar to a cellular network except that it has only one base station with high-power transceivers. The high-power transceivers can service a much larger area than a single base station of a cellular network. Through the relay of the high-power transceivers, mobile units can



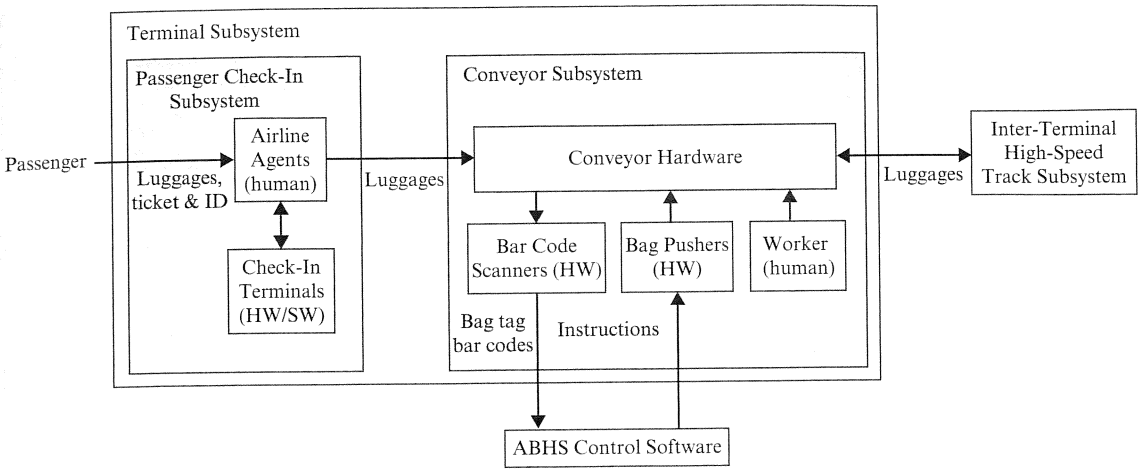


FIGURE 3.6 Block diagram showing refinement of a subsystem

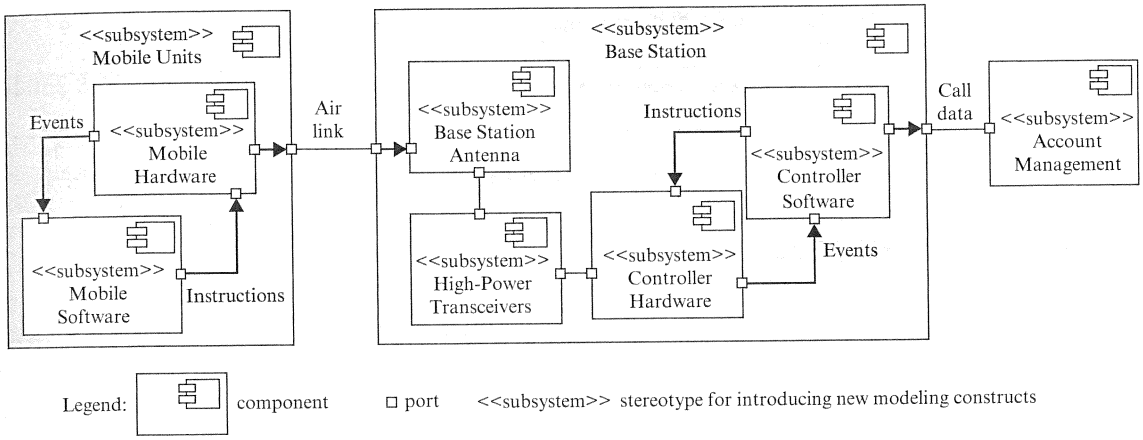


FIGURE 3.7 System modeling using a stereotyped component diagram

communicate in the service area. The figure shows that each mobile unit consists of a mobile hardware subsystem and a mobile software subsystem. The hardware sends events to the software, which issues instructions to operate the hardware. For example, when the user makes a call, the event is sent to the software. The software instructs the hardware to contact the base station through a radio frequency channel, called an airlink. The high-power transceivers of the base station receive the request and forward it to controller hardware, which forwards it to the controller software. The controller software validates the call request. If the caller and callee are valid subscribers, then it instructs the hardware and the transceivers to establish a connection.

