Chapter 11

# Integration and Qualification

## 11.1 INTRODUCTION

*Integration* is the process of assembling the system from its components, which must be assembled from their configuration items (CIs). *Qualification* is the process of verifying and validating the system design and then obtaining the stakeholders' acceptance of the design. Recall that verification is the determination that the system was built right while validation determines that the right system was built. Both of these activities are conducted by the systems engineering team as part of the development process, primarily during integration. Validation has critical early elements (conceptual, design requirements, and validity) that are completed during the design phase. The system that is used to qualify the system being designed must be built for that purpose. So while the operational system is being designed, the qualification system for the operational system is also being designed and integrated. The operational phase for this qualification system is during integration and qualification. Also keep in mind that other systems are being developed concurrently with the operational system, namely, some or all of the manufacturing, deployment, training, refinement, and retirement systems. Each of these also has a qualification system.

The terms *testing* and *qualification* are used interchangeably in parts of this chapter. The word *testing* is associated with the key words of *acceptance, validation, and verification* by most systems engineers. However, the process of acceptance, validation, and verification comprise what is being called qualification in this chapter. The confusing usage arises when an instrumented test is mentioned as one of four methods that comprise qualification (testing), and the other three methods do not contain the word test: inspection, demonstration, and analysis and simulation. In fact, these three methods are forms of test. The word *qualification* is used in this chapter as often as possible to mean the process that comprises acceptance, validation, and verification testing. The word *testing* will be used with these three terms but is meant to be associated with the methods used in the qualification process during integration.

This chapter begins by providing a detailed definition of the elements of qualification: acceptance testing, validation, and verification. Section 11.3 discusses the concept of integration since qualification takes place as integration is progressing; alternate processes for integration are discussed in Section 11.4. Then qualification is described in detail, beginning with planning and proceeding to a detailed discussion of qualification methods. Special topics in acceptance testing are described in Section 11.7.

The *exit criterion* for integration and qualification is acceptance of the design by the stakeholders. This is often done conditionally, that is, with the provision that certain system elements be revised to enable greater cost-effectiveness during operation.

## 11.2 DISTINCTIONS AMONG ACCEPTANCE, VALIDATION AND VERIFICATION TESTING

In Chapter 1 the concepts of verification, validation, and acceptance were introduced. (Grady [1997] provides additional detail on the distinctions being discussed here.) *Acceptance* is a stakeholder function for agreeing that the designed system, as tested or otherwise evaluated by the stakeholders, is acceptable. As such acceptance is driven by the stakeholders, with the knowledge of the results of validation and verification activities that have preceded it. See Figure 11.1.

*Validation* is the process of determining that the systems engineering process has produced the *right system*, based upon the needs expressed by the stakeholder. Validation is carried out by the systems engineers, based upon what they believe the stakeholders' needs to be. The most reliable and early statement of the stakeholders' needs is the operational concept. Therefore *operational validity* is the matching of the capabilities of the designed system to the operational concept; this naturally occurs late in the integration phase after the designed system has been verified. However, conceptual validity, requirements validity, and design validity are important aspects of validity and need to be addressed early in the design phase. Conceptual, requirements, and design validity are called *early validation*, the determination that the right problem is being defined at the current level of abstraction, given the validity of the problem definition at a higher level of abstraction.

*Conceptual validity* is the correspondence between the stakeholders' needs and the operational concept. Conceptual validity needs to be established at the outset of the design process via interactions among the systems engineers and the stakeholders; however, the systems engineer cannot assume that once

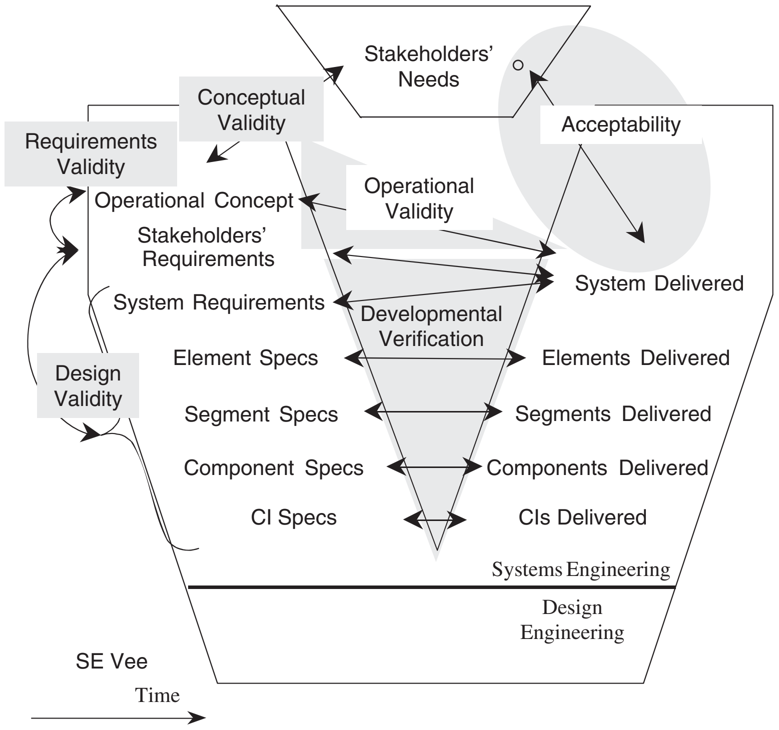

**FIGURE 11.1** Verification, validation, and acceptance.

established there is no more work to be done. Stakeholders' needs change and the operational concept must change with those needs. Note operational validity only makes sense if conceptual validity has been established. If both conceptual and operational validity are solid, then the stakeholders' acceptance should be nearly guaranteed.

*Requirements validity* is the correspondence between the operational concept and the stakeholders' requirements. In requirements validity the operational concept is assumed to be an accurate reflection of the stakeholders' needs; the validation occurs by establishing that the stakeholders' requirements have neither introduced new issues nor left issues out of the operational concept, thus causing the design of a different system than envisioned in the operational concept. But recall that the operational concept and stakeholders' requirements should be stated in design independent terms, making this task of requirements validity quite difficult. Elements of requirements validity are ensuring there are input/output *requirements* for all of the inputs and outputs *in* the operational concept; that every objective in the objectives hierarchy has a performance requirement in the StkhldrsRD; that every external interface to the system has been considered for an external interface requirement; and so forth.

The external systems diagram and objectives hierarchy (discussed in Chapter 6) are key tools for establishing this requirements validity. In addition, intermediate products such as a data model that relates the inputs to and outputs from the system in the operational concept to the aggregate inputs and outputs of the system in the external systems diagram can and should be developed to support requirements validation. At a higher level of abstraction, the systems engineers should be asking "Can we get something we do not want even though these requirements stating our needs are met?" In addition they should ask "Can we get what we want (the problem solved) without getting what we have asked for in the requirements?" If either of these questions can be answered positively, there is more work to do on the requirements.

*Design validity* assumes that the Stakeholders' Requirements Document (StkhldrsRD) is a valid statement of the stakeholders' needs and addresses the congruence between the StkhldrsRD and the derived requirements. The derived requirements begin with the Systems Requirements Document (SysRD), evolve to subsystem and component specifications, and culminate in CI specifications. In Chapter 9 three techniques for flowdown or derivation of requirements were discussed: apportionment, equivalence, and synthesis. Establishing design validity for apportionment and equivalence is straightforward. Design validation when synthesis is involved, on the other hand, requires establishing the validity of the models used to complete flowdown via synthesis. These models are used to transform requirements on one or more variables to requirements on parameters that have a functional relationship with these variables. A common cause for failure in this synthesis process is that the models being used were valid in previous engineering efforts but are not valid for the current system; yet the validity of the models from previous developments of similar systems is assumed to pertain to the current development. Petroski [1994] provides extensive evidence of such failures in structural design engineering; failures of bridges are highlighted in particular. The designers forgot the lessons of past failure modes and built bridges that were extrapolations of previous efforts: Extrapolations that were not justified based upon modeling assumptions that were not examined in sufficient detail.

Conceptual requirements and design validity are the province of the systems engineering team and must be undertaken very seriously to ensure that the requirements development process does not redefine the problem being solved. There are two chains that must be strong; see Figure 11.2. The first chain consists of conceptual validity, operational validity, and acceptance testing. Requirements validity, design validity, verification, and operational validity comprise the second chain. Each of these chains is only as strong as the weakest link.