The ability of UML to support system modeling leads to an extension of UML, that is, the System Modeling Language (SysML). The nine diagrams of SysML and how they relate to UML are summarized in Figure 3.8. The last column of the table

SysML	UML	Description	Remark	Presented in Chapter
Activity diagram	Activity diagram	Model activities that relate to each other via workflows, and exhibit sequencing, exclusion, synchronization, and concurrency relationships	SysML activity diagram extends UML activity diagram.	14
Block definition diagram		Model structural elements called blocks and their composition and classification	Block definition diagram extends UML class diagram.	
Internal block diagram		Model interconnection and interfacing of internal elements of a block	Internal block diagram extends UML composite structure diagram.	
Package diagram	Package diagram	Model the logical organization of modeling artifacts and software artifacts	Same diagrams	
Parametric diagram		Specifies constraints to support engineering analysis		
Requirement diagram		Model text-based requirements and their relationships with other requirements and artifacts such as design elements, test cases, etc.		
Sequence diagram	Sequence diagram	Model time-ordered interaction behavior between objects	Same diagrams	9
State diagram	State diagram	Model state dependent behavior of an object	Same diagrams	13
Use case diagram	Use case diagram	Show the functions or business processes of an application or system as well as relationships of these to external entities called actors	Same diagrams	7

FIGURE 3.8 Relating SysML and UML diagrams

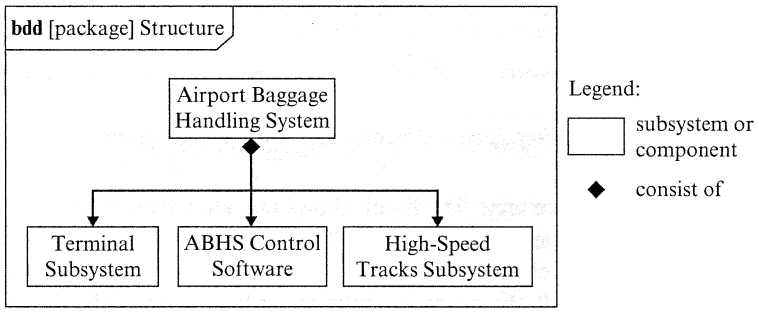


FIGURE 3.9 SysML block definition diagram for the ABHS

and 3.10 show an SysML block definition diagram (bdd) and an SysML internal block diagram (ibd) for the ABHS. These diagrams correspond to the block diagrams displayed in Figures 3.5 and 3.6, respectively.

Sometimes, it is desirable to show material flows in addition to information flows in a model. For example, the ABHS moves luggage from the check-in areas to the departure gates and the arrival gates to baggage claim areas. Such flows are

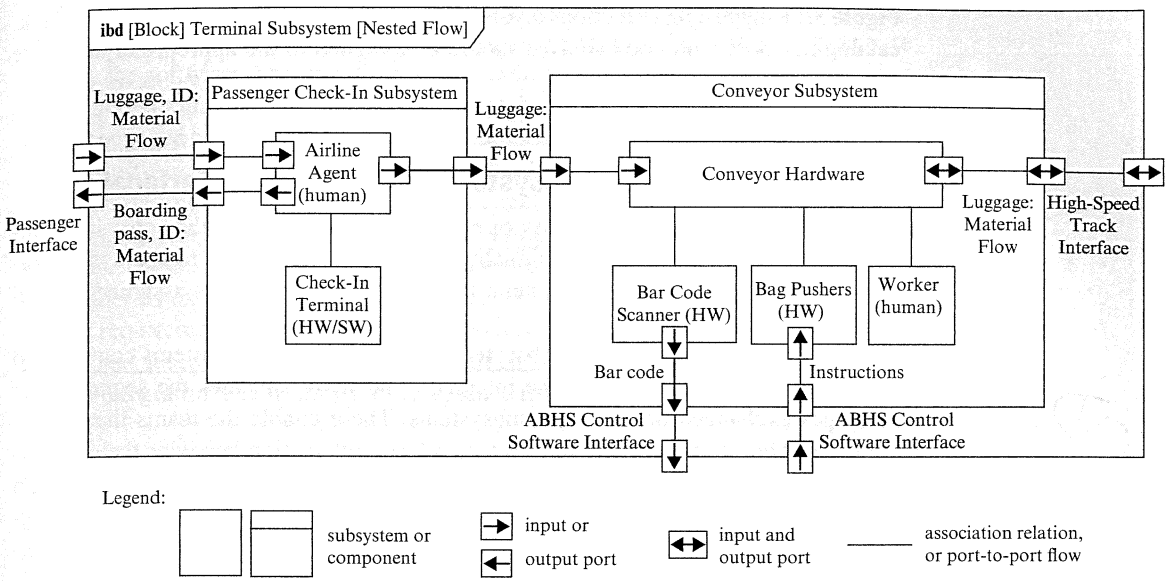


FIGURE 3.10 SysML internal block diagram for ABHS terminal subsystem

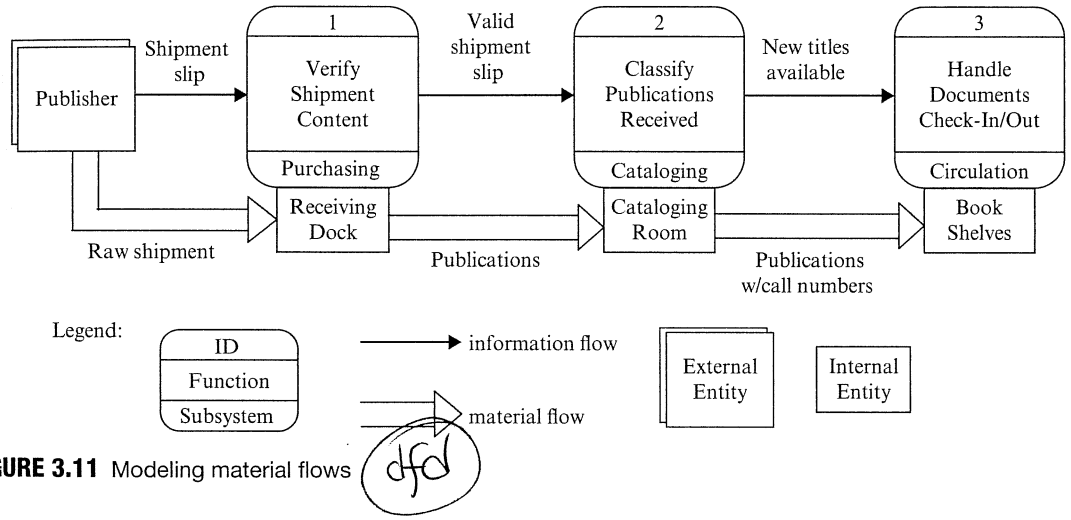


FIGURE 3.11 Modeling material flows

called material flows. In Figures 3.5 and 3.6, material flows and information flows are represented using the same notation. The SysML diagrams shown in Figure 3.9 and 3.10 also use the same notation for information and material flows. To distinguish, material flows should be modeled differently. To illustrate, Figure 3.11 shows a data flow diagram for a library system. The block arrows represent material flows. The ovals represent tasks or processing functions. The entity or subsystem that performs that function is shown in the lower compartment of the oval. The upper compartment of the oval shows the task ID. Material flows are useful in assigning system requirements

Figure 3.11 suggest that functions involving material flows such as the receiving dock, cataloging room, and book shelves should be assigned to the appropriate hardware and human subsystems.