*Verification* is the matching of CIs, components, subsystems, and the system to their corresponding requirements to ensure that each has been *built right*. This process of design verification is also carried out by the systems engineering team to ensure that the design problem defined in conjunction with the stakeholders is being solved appropriately. In order for verification to be

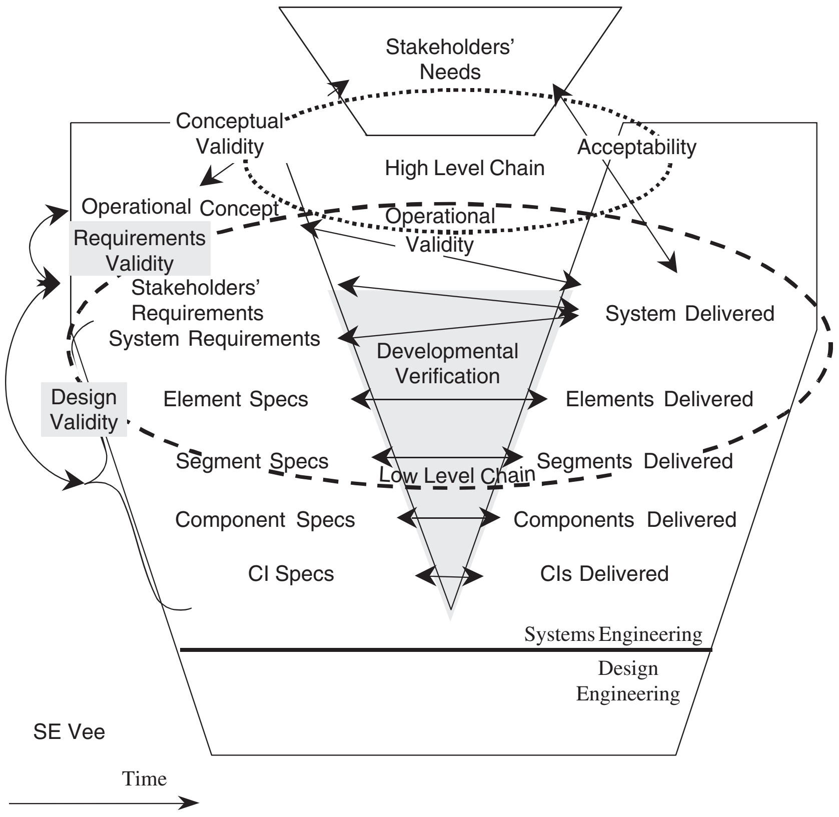

**FIGURE 11.2** Two qualification chains. The high level chain consists of conceptual validity, operational validity, and acceptability. The low level chain consists of design validity, requirements validity, developmental verification, and operational validity.

successful, the originating and derived requirements must be testable; that is, the requirements must be single statements that are unambiguous, understandable, and verifiable (see Chapter 6).

Verification begins in the design phase with the definition of the derived requirements and becomes the focus of activity early in the integration phase when the systems engineers can match the derived requirements to the capabilities of the CIs and the components. However, the design of the test system to achieve this verification must occur in the design phase of the system.

It is a misconception to picture verification as beginning and ending before validation, which begins and ends before acceptance testing. In fact, as can be seen in Figure 11.1, validation has to begin with the definition of requirements to ensure that there is conceptual validity between the operational concept and the stakeholders' needs. Requirements validity also begins almost immediately to address the congruence between the stakeholders' requirements and the operational concept. Finally, design validity addresses the consistency and congruence between stakeholders' requirements and derived requirements. For example, does every input and output to the system have at least one requirement associated with it? Does the system have all of the system-wide requirements it should have? Before operational validation can begin, design of a qualification system must occur. The IDEFO (Integrated Definition for Function Modeling) representation in Figure 11.3 of early validation, verification, operational validation, and acceptance testing suggests the most likely sequential ordering. In practice, though, there is substantial concurrency involving these processes, making the results even more difficult to get right.

Finally, in order for the acceptance test to be successful, there must be clear agreement between the acceptance thresholds and the early design documents of the operational concept and stakeholders' requirements. Therefore, design of the acceptance test must begin early enough to enable both conceptual and design validity.

Successful integration relies critically on the complete and consistent development of stakeholders' requirements, the proper flowdown of stakeholders' requirements into derived requirements and tracing of requirements to functions and components/CIs, and the analysis of system performance and cost in light of the stakeholders' fundamental objectives. These are design activities associated with the system. The development of test requirements, including the verification, validation, and acceptance test plans, initializes integration and helps formalize the design process.

## 11.3 OVERVIEW OF INTEGRATION

Textbook integration is a bottom-up process (see the top half of Figure 11.4) that combines multiple CIs into components, and multiple components into subsystems, and multiple subsystems into the system. At each level of integration the appropriate interfaces and models of the external systems, components, and CIs must exist for this subset of the system. These interfaces and models are stimulated by defined sets of inputs and tested to determine if the appropriate outputs are obtained. In addition, the physical combination of the CIs, components, or subsystems is examined to determine that the fit of these system elements is acceptable. This is not to say that integration can only be bottom up and must wait for the last available CI before proceeding to the component level. In fact, design stubs (shells or model replicas) for specific CIs, components, or even subsystems can be developed as part of the integration process to reduce risk, speed up integration, and enhance the testing effort. Alternate integration processes are discussed later.

Figures 11.4, 11.5 and 11.6 show three different representations of the major integration functions. The bottom half of Figure 11.4 shows this information as an IDEFO diagram with the functions and flow of data among the functions; the major functions are (1) inspect and test the CI (component or subsystem), (2) identify and fix any correctable deficiencies found in the first function, (3)

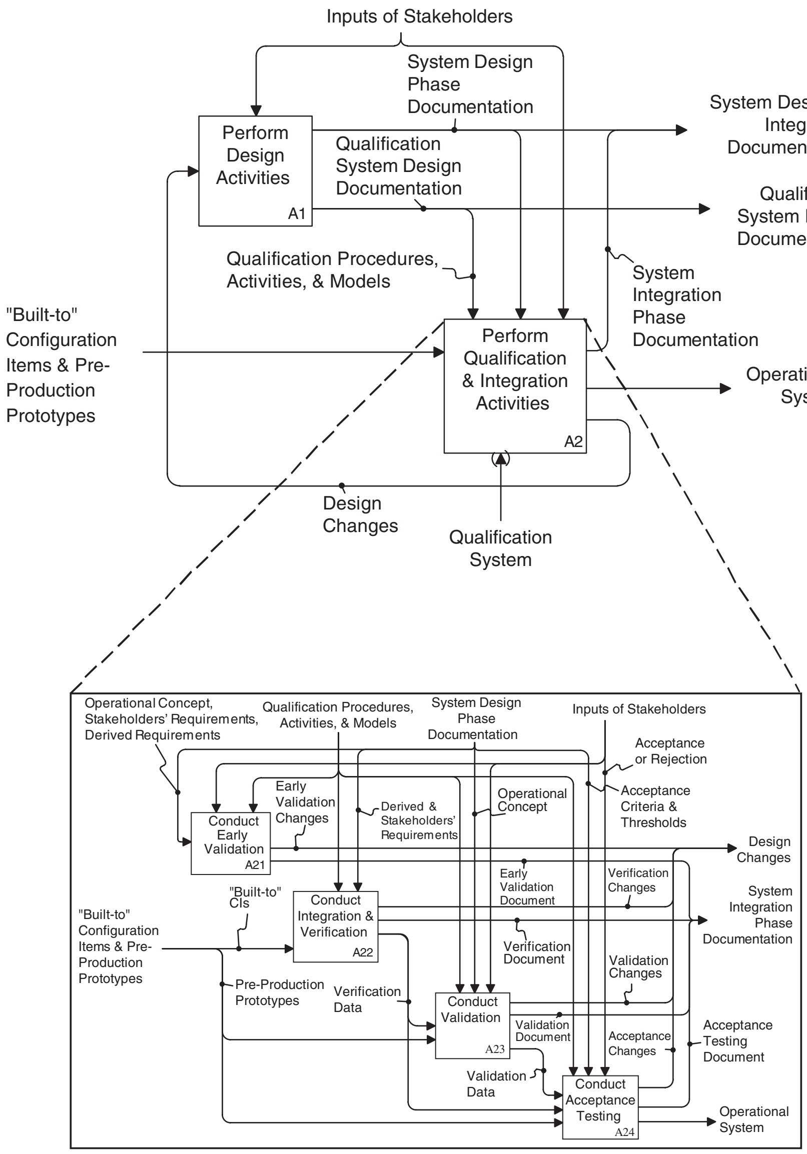
Inputs of Stakeholders
System Design Phase Documentation
System Design & Integration Documentation
Perform Design Activities
A1
Qualification System Design Documentation
Qualification System Design Documentation
Qualification Procedures, Activities, & Models
System Integration Phase Documentation
"Built-to" Configuration Items & Pre-Production Prototypes
Perform Qualification & Integration Activities
A2
Operational System
Design Changes
Qualification System
Operational Concept, Stakeholders' Requirements, Derived Requirements
Qualification Procedures, Activities, & Models
System Design Phase Documentation
Inputs of Stakeholders
Acceptance or Rejection
Early Validation Changes
Derived & Stakeholders' Requirements
Operational Concept
Acceptance Criteria & Thresholds
Conduct Early Validation
A21
Design Changes
Early Validation Document
Verification Changes
"Built-to" CIs
Conduct Integration & Verification
A22
System Integration Phase Documentation
"Built-to" Configuration Items & Pre-Production Prototypes
Verification Document
Validation Changes
Pre-Production Prototypes
Verification Data
Conduct Validation
A23
Validation Document
Acceptance Changes
Acceptance Testing Document
Validation Data
Conduct Acceptance Testing
A24
Operational System