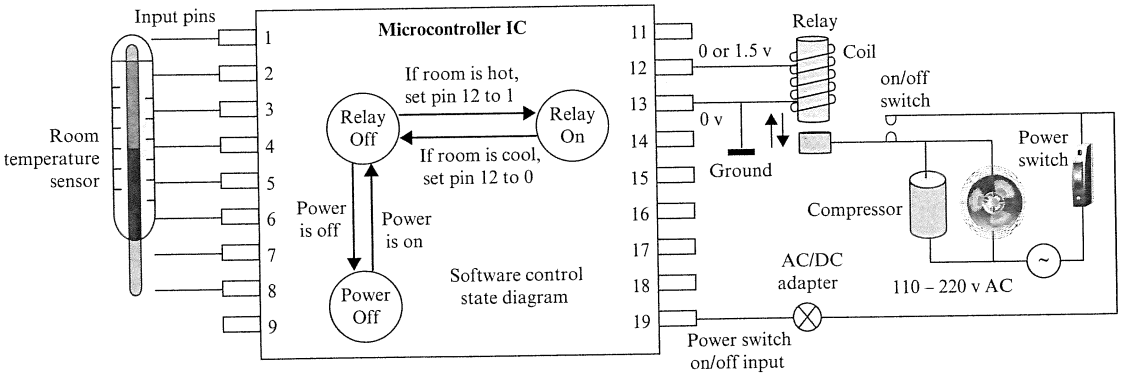
3.4.4 Specification of Subsystem Functions and Interfaces

This step specifies the functionality of each subsystem and how the subsystems interact with each other. The functionality is specified according to the system requirements allocated to the subsystem. It refines the requirements assigned to each subsystem.

The interfaces between the subsystems specify how the subsystems connect and communicate with each other. The interaction behavior specifies the sequences of messages exchanged between the subsystems. These enable the teams that implement the subsystems to know what interfaces and interaction behavior they can expect and need to provide. Hardware-software interfaces are the most common in embedded systems. In many cases, a microcontroller integrated circuit is used. Figure 3.12 illustrates this for an air conditioner. The figure shows only the most relevant items. For example, the unused pins should, in fact, connect to other electronic components.

ICD
Interface
Control
Document

The state diagram shown in Figure 3.12 represents the behavior of the software running inside the IC chip. The software enters into the Relay Off state when the power switch is turned on. The transition from the Power Off state to the Relay Off state indicates this. The software periodically reads the byte representing the eight pins that connect to the temperature sensor to obtain the room temperature. If the room is hot, the software sets pin 12 to 1. This applies a 1.5 volt direct current (DC) to the coil of the relay. The coil generates a magnetic field, which attracts the on/off switch to close the power circuit of the compressor and fan. Thus, cooling begins. When the room becomes cool, the software sets pin 12 to 0. This opens the power circuit of the compressor and fan. Thus, cooling stops.



In the case shown in Figure 3.12, the interface and interaction behavior specification defines the pins that represent the room temperature, the meaning of the byte read by the software. The specification also defines the output pins and the meaning of the output values. In this case, the output pin is pin 12. When the pin is set, the relay is engaged and the cooling begins. Besides software-hardware interface, there are human-software, software-software, human-hardware, human-human, and hardware-hardware interfaces. These interfaces and interaction behavior are specified similarly.

System design also produces a system integration test plan and an acceptance test plan. These specify test procedures to ensure that the subsystems will work with each other as expected, and the system delivers the capabilities as specified in the system requirements specification. Required resources to conduct system integration testing and acceptance testing are also specified. These test plans are used during the system integration and testing, and acceptance testing phases.

3.5 SUBSYSTEMS DEVELOPMENT

After system design and allocation, the subsystems are assigned to different engineering teams. The engineering teams construct and test the subsystems separately. In particular, the software engineering (SE) team applies software engineering processes and methodologies to develop the software subsystem. The team also performs quality assurance and management activities. The engineering teams maintain close contact, exchange progress status, and collaborate with each other to solve interdisciplinary design and implementation problems. This section presents activities that are relevant to the software engineering team.

3.5.1 Object-Oriented Context Diagram

The software development activity needs to consider the context of the software subsystem. The context consists of real-world objects and other subsystems that interact with the software subsystem. Object-oriented software engineering models the world and the software system as consisting of objects that interact with other. For example, the context of the air-conditioning software shown in Figure 3.12 includes the room temperature sensor, the AC/DC adapter, and the relay. The context of the ABHS control software displayed in Figure 3.5 includes the terminal subsystem, the high-speed track subsystem and the flight information subsystem. Figure 3.6 shows that the software interacts with the bar code scanners and bag pushers, which are components of the terminal subsystem. Object-oriented software development models these as objects that interact with the software under development. The result is displayed in a UML class diagram called a context domain model. It is also called a context diagram. The context domain model shows the context objects, properties of the context objects, and relationships between the context objects and the software.

To illustrate, Figure 3.13 shows a context diagram for the ABHS control software. The diagram specifies the types of objects that interact with the ABHS control software. It also specifies the interfaces and interaction protocols with these objects. The

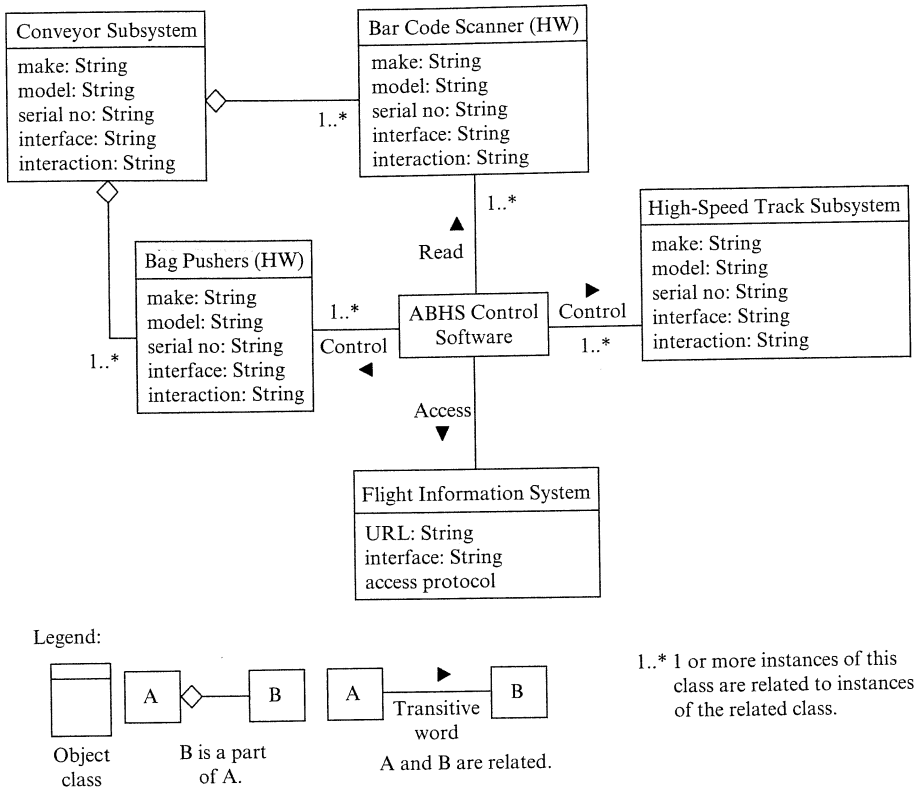


FIGURE 3.13 A context domain model for ABHS control software

information contained in the context diagram is useful for the design of the software subsystem.