**FIGURE 11.3** Bottom-up integration process.

assess the impact of any uncorrectable deficiencies found in the first function, (4) redesign the CI (component or subsystem) to address unacceptable impacts of any uncorrectable deficiencies as identified in the third function, (5) modify the baseline of the design to account for any fixes (function 2) or acceptable impacts (requirements changes from function 3), and (6) integrate with the next CI (component or subsystem) and repeat until all CIs (components or

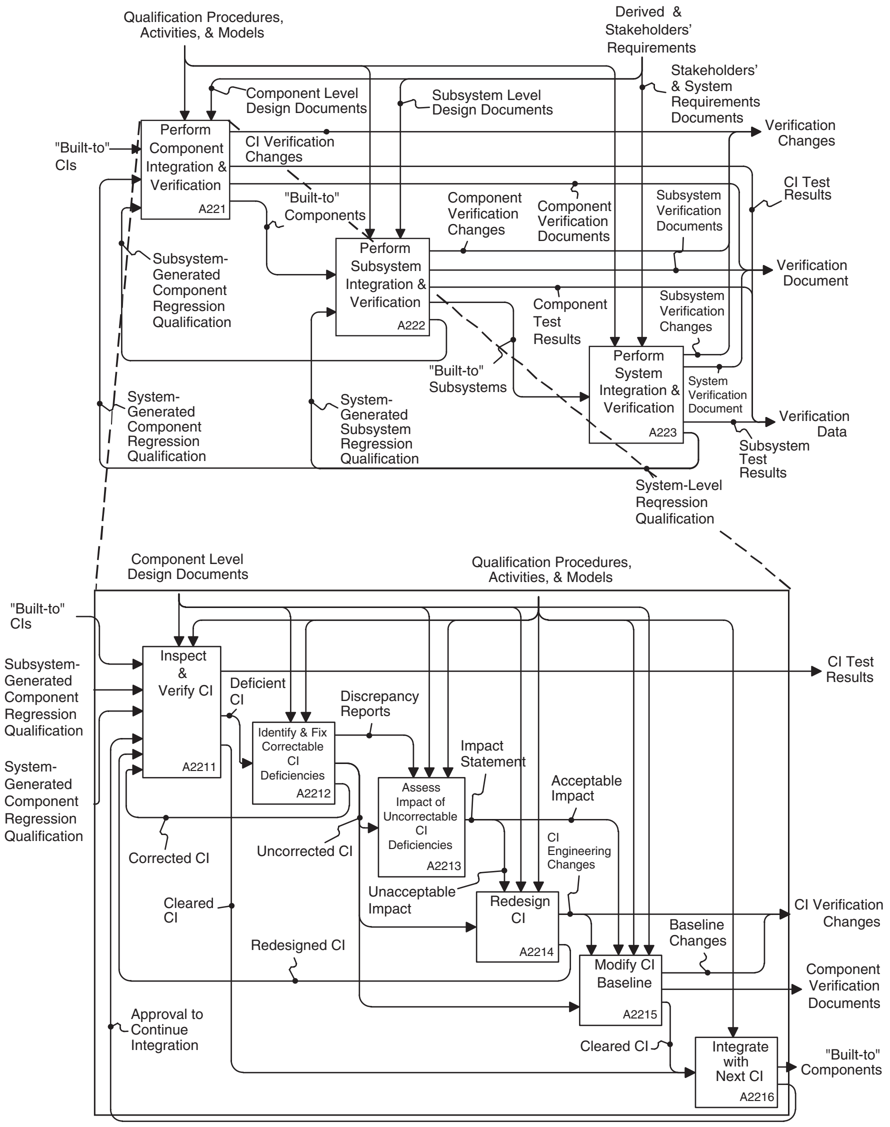
Qualification Procedures, Activities, & Models
Derived & Stakeholders' Requirements
Stakeholders' & System Requirements Documents
Component Level Design Documents
Subsystem Level Design Documents
"Built-to" CIs
Perform Component Integration & Verification
A221
CI Verification Changes
Verification Changes
CI Test Results
"Built-to" Components
Component Verification Changes
Component Verification Documents
Subsystem Verification Documents
Perform Subsystem Integration & Verification
A222
Verification Document
Subsystem-Generated Component Regression Qualification
Component Test Results
Subsystem Verification Changes
"Built-to" Subsystems
Perform System Integration & Verification
A223
System Verification Document
Verification Data
System-Generated Component Regression Qualification
System-Generated Subsystem Regression Qualification
Subsystem Test Results
System-Level Reqression Qualification
Component Level Design Documents
Qualification Procedures, Activities, & Models
"Built-to" CIs
Subsystem-Generated Component Regression Qualification
System-Generated Component Regression Qualification
Inspect & Verify CI
A2211
CI Test Results
Deficient CI
Discrepancy Reports
Identify & Fix Correctable CI Deficiencies
A2212
Impact Statement
Acceptable Impact
Assess Impact of Uncorrectable CI Deficiencies
A2213
Corrected CI
Uncorrected CI
CI Engineering Changes
Unacceptable Impact
Cleared CI
Redesign CI
A2214
CI Verification Changes
Baseline Changes
Redesigned CI
Modify CI Baseline
A2215
Component Verification Documents
Approval to Continue Integration
Cleared CI
Integrate with Next CI
A2216
"Built-to" Components

**FIGURE 11.4** Major integration functions for component integration. (The same six functions apply for subsystem and system integration.)

subsystems) have been integrated. Figure 11.4 addresses component integration but has the identical structure for the higher level integration at the subsystem and system levels.

Figure 11.5 shows logic structure of integration at the subsystem level, that is, integrating every subsystem of the system until all subsystems have been

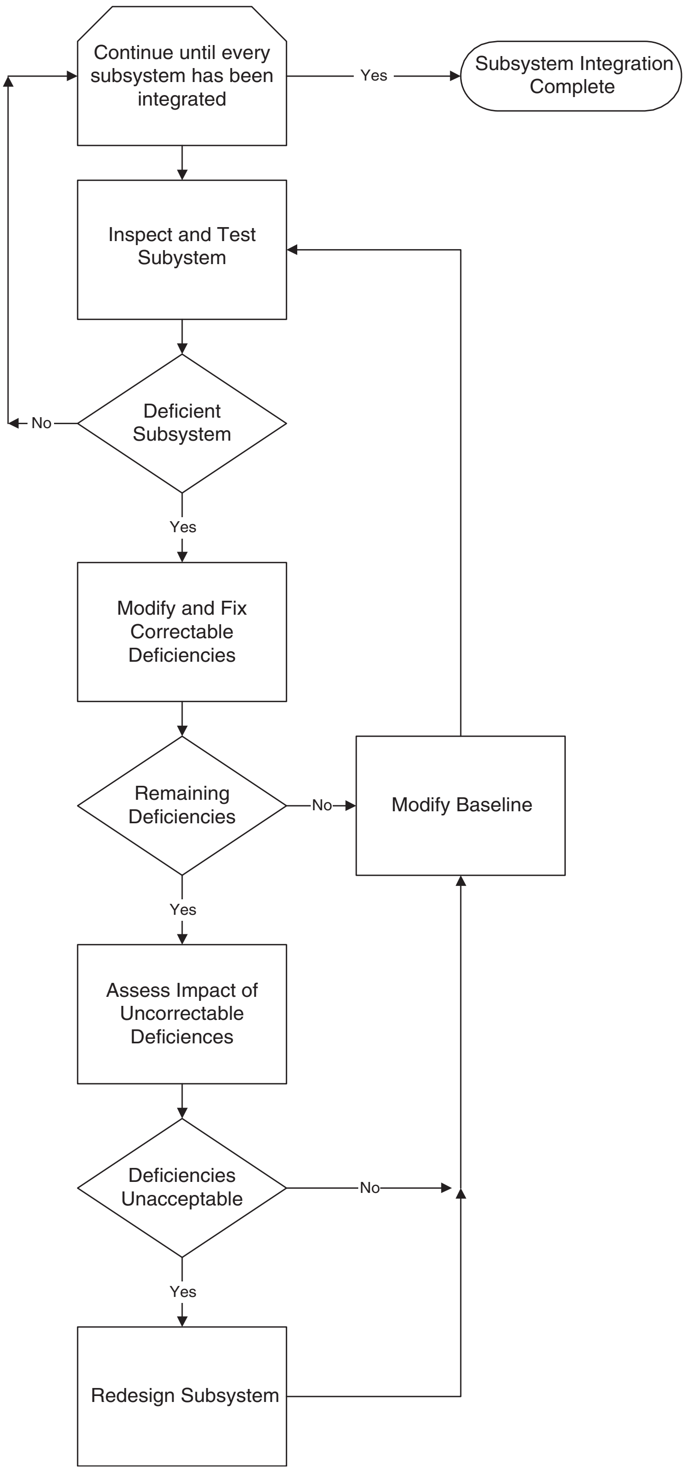
Continue until every subsystem has been integrated
Yes
Subsystem Integration Complete
Inspect and Test Subsystem
No
Deficient Subsystem
Yes
Modify and Fix Correctable Deficiencies
Remaining Deficiencies
No
Modify Baseline
Yes
Assess Impact of Uncorrectable Deficiences
Deficiencies Unacceptable
No
Yes
Redesign Subsystem