3.5.2 Usefulness of an Object-Oriented Context Diagram

The advantages of an object-oriented context diagram are as follows:

1. *It provides a unified view of the software objects and the context objects.* This helps the team understand the context objects and their relationships to the software subsystem.
2. *It highlights the interfaces and interaction with the context objects.* The context domain model treats the software as a black box, and shows only the context objects, their relationships, and attributes that are useful to the design of the software subsystem.
3. *It helps the development of the software subsystem.* For example, the attributes of the bar code scanners are useful for the design and implementation of the classes that interface with the bar code scanners. The multiplicity, such as "1..*"

support. The information shown in the context diagram is also useful for testing and maintenance. For example, when adding a new type of bar code scanner only the classes that interface with the new scanner need to be designed and implemented.

4. *It facilitates the communication and collaboration with other engineering teams.* The object-oriented context diagram highlights the interfaces and interaction with other engineering subsystems. This helps the software engineering team to focus on the interface and interaction issues when working with the other engineering teams.
5. *It is useful for training.* A software subsystem of a large complex system needs to interact with many other complex systems. It is not easy to understand the context. The context diagram is a useful tool for new members to learn the context of a software subsystem.

3.5.3 Collaboration of Engineering Teams

During the subsystems development process, the engineering teams maintain close contact to exchange status and solve interdisciplinary problems. Consider, for example, the design and prototyping of a radio transceiver subsystem. The electronic engineering (EE) team may discover that a certain feature should be implemented with software rather than hardware. The EE team would meet with the SE team to assess the feasibility. A change proposal is submitted if the two teams agree on the change. A change control board reviews the proposal and either approves or disapproves the change. If the proposal is approved, the change is implemented.

3.6 SYSTEM INTEGRATION, TESTING, AND DEPLOYMENT

The subsystems developed by the various engineering teams are then integrated, and integration and system testings are performed. System integration testing is conducted according to the system integration test plan produced in the system design and allocation phase. Due to changes during the subsystems development phase, modification to the test plan may be required. System integration testing refines the test procedures specified in the test plan and executes the tests to ensure that the subsystems communicate properly through the subsystem interfaces.

System testing is performed according to the acceptance test plan. It ensures that the system in fact can deliver all the capabilities stated in the system requirements. Depending on the system, integration and system testings may require special equipment. For example, test equipment for telecommunication systems can generate millions of simulated calls to test the system under development, and accurately measure the throughput, performance, and response times of the system. After system testing, the system is shipped and installed in the target environment. The system is tested by the users. This is called beta testing. Defects found during beta testing are reported to the development team. The defects are removed and the system retested. After beta testing, the system enters into the maintenance phase. During maintenance,

3.7 SYSTEM CONFIGURATION MANAGEMENT

System engineering involves multiple engineering teams. The teams must work in a coordinated and controlled manner. Suppose that the design of a software component depends on the design of an electronic component. In this case, the design of the software component cannot be finalized until the design of the electronic component is finalized. The question is: how is the software engineering team notified when the electronic component design is finalized? Sending an email is an option, but this informal approach is problematic for a large system development project. Who should be responsible for sending the email? Who should receive the email? What if the email is not sent, or does not reach the receiver? What should be written in the email? A formal means to notify the engineering teams of such events is needed.

Change control is another issue that must be coordinated. For example, if the design of the electronic component is changed, then the design of the software component must change as well. This means that the software engineering team should be notified. A notification mechanism is needed. If the design of the electronic component could be changed frequently and arbitrarily, then the software engineering team would feel difficult to cope with the changes. On the other hand, if the design cannot be changed, then how can the electronic team correct design flaws or design errors? System configuration management solves these problems.