**FIGURE 11.5** Logic diagram for subsystem integration.

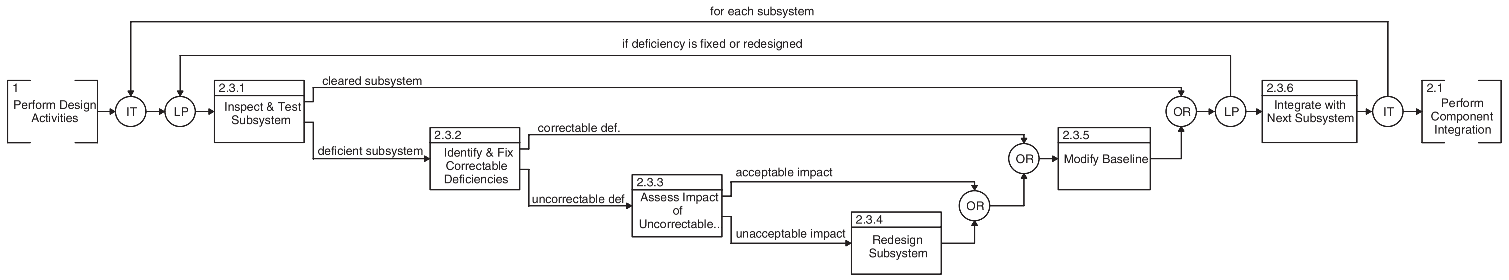
for each subsystem
if deficiency is fixed or redesigned
1 Perform Design Activities
IT
LP
2.3.1 Inspect & Test Subsystem
cleared subsystem
deficient subsystem
2.3.2 Identify & Fix Correctable Deficiencies
correctable def.
uncorrectable def
2.3.3 Assess Impact of Uncorrectable..
acceptable impact
unacceptable impact
2.3.4 Redesign Subsystem
OR
OR
2.3.5 Modify Baseline
OR
LP
2.3.6 Integrate with Next Subsystem
IT
2.1 Perform Component Integration

**FIGURE 11.6** Integration control structure. (Subsystem integration into the System.)

integrated. First a selected subsystem is inspected and tested to determine if it meets the requirements defined in the specification for that subsystem; this is verification. If the subsystem is not deficient, the next subsystem begins the verification process. If the subsystem is deficient, modifications and fixes are made if possible, and the design baseline is modified accordingly. However, if there are remaining deficiencies, the impact of these deficiencies must be assessed. If the deficiencies are acceptable, no redesign is necessary and the requirements baseline is modified. However, if the deficiencies are unacceptable, the subsystem must be redesigned, usually at great cost and delay in time. If any changes are made at all, the subsystem must be retested (called *regression testing*) in case any new problems were introduced.

These six functions cannot flow in serial sequence. In fact, some functions may not be executed at all. If there are no deficiencies, functions 2 through 5 are never executed. If all deficiencies are correctable, functions 3 and 4 are not executed. Figure 11.6 shows the control structure needed to make these function work as a function flow block diagram (FFBD). (The details of reading FFBDs can be found in Chapter 12.) Figure 11.6 shows the functions at the subsystem level of integration, but again this structure applies equally at the component and system levels.

## 11.4 ALTERNATE INTEGRATION PROCESSES

As discussed earlier bottom-up integration is commonly discussed in textbooks as the desired approach. In fact, in Chapter 1 the Vee model of systems engineering represented the bottom-up integration process as the appropriate one. However, there are alternate integration processes (described in Table 11.1) that are appropriate to systems engineering; these alternate approaches have been investigated and described by the software engineering community [Perry, 1988]. The top-down integration process was commonly used in software engineering as part of top-down software design and development. The most commonly used integration process in the software industry [Perry, 1988] is "big bang" integration, in which CIs are combined as they become available and have completed testing.

Top-down integration begins by examining the top-level core of the system, is followed by adding major components to this core and testing, and ends by adding the individual CIs to the cores of the components and testing. Top-down integration is very difficult to accomplish for systems with hardware, people, and facilities that are designed from scratch. It is difficult to define a system core that is hardware, people, and facilities unless a large part of the system already exists, commonly referred to as "commercial off-the-shelf" (COTS) components or CIs. However, as more and more new systems are made up of larger and larger amounts of COTS components, top-down integration has greater usefulness in systems engineering.

**TABLE 11.1 Principal Integration Processes**

| | |
|---|---|
| **Top-Down** | • Integration begins with a major or top-level module.<br>• All modules are called from the top-level module are simulated by “stubs” (shell or model replica).<br>• Once the top-level module is qualified, actual modules replace the stubs until the entire system has been qualified.<br>• This is most useful for systems using large amounts of COTS components.<br>**Phase Integration**: Integration is done from the top down to the lowest level; one peel of the onion at a time.<br>**Incremental Integration**: Integration is done for a specific module from top to bottom; one slice of the system at a time.<br>**Advantage**:<br>Early demonstration of the system is allowed.<br>Representation of the test cases is easier.<br>This is more productive if major flaws occur toward the top of the system.<br>**Disadvantage**:<br>Stubs have to be developed.<br>Representation of test cases in the stubs may be difficult.<br>Observation of test output may be artificial and difficult.<br>This requires a hierarchical system architecture. |
| **Bottom-Up** | • Integration begins with the elementary pieces (or CIs) that comprise the system.<br>• After each CI is tested, components comprising multiple CIs are tested.<br>• This process continues until the entire system is assembled and tested.<br>• This is the traditional systems engineering integration approach.<br>**Phase Integration**: At any point in the integration, all of the subsystems are at the same stage of integration testing.<br>**Incremental Integration**: Integration proceeds one slice of the system at a time.<br>**Advantage**:<br>It is easier to detect flaws in the tiniest pieces of the system.<br>Test conditions are easier to create.<br>Observation of the test results is easier.<br>**Disadvantage**:<br>“Scaffold” systems must be produced to support the pieces as they are integrated.<br>System’s control structure cannot be tested until the end.<br>Major errors in the system design are typically not caught until the end.<br>System does not exist until the last integration test is completed.<br>This requires a hierarchical system architecture. |
| **Big Bang** | • Untested CIs are assembled and the combination is tested.<br>• This is a commonly used and maligned approach. |

(*Continued*)

**TABLE 11.1. Continued**

| | |
|---|---|
| | **Advantage**: |
| | Immediate feedback on the status of system elements is provided. |
| | Little or no pre-test planning is required. |
| | Little or no training is required. |
| | **Disadvantage**: |
| | Source of errors is difficult to trace. |
| | Many errors are never detected. |

Both the bottom-up and top-down integration processes can proceed for the entire system by adding or peeling a layer of the system as one would an onion; this is referred to as phase integration. For bottom-up integration this means that all of the CIs are integrated into their respective components before any components are integrated. However, it is commonly counterproductive from schedule and cost perspectives to delay the integration of some of the components until all of the CIs are ready.

At the other extreme is incremental integration in which one subsystem at a time is integrated from the CIs up through its components before the integration of any other subsystem is begun. Just as phase integration is impractical, so to is pure incremental integration. A major element of test planning is the creation of a realistic schedule for when each CI will be ready so that integration can proceed at an orderly pace and test system devices and models can be ready when needed. This typically involves a mixture of phase and incremental integration.

Finally, big-bang integration is a relatively undisciplined, but much used, approach to integration. At the worst extreme this approach begins assembling CIs as they become available and undertakes testing as an afterthought. Since there is no serious planning for testing sequences, fault detection and fault localization and diagnosis become very difficult. At its best this approach combines bottom-up and top-down integration in a disciplined and rigorous manner. When done well, this approach often takes more planning and development of test rigs but can be accomplished more quickly.

Another major element of the development of the qualification system and qualification planning is the creation of the appropriate test stubs and scaffolds with drivers for the relevant qualification scenarios. Each CI, component, and the system as a whole must be stimulated by a given set of inputs for each qualification case. In addition, test equipment must be put in place to capture the outputs of these CIs, components, and the system. The qualification plan ensures that these qualification system elements will be in place at the right time to enable the planned integration sequence of CIs and components. The plan typically breaks down when planned tests are failed by specific CIs, components, or the system. A well-designed qualification plan will address schedule adjustments for possible qualification failures as part of risk mitigation.

## 11.5 SOME QUALIFICATION TERMINOLOGY

The purpose of qualification is not only to find faults and failures but also to prevent them and to provide comprehensible diagnoses about their location and cause. Recall the following definitions from Chapter 7:

**Failure:** deviation in behavior between the system and its requirements. Since the system does not maintain a copy of its requirements, a failure is not observable by the system.

**Error:** a subset of the system state, which may lead to a failure. The system can monitor its own state, so errors are observable in principle. Failures are inferred when errors are observed. Since a system is usually not able to monitor its entire state continuously, not all errors are observable. As a result, not all failures are going to be detected (inferred).

**Fault:** defects in the system that can cause an error. Faults can be permanent (e.g., a failure of system component that requires replacement) or temporary due to either an internal malfunction or an external transient. Temporary faults may not cause a sufficiently noticeable error or may cause a permanent fault in addition to a temporary error.

The qualification designer should realize that the design of the qualification system is not only important in terms of finding and defining faults and errors but also in guiding designers to preclude them from introducing faults in the first place. In addition, the qualification designer must realize that no qualification procedure is perfect. As Glegg [1981] points out, no procedure can answer all questions of interest. Some procedures do well at capturing what happened; others do much better at explaining why these things happened. As a result a number of complementary procedures must be employed for success. When complete the qualification design must document the qualification procedures in detail and the expected qualification results (requirements) for each procedure. In fact, recall that the qualification process is being conducted by a qualification system; the qualification design should be tested just as any system would be.

To design the qualification system, some basic knowledge of faults is needed and some modeling of fault importance should be completed. The software community [Beizer, 1990] has written much more extensively on these topics than has the systems engineering community. Beizer [1990] presents three laws of software testing that are directly relevant to systems:

*First Law: The Pesticide Paradox*—Every method you use to prevent or find bugs leaves a residue of subtler bugs against which those methods are ineffectual.

*Corollary to the First Law*—Test suites wear out.

*Second Law: The Complexity Barrier*—Software complexity (and therefore that of bugs) grows to the limits of our ability to manage that complexity.

*Third Law*—Code migrates to data.

For systems, replace the word bug with fault. The third law becomes "hardware and people migrate to software which eventually migrate to data." Theoretically Manna and Waldinger [1978 p. 208] summarized the barriers to verification (the easy part of qualification) as:

- "We can never be sure that the specifications are correct."
- "No verification system can verify every correct program."
- "We can never be certain that a verification system is correct."

These barriers generalize to validation.

Beizer [1990] also provides a taxonomy of bug (fault) consequences:

*Mild*: The symptom offends us aesthetically, for example, misspelling or poor formatting.

*Moderate*: Outputs are misleading or redundant, affecting system performance.

*Annoying*: The system's behavior is dehumanizing, for example, names are truncated, bills for $0.00 are sent, operators must resort to unnatural command actions to obtain the desired response.

*Disturbing*: The system refuses to handle legitimate functions.

*Serious*: The system loses track of functions and gobbles unique inputs, for example, your deposit is lost.

*Very Serious*: The system mixes input and output streams, for example, your deposit is credited to another account.

*Extreme*: The problems are not limited to a few situations but occur on a frequent basis.

*Intolerable*: The system causes long-term, unrecoverable corruption of the database and this corruption is not easily detected.

*Catastrophic*: The system decides on its own to shut down, causing unrecoverable corruption of the database.

*Infectious*: The system completes its own functions, but in so doing it corrupts the functioning of other systems.

This type of fault categorization is the first step in defining the importance of faults; these categories define distinctions among the consequences of faults. The other key element of fault importance is the frequency with which the fault occurs. (Note Beizer's extreme category is a variation of very serious that increases the frequency. In a taxonomy on consequences, extreme should be removed.) Consider the set of scenarios ($j = 1, 2, \ldots, J$) in the operational concept (or preferably some aggregation of these scenarios). Develop the following two metrics for each scenario and each fault category ($i = 1, 2, \ldots, I$):

$p_{ij}$ = probability of fault $i$ in scenario $j$;

$c_{ij}$ = dollar (or some other value measure) consequence of fault $i$ in scenario $j$.

The measure of the importance of the fault types $I_i$ is:

$$I_i = \sum_{j=1}^{J} V_j p_{ij} c_{ij}$$

where $V_j$ is the relative measure of the importance of each scenario. (Note, if $c_{ij}$ is in dollars, the term $V_i$ can be set to 1.0; however, if $c_{ij}$ is in non-dollar units, $V_i$ will be needed to calibrate across scenarios.) This measure works well if the likelihood of each fault type in each scenario is relatively rare. If some fault types may occur multiple times in a scenario, then a more complex measure should be used.

Bezier [1990] also presents a taxonomy of "bugs" (software faults) for software programs based upon the cause or source of introduction of the bug. This taxonomy includes requirements, features and functionality, structure, data, implementation and coding, integration, system and software architecture, and testing. Beizer [1990] provides detailed summary statistics for the frequency of these types of bugs.

## 11.6 DEFINING THE QUALIFICATION SYSTEM

There are four major levels of qualification planning: Plan the qualification process, plan the qualification approaches, plan qualification activities, and plan specific tests. The first three qualification planning functions are conducted for verification, validation, and acceptance testing. The fourth planning function is conducted for every specific qualification activity identified in the three prior planning functions. These final plans should stipulate that every requirement be tested individually. Table 11.2 shows the elements of each of the four qualification planning functions. Recent research has been conducted in this area by Meisenzahl et al. [2006], Levardy et al. [2004], and Hoppe et al. [2003].

The system's objectives discussed in Chapter 6 become key for the initial activity of planning the qualification process. These objectives of the system drive the qualification objectives. A key part of the qualification objectives is determining whether the test was passed by the system design or not. Defining the threshold for passing the test is a difficult balancing act; the threshold cannot be too low or there is no reason to conduct the test. At the same time the threshold cannot be too high or there is too great a chance that development money will be wasted fixing deficiencies that were not worth fixing and delaying the production and delivery of a system that is badly needed by the stakeholders, especially when competitive advantage is involved. The qualification objectives must be focused on determining whether the system passes or fails the threshold criteria. This focus on qualification objectives and pass/fail thresholds is the identification of alternate concepts for the qualification

**TABLE 11.2 Qualification Planning Functions**

| | |
|---|---|
| Plan the qualification process<br>Acceptance test<br>Validation test<br>Verification test | • Review system objectives<br>• Identify qualification system objectives<br>• Identify pass/fail thresholds<br>• Define qualification operational concept<br>• Define qualification requirements<br>• Define qualification functional architecture<br>• Define qualification generic physical architecture<br>• Generate qualification coverage matrices (allocate requirements to functional architecture and functions to the generic physical architecture)<br>• Identify risks and mitigation strategies<br>• Create master qualification plan |
| Plan the qualification approaches<br>Acceptance test<br>Validation test<br>Verification test | • Define subfunctions (or test activities) for the functional architecture<br>• Define qualification resources and organizations (instantiated physical architecture)<br>• Assign qualification activities to organizations<br>• Allocate qualification activities to resources<br>• Develop qualification schedules consistent with development schedule |
| Plan qualification activities<br>Acceptance test<br>Validation test<br>Verification test | • Develop detailed derived qualification requirements for the test activities<br>• Develop functional architectures for fulfilling the test activities<br>• Define detailed component architectures for the test resources (identifying what special test fixtures and test stubs are needed)<br>• Generate coverage matrices (allocate derived requirements to functional architectures and functions to physical architectures)<br>• Write activity level qualification plans for each qualification component<br>• Assign qualification responsibilities |
| Plan specific tests<br>Acceptance test<br>Validation test<br>Verification test | • Create test scenarios<br>• Identify required stimulation data for each activity<br>• Write test procedures<br>• Write analysis procedures<br>• Define test and analysis schedules |

system, culminating in the selection of that concept that is deemed most appropriate. This concept selection decision must trace back to the original system concept selection.

Once the qualification objectives have been established, the operational concept for qualification (including key scenarios) can be defined. This operational concept will produce a definition of all high level inputs and outputs of the tests. The definition of the qualification scenarios in consideration of the qualification objectives is establishing at a high level what should be tested and to what precision of confidence. The qualification requirements, based upon the threshold criteria for passing, determine how well the test should be conducted in each area. Each specific test should be considered a system; the major test functions are needed to help define the resources needed for the test. These qualification functions enable the development of qualification requirements; both input/output requirements and qualification-wide/ technology requirements. The qualification requirements in this case involve the examination of the qualification system design to ensure that it satisfies the requirements involved in meeting the qualification objectives. Qualification coverage matrices involve comparisons of the qualification requirements to the qualification activities; these matrices enable the management of qualification requirements to ensure that every requirement is being met by some activity. Even more so than with most systems, there may be risks that the testing process will not be completed in a timely manner; test failures at certain points may cause delays in fixing deficiencies or replacing test items. Therefore, extra effort should be expended to identify risks to meeting qualification-wide requirements (such as schedule and time) and develop risk mitigation strategies for dealing with such risks. Finally, the plan for the qualification process should be documented in a master qualification plan.