System configuration management is based on the concept of a baseline. A baseline denotes an important stage of the project. For example, a system development project consists of system requirements baseline, system design baseline, allocation baseline, subsystem design baselines, and more. The system requirements baseline signifies that the system requirements analysis phase is successfully completed. A baseline is associated with a set of artifacts called configuration items. For example, the system requirements baseline may include the system requirements specification and the project plan. When these are produced and reviewed, and the defects found by the reviewers are removed, the system requirements baseline can be established. Before establishing the system requirements baseline, these artifacts can be modified freely. Once the baseline is established, changes to these artifacts must go through a change control procedure. That is, the proposed changes must be evaluated by a change control board representing the stakeholders. If the proposed changes are accepted, then they are implemented and the progress status is monitored. If the proposed changes are rejected, then they are archived. In summary, system configuration management consists of four main functions:

1. *Configuration item identification.* This function defines the project baselines and associated configuration items. Some examples of baselines are the system requirements baseline, system design baseline, allocation baseline, and subsystem design baselines. Example configuration items include system requirements specification, project plan, system acceptance test plan, system design specification, allocation plan, system integration test plan, and more. A configuration item can be changed freely before the baseline is established. A baseline can be established when all the associated configuration items are checked into the configuration

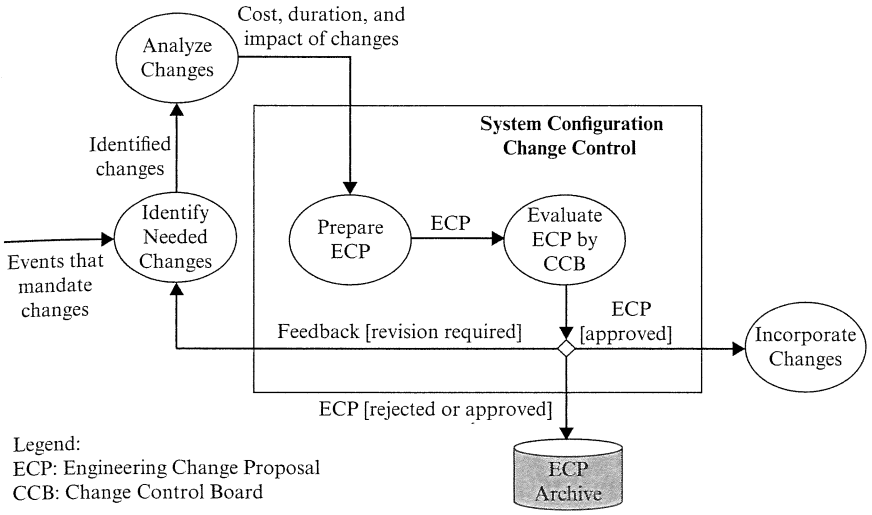


FIGURE 3.14 Configuration change control

of the system requirements specification and the project plan, then the baseline can be established when both of these documents are checked into the configuration management system. Once the baseline is established, changes to the configuration item need to go through a change control procedure.

2. *Configuration change control.* This function defines the change control procedure and executes the procedure. Figure 3.14 illustrates the change control procedure. First, the needed change is identified and analyzed. Next, an engineering change proposal (ECP) is prepared. It specifies the change, its impact, duration, and cost. The ECP is evaluated by a change control board (CCB). The CCB consists of representatives of parties that may be affected by the proposed change. The evaluation either accepts or rejects the ECP. As a result, the proposed change is either implemented or archived.
3. *Configuration auditing.* This function formally establishes the project baselines. It also ensures that the proposed engineering changes are made properly. To achieve these goals, configuration item verification and validation are performed. Configuration item verification ensures that the required configuration items are produced when a baseline is established. For example, it ensures that the system requirements specification and the project plan are produced prior to establishing the system requirements baseline. Configuration item validation ensures that the configuration items are correct, for example, the system requirements specification is reviewed by the engineering teams, domain experts, customer, and users, identified defects are removed and concerns are addressed.
4. *Configuration status reporting.* This function provides database support to the other three functions. The database can be queried for information about the configuration items. The function also publishes events about the system configuration. These include the establishment of a baseline and changes to the

SUMMARY

This chapter presents system development activities that involve hardware, software, and human components. It describes the system engineering process and techniques for system modeling, design, and system requirements allocation. After allocation, the subsystems are developed by separate teams of various engineering disciplines. The subsystems are then integrated and tested to ensure that the system satisfies the system requirements and constraints. During the entire system development process, numerous analysis, design, implementation, and testing documents are produced and updated. Updating one document may affect many other documents. The updates must be coordinated, which is done by system configuration management.