The second major qualification planning function of Table 11.2, plan the qualification approach, involves creating specific test activities (subfunctions) as well as the physical and allocated architectures for the qualification system. The physical architecture for a test includes test equipment and facilities, as well as the organizations (people) that will conduct a specific test. After one or more generic qualification architectures have been devised and several instantiated qualification architectures are identified, decisions can be made about the most cost-effective means for achieving the qualification objectives with a reasonable risk. As part of this process for selecting an allocated qualification architecture, the allocation of qualification activities to equipment, facilities, and organizations must be considered. Planned previous qualification data must also be considered so that each test does not retest or overtest certain requirements. Finally, these qualification activities can now be planned in time so that the qualification resources are used efficiently and development schedule requirements are met.

The last two qualification planning functions in Table 11.2 define the qualification activities in greater detail, that is, at the component and CI levels. Planning the qualification activities decomposes each activity to two or three levels of detail, and matches these subactivities to requirements and resources. Planning the specific tests takes each test activity and creates detailed scenario and data specifications of the activity. Test procedures for handling the test equipment and test data are also produced. Finally detailed schedules are produced.

Figure 11.7 depicts the design process of the qualification system as an IDEF0 diagram. Note this is essentially the same process discussed in Chapters 6 to 10 for any system. However, a final activity is added to address the development of all the models needed for qualification.

## 11.7 QUALIFICATION METHODS

Four categories of *qualification methods* are inspection, analysis and simulation, instrumented test, and demonstration. Table 11.3 summarizes each of these methods by describing each, discussing when each is used and when each is most effective.

Inspection is used for physical, human verification of specific requirements. As automation has come to replace humans in the performance of certain activities, more and more of inspection can be accomplished by computers, which falls under instrumented test. A major example of this migration from inspection to instrumented test is the examination of software code for key features or the lack of key features. Finally, qualitative models that are now available with systems engineering tools that allow for extensive inspection opportunities related to design validity and design verification.

Analysis and simulation involves the use of models to test key aspects of the system. Models have always been used in engineering; see Chapter 3 and the discussion of mental models. The most common use of models is to examine the performance of the system in a range of environmental conditions. Initially these models support the design process by enabling the comparison of alternate physical architectures. However, as verification and validation begin, these same models can be used to augment instrumented test and demonstration. Initially, the results of the instrumented test are fed back to the models and used to refine parameters embedded in the model. Later, the models can be used to predict the results of instrumented tests and demonstrations. As confidence in a specific model increases, the model can be used to replace some of the instrumented tests and demonstrations. An important example of this interplay between models and instrumented tests is the development of estimates for such parameters as reliability, availability, and durability [see Holmberg and Folkeson, 1991]. Lee and Yannakakis [1996] provide a detailed survey of the use of one class of models (finite-state machines) in testing. Additional advances are being made in the verification of models that directly relates to verifying systems; see Baier and Katoen [2008].

Table 11.4 describes testing methods that can be used at the system level and lower. These functional and structural testing methods are used in conjunction

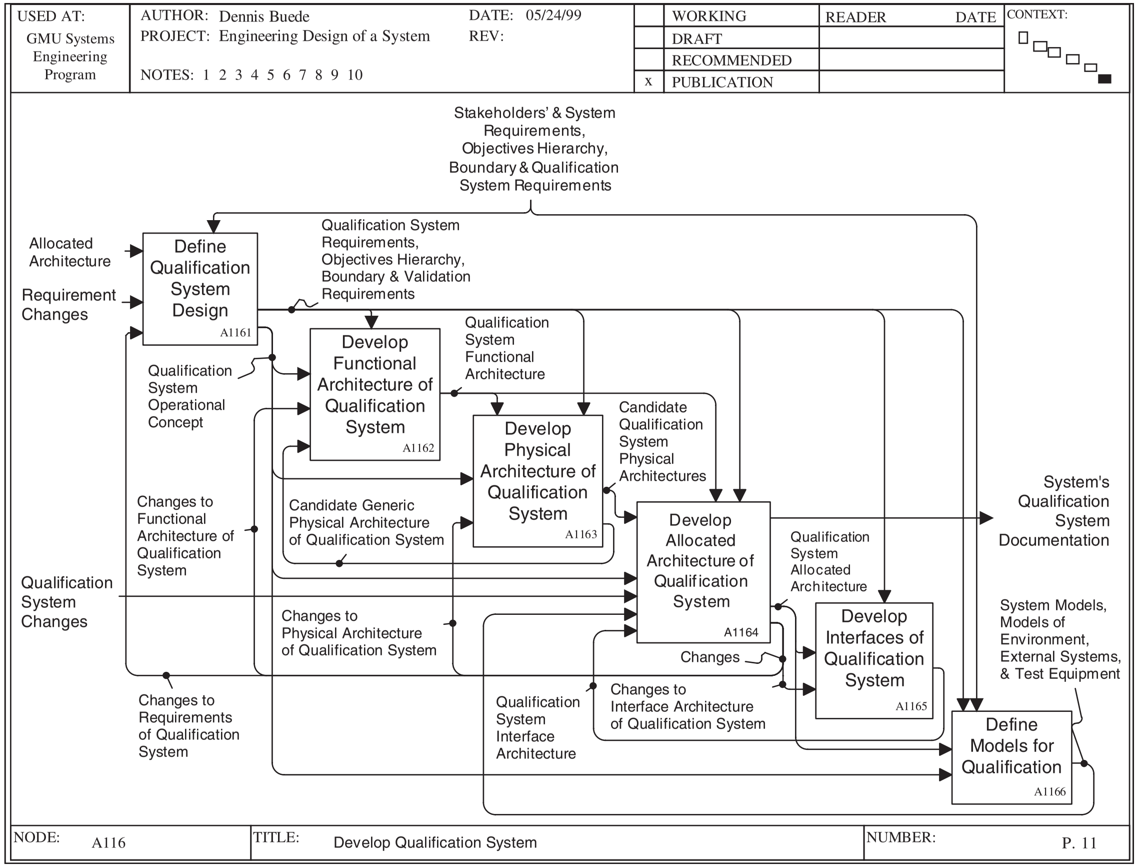
USED AT: GMU Systems Engineering Program
AUTHOR: Dennis Buede
PROJECT: Engineering Design of a System
NOTES: 1 2 3 4 5 6 7 8 9 10
DATE: 05/24/99
REV:
WORKING
DRAFT
RECOMMENDED
x PUBLICATION
READER
DATE
CONTEXT:
Stakeholders' & System Requirements, Objectives Hierarchy, Boundary & Qualification System Requirements
Allocated Architecture
Requirement Changes
Define Qualification System Design
A1161
Qualification System Requirements, Objectives Hierarchy, Boundary & Validation Requirements
Qualification System Operational Concept
Develop Functional Architecture of Qualification System
A1162
Qualification System Functional Architecture
Develop Physical Architecture of Qualification System
A1163
Candidate Qualification System Physical Architectures
Candidate Generic Physical Architecture of Qualification System
Changes to Functional Architecture of Qualification System
Qualification System Changes
Develop Allocated Architecture of Qualification System
A1164
System's Qualification System Documentation
Qualification System Allocated Architecture
Changes to Physical Architecture of Qualification System
Changes
Develop Interfaces of Qualification System
A1165
Changes to Interface Architecture of Qualification System
Qualification System Interface Architecture
Changes to Requirements of Qualification System
System Models, Models of Environment, External Systems, & Test Equipment
Define Models for Qualification
A1166
NODE: A116
TITLE: Develop Qualification System
NUMBER: P. 11

**FIGURE 11.7** The process for developing the qualification system.

**TABLE 11.3 Qualification Methods**

| Method | Description | Used during: | Most effective when: |
|---|---|---|---|
| Inspection (Static Test) | Compare system attributes to requirements. | During all segments of verification, validation and acceptance testing for requirements that can be addressed by human examination. | Success or failure can be judged by humans; examples include inspection of physical attributes, code walk-throughs and evaluation of user's manuals. |
| Analysis and Simulation | Use models that represent some aspect of the system.<br>Examples of models might address system's environment, system process, system failures. | Used throughout qualification, but emphasis is early in verification and during acceptance.<br>Often used in conjunction with demonstration. | Physical elements are not yet available.<br>Expense prohibits instrumented test, and demonstration is not sufficient.<br>Issue involves all or most of the system's life span.<br>Issue cannot be tested (e.g., survive nuclear blast). |
| Instrumented Test | Use calibrated instruments to measure system's outputs. Examples of calibrated instruments are oscilloscope, voltmeter, LAN analyzer. | Verification testing. | Engineering test models through system elements are available.<br>Detailed information is required to understand and trace failures.<br>Life and reliability data is needed for analysis and simulation. |
| Demonstration or Field Test | Exercise system in front of unbiased reviewers in expected system environment. | Primarily used for validation and acceptance testing. | Complete instrumented test is too expensive.<br>High level data/information is needed to corroborate results from analysis and simulation or instrumented test. |

**TABLE 11.4 Testing Methods**

| | |
|---|---|
| Functional testing | Test conditions are set up to ensure that the correct outputs are produced, based upon the inputs of the test conditions. Focus is on whether the outputs are correct given the inputs (also called black box testing). |
| Structural testing | Examines the structure of the system and its proper functioning. Includes such elements as performance, recovery, stress, security, safety, availability. Some of the key elements are described below. |
| Performance | Examination of the system performance under a range of nominal conditions, ensures system is operational as well. |
| Recovery | Various failure modes are created and the system's ability to return to an operational mode is determined. |
| Interface | Examination of all interface conditions associated with the system's reception of inputs and sending of outputs. |
| Stress testing | Above-normal loads are placed on the system to ensure that the system can handle them; these above-normal loads are increased to determine the system's breaking point; these tests may proceed for a long period of time in an environment as close to real as possible. |

with top-down, bottom-up, and big-bang integration. Functional testing examines the system at the level of inputs and outputs under mostly nominal conditions. Structural testing deals with specific characteristics of the outputs as well as the system-wide properties such as safety, availability, and recovery. Structural testing pays particular attention to the most extreme environments that the system will experience.

Samson [1993] postulates four facets for any qualification activity: structural (relation to system implementation), function (relation to system functions), environment (relation to environmental conditions), and conditions (relation to requirement characteristic). The first two of these facets are mutually exclusive and are described in Table 11.4. The second two need to be added to each specific structural or functional test to make it complete. In other words there has to be an environmental facet and a conditional facet for each functional test and each structural test. Table 11.5 shows Samson's examples of these facets.

Black box and white box testing methods (Table 11.6) are commonly employed in software testing. For each method test cases must be specified and test data generated as inputs. These inputs are then injected into both the system prototype (which is essentially a model of the eventual system) and a model of the system. The outputs of the system and the model are compared; any discrepancies are checked to determine whether the system or the model is incorrect [see Chusho, 1987; Richardson and Clarke, 1985; Voges and Taylor, 1985].

## 11.8 ACCEPTANCE TESTING

Acceptance testing is the final step in qualification and is separated from validation because acceptance testing is conducted by the stakeholders, whereas

**TABLE 11.5 Examples of Testing Facets**

| Structural Facet | Functional Facet | Environmental Facet | Conditional Facet |
|---|---|---|---|
| Compliance | Algorithm analysis | Computer-supported | Accuracy |
| Execution | Control | Live | Adequacy |
| External | Error handling | Manual | Boundary |
| Inspection | Intersystem | Prototype | Compliance |
| Operations | Parallel | Simulator | Existence |
| Path | Regression | Testbed | Load |
| Recovery | Requirements | | Location |
| Security | | | Logic |
| | | | Quality |
| | | | Sequence |
| | | | Size |
| | | | Timing |
| | | | Typing |
| | | | Utilization |

verification and validation have been conducted by the development team of systems engineers. In order for the development process to proceed efficiently and effectively, the thresholds for acceptance need to be defined early in the requirements development process by the stakeholders with the help of the systems engineering team. In fact, in Chapter 6 the agreement on the

**TABLE 11.6 Black and White Box Testing**

| | |
|---|---|
| Black box testing | Outputs are determined correct or incorrect based upon inputs; inner workings of the module are ignored. Both positive and negative testing have to be employed. This approach is scalable to system-level testing<br>• Positive testing pulls the test data and sequences from the requirements documents.<br>• Negative testing attempts to find input sequences missed in the requirements documents and then determine how the module reacts. Crash testing is an example. |
| White box testing | Inner workings of the module are examined as part of the testing to ensure proper functioning. Usually used at the CI level of testing; this method becomes impractical at the system level<br>• Path testing addresses each possible simple functionality and is based upon a prescribed set of inputs.<br>• Path domain testing partitions the input space and then examines the outputs for each partition of the input space.<br>• Mutation analysis injects pre-defined errors and tests the error detection and recovery functionalities. |

acceptance criteria was defined to be the exit criterion for the requirements development.

The acceptance test determines whether the stakeholders, especially the bill payer, is willing to accept the system as it is; accept it subject to certain changes; not accept it; or accept it after certain changes have been made. Acceptance testing focuses on the use of the system by true users, typically a small, but representative sample of users. (During verification and validation, members of the systems engineering team and discipline engineers conducted the use of the system.) As a result, usability characteristics of the system are a major focus. Another characteristic of acceptance testing is the lack of time and money to conduct thorough, controlled tests of the system with users from which inferences, based on classical statistics, can be drawn. The two big issues in acceptance testing are what to test and how to test the usability of the system.

### 11.8.1 Deciding What to Test

Common wisdom says that everything possible, including all functionalities or paths, should be tested. The case study about the Ariane 5 failure is one of many examples that support this wisdom. In fact, during verification and validation the key question is not "what should be tested?" but "what have we forgotten to test?" The more systematic the design process the more likely it is that key issues for testing will arise. Nonetheless, it is imperative that everyone involved in the design and integration process constantly question where problems might arise. If only someone on the Ariane 5 development team had insisted on running the new flight envelope through the software of the inertial reference system, the design flaw would have surfaced. This is an area in which the brainstorming techniques discussed in Chapter 9 can be useful to generate potential test issues, not all of which will be meaningful, but some of which may save the system from the disasters of Ariane 5 and Hubble.

The question of "what should be tested?" becomes very relevant during acceptance testing. Acceptance testing substitutes developmental testers with real users but must rely on all of the previous testing activities. Exhaustive repetition of verification and validation is not feasible during acceptance testing due to the limits of time and money. The focus of acceptance testing is whether the system is acceptable or not as is; and if not, why. But what does it mean to say that the system is acceptable? Can we distinguish only between acceptable and unacceptable? Acceptability is defined here to mean the stakeholders want to deploy the system as it is as soon as practically possible, with whatever flaws there are. More flaws are acceptable to stakeholders when the current system's deficiencies are causing severe problems for the users in accomplishing their goals, for the buyers in maintaining market share, or with the victims in suffering too many losses. However, the stakeholders may be willing to accept the system, yet still demand major changes quickly. The system is unacceptable when it will cause more problems than the current system. Similarly, the system can be totally unacceptable beyond the possibility of improvement or unacceptable until certain changes are made.

The acceptance test can either be designed under the assumption that the system is acceptable or that it is not. If the assumption that the system is acceptable is chosen, the test should be designed to prove it is not. A test designed to try to prove that the system is not acceptable would probably include a relatively small set of challenging activities that are key to the system's performance. If the system cannot perform some of these challenging activities, then it can be failed. On the other hand, if the test design assumption is that the system is not acceptable, then a reasonable amount of standard activity would be included in the test in order for the test to prove that the system is acceptable. If the system can pass most of these standard activities, then it can be accepted. Recall that a statement cannot be proven true by example, but it can be proven false by example. This latter approach is the more common in acceptance tests but not the more defensible.

Decision analysis (see Chapter 13) provides a rational, defensible way to analyze alternate acceptance test designs, including a seldom used option of no acceptance test or accept the system after verification and validation. The decision is whether to accept or not accept the system; the other options of accept but fix and do not accept until fixed should also be included. Now test designs are ways to gather information about system parameters about which uncertainty exists. This increased information, when collected during the test, may update this uncertainty in ways that are sufficient to justify accepting or not accepting the system.

## 11.8.2 Usability

In Chapter 6 usability testing with prototypes was discussed as a method of generating requirements. In qualification, usability testing is again used as part of acceptance testing to determine the success with which the requirements have been met.

In fact, usability testing is also used as part of verification testing when an iterative or evolutionary design process is employed. Limited experimental results for evaluating the effectiveness of evolutionary design are reported by Nielsen [1993, p. 107]. The median improvement over four projects was 38% per iteration, but with a high degree of variability. As a result at least three iterations are recommended.

Recall from Chapter 6 that usability concerns five aspects of a user's interaction with a system: learnability, efficiency or ease of use, memorability, error rate, and satisfaction. These characteristics should be part of the system-wide requirements for most systems. These characteristics can typically not be tested adequately until the entire system has been assembled or simulated. During validation, the characteristics are tested by specially defined sets of users. Larger samples of users, often uncontrolled sets of users called beta testers, address these five aspects during acceptance testing.

When designing any test queries, there are two central issues: Is the query reliable and is the query valid? Reliable queries are queries that will result in the same response when repeated. Reliability is a major problem that cannot be solved completely due to the large individual differences among users. Segmenting the users into relatively homogeneous groups along the dimensions of domain experience, computer experience, and experience with the system under development helps significantly to obtain a reasonable chance of repeatability. To obtain sample users in this last of the three dimensions, there must be a sustained effort to train selected users to become very experienced users. Care must be used in defining homogeneous segments of users. If each of the three dimensions is categorized at two extremes, there are 8 ($2^3$) different combinations. Not all of these combinations may be that interesting for the system in question. There may be some interest in user groups that are midrange in one or more of these dimensions; for most systems the predominant number of users will be neither naive nor expert along any of these three dimensions. However, there are some systems for which all users will be trained extensively before even being allowed access to the system, for example, air traffic control systems, and aircraft. However, for these systems the memorability factor of usability may be critical.