FURTHER READING

Excellent presentations of various aspects of system engineering are given by Blanchard [31] and Kossiakoff et al. [99]. SysML is presented in [65] and UML in [36]. In [152], Thayer presents how to apply system engineering principles to developing large, complex software systems. The paper integrates IEEE software engineering standards and processes into the software system engineering process.

CHAPTER REVIEW QUESTIONS

1. What is system engineering?
2. Why do we need system engineering?
3. Why is system engineering considered an interdisciplinary approach?
4. What are the phases and activities of the system engineering process described in this chapter?
5. What are the purposes of hardware development, software development, and human resource development, and why are they performed separately?
6. How does the system engineering process tackle the system development challenges?
7. What is the relationship between system engineering and software engineering?
8. What factors should the system engineering team consider during system design?
9. What are the applicable strategies for decomposing a system or subsystem during system design?
10. What is system modeling?
11. What is a requirement-subsystem traceability matrix?
12. What are the potential challenges of system integration and testing?

EXERCISES

3.1 Provide a brief description of the functions of a vending machine. Identify and formulate all functional and performance requirements.

3.2 Identify two embedded systems not mentioned in this chapter. For each of these systems, perform the following:

a. Briefly describe the functions of the system. The description should be about a half of a page to one page including diagrams, if any.

b. Identify and formulate five functional requirements and two nonfunctional requirements. One of the nonfunctional requirements must be a performance requirement.

c. Decompose the system and allocate the system requirements to the subsystems.

3.3 A coin-operated car wash system is a self-service car wash system. A customer inserts the required number of quarters to buy a preset period of wash time. The

customer can turn a dial to select soap, foam, rinse, and wax any time during the wash period. The system beeps when one minute is remaining. The customer can insert more quarters to prolong the wash period. Perform the following for the car wash system:

a. Identify and specify the system requirements including functional and nonfunctional requirements.

b. Decompose the system into functional subsystems. Decompose the system requirements if necessary.

c. Allocate the system requirements to the subsystems.

d. Construct a system architectural design diagram using a diagramming technique of your choice or as designated by the instructor.

3.4 A railroad crossing system employs sensors, flashing lights, sounding bells, and gates to control the traffic at a railroad intersection. When a train approaches the intersection from either direction, a sensor device senses the train and communicates with the software. The software turns on the flashing yellow warning light and the sounding bell for a given period. It then changes the light to flashing red and closes the gates. After the train has left the intersection, another sensor device detects this and communicates the event to the software. The software turns off the lights and the sounding bell and opens the gates. Perform the following for the railroad crossing system:

a. Identify the hardware and software subsystems and specify the functionality of each of the subsystems.

b. Describe how the subsystems relate to and interact with each other. That is, specify the subsystem interfaces and interaction behavior.

c. Construct a system architectural design diagram to show the relationships between the subsystems.

d. Identify and formulate safety requirements and allocate them to the subsystems. Describe how the subsystems will satisfy the safety requirements.

3.5 Managers of a department store want to expand into online retailing. This means that the company needs to develop an online system that can take orders online, and ship the ordered items through a designated national shipment carrier. To reduce labor costs, the company wants a fully automated system. Perform the following for this system:

a. Identify and formulate five functional requirements and two performance requirements for the system.

b. Decompose the system into a hierarchy of subsystems and allocate the system requirements to the subsystems.

c. Produce a system architectural design diagram.

3.6 If you have learned UML class diagramming, then produce an object-oriented context diagram for the software subsystem of the online retailing system you produced in exercise 3.5. Discuss the usefulness of the context diagram for the design, implementation, testing, and maintenance of the software subsystem.