Valid queries are those that are measuring the right or appropriate aspect of the system. For usability this will refer back to the five concerns outlined above. See the metrics in Table 6.5.

The best way to achieve reliability and validity of test measures is to set up relevant tasks on which tests will be conducted and measures taken. These tasks should be drawn from the operational concept; each task may be a complete scenario or a small segment of a scenario, depending on where in the qualification process the test is being used. Complete scenarios should be used during acceptance and the latter stages of validation. Segments can be used during prototyping and the early stages of validation. Each task must define a realistic setting for the user in terms of the system and its context, a specified set of circumstances in which to be performing the task, a well-defined outcome that the user is expected to achieve, and a realistic time interval in which to complete the task.

Cox et al. [1994] state the most serious obstacles to successful usability tests are:

- Obtaining test participants that represent the real users of the system
- Securing a representative sample that will be predictive of how the total population will evaluate the system
- Selecting the tasks that are most critical to the usability needs of real users
- Writing test scenarios that accurately represent real task situations that a user will encounter in the system's environment
- Predicting which of the user interface characteristics are most critical or most often used

Yet these obstacles must be overcome for usability testing to be successful.

## 11.9 SUMMARY

Integration begins when assemblies of CIs and components are evaluated in terms of the derived requirements. This process is part of verification, determining that the system was built right. There are several approaches to integration, bottom-up being the most common one to systems engineering. Top-down and big-bang integration are more common in software engineering. Verification and integration end at the system level.

Qualification consists of verification, validation, and acceptance testing. Verification addresses the comparison of the specifications for the system's CIs, components, subsystems, and the system to the actual designs to make sure the designs are right, that is, meet the specifications.

Validation consists of early validation and operational validation. Early validation (conceptual, requirements, and design validity) proceeds during design to ensure that the design process is valid. Conceptual validity addresses the congruence between the stakeholders' needs and the operational concept. This is the hardest element of validation to complete successfully. Requirements validity applies to the conformity between the operational concept and the stakeholders' requirements. Design validity addresses the coherence between the stakeholders' requirements and the layers of derived requirements associated with the system, components, and CIs.

Operational validity may begin before verification is complete, but ends after verification is complete and addresses the conformance of the system as it has been built with the operational concept. This is the last phase of the development process under the complete control of the systems engineering team.

Acceptance testing is controlled by the stakeholders and provides the stakeholders the final opportunity to review the design and verify that it meets their needs. Acceptance testing should fully utilize all of the data and analyses that have been part of verification and validation. At the same time, though, acceptance testing is focused on the use of the system by representatives of the stakeholders' community, whereas verification and validation employ highly qualified users (i.e., engineers) as stakeholders for the most part. As a result the system's usability is a major focus during acceptance testing.

There are two critical chains whose links are checked during qualification. The top-level chain consists of these links: conceptual validity, operational validity, and acceptability. The first link is validated early in the design phase; the last two links are addressed at the end of integration. The second chain consists of requirements validity, design validity, verification, and operational validity. Note that operational validity is common to both chains, and recall that the chain is as strong as its weakest length. Therefore, it is a mistake to assert that any one of the links is more important than any of the others.

## CASE STUDY: THERAC-25

The Therac-25 was a computer-controlled machine that provided radiation therapy in the late 1980s. Three patients were killed and one seriously injured by radiation overdoses in the 1985–1987 time frame when four different operators entered an acceptable, but infrequently used, sequence of commands. While this tragedy can be traced to requirements and design errors, the qualification process should be focused on catching just this sort of flaw. This was clearly a case in which all possible data entry sequences should have been tested [Jacky, 1990].

The development of the qualification system should be approached just as the development of any system, as described in Chapters 6 to 10. The operational concept, external systems diagram, objectives hierarchy, requirements, and architectures (functional, physical, and allocated) are all critical elements of the development of the qualification system. Besides addressing verification, validation, and acceptance, the qualification system is often broken into four methods (or components): inspection, analysis and simulation, instrumented test, and demonstration.

While it is common to visualize the qualification system as the system that will detect and isolate faults in the product system's design, design of the qualification system, when done right, also reinforces the design process and reduces the introduction of faults into the design of the system.

In summary for Section 2 of this book, the Traditional, Top-Down Systems Engineering (TTDSE) process has been described in some detail. Figure 11.8 integrates Figures 1.6 and 1.19 to bring the major elements of Chapters 6 through 11 together into a single picture. The point of this figure is that the process described in Chapters 6 through 11 is repeatedly applied to the process of "peeling the onion" of the layers of the system. Each preceding layer provides the starting information for the layer before it. The major difficulty is getting started when very little needed information is available.

## CASE STUDY: FAILURE OF THE ARIANE 5

Ariane 5, a launch vehicle developed by the European Space Agency (ESA), was first launched on June 4, 1996, with four satellites that would become the backbone of the Solar Terrestrial Science Programme. These four satellites were developed by 500 scientists in over 10 years for about $500 million. But at 37 seconds into the flight Ariane 5 veered off course and disintegrated shortly after. The failure was traced to the two inertial

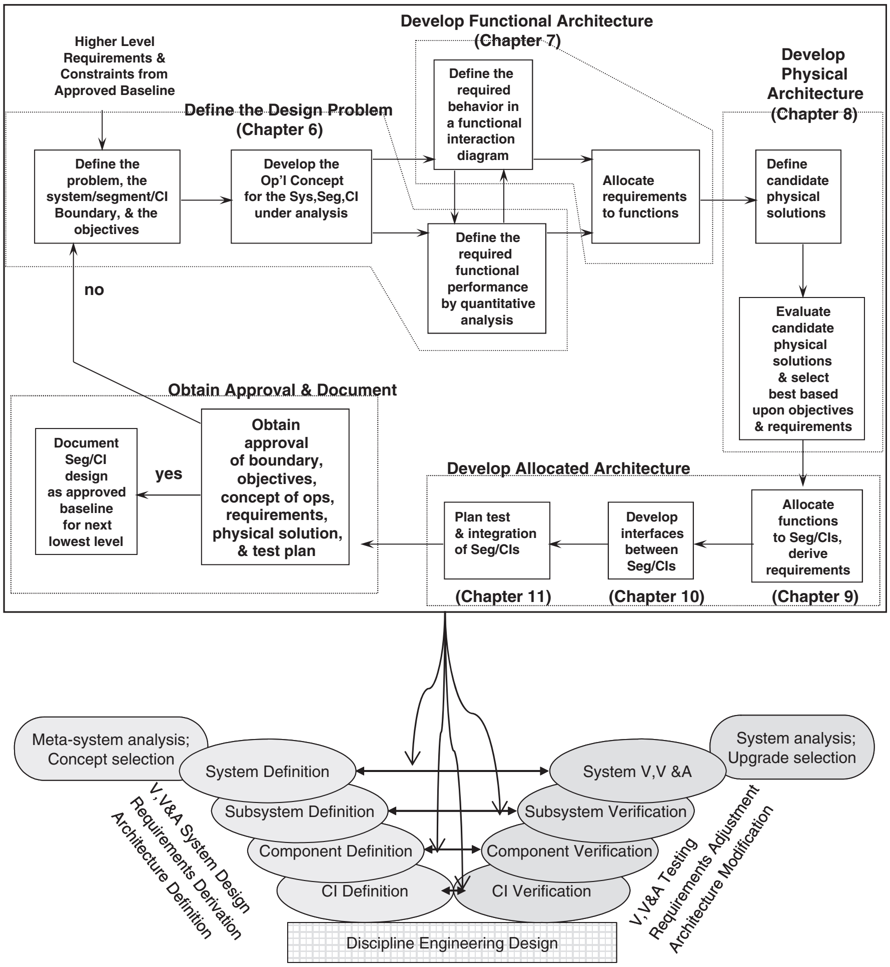
Higher Level Requirements & Constraints from Approved Baseline
Define the Design Problem (Chapter 6)
Define the problem, the system/segment/CI Boundary, & the objectives
Develop the Op'l Concept for the Sys,Seg,CI under analysis
Develop Functional Architecture (Chapter 7)
Define the required behavior in a functional interaction diagram
Define the required functional performance by quantitative analysis
Allocate requirements to functions
Develop Physical Architecture (Chapter 8)
Define candidate physical solutions
Evaluate candidate physical solutions & select best based upon objectives & requirements
Develop Allocated Architecture
Allocate functions to Seg/CIs, derive requirements
Develop interfaces between Seg/CIs
Plan test & integration of Seg/CIs
(Chapter 11)
(Chapter 10)
(Chapter 9)
Obtain Approval & Document
Obtain approval of boundary, objectives, concept of ops, requirements, physical solution, & test plan
no
yes
Document Seg/CI design as approved baseline for next lowest level
Meta-system analysis; Concept selection
System Definition
Subsystem Definition
Component Definition
CI Definition
System analysis; Upgrade selection
System V,V &A
Subsystem Verification
Component Verification
CI Verification
Discipline Engineering Design
V,V&A System Design
Requirements Derivation
Architecture Definition
V,V&A Testing
Requirements Adjustment
Architecture Modification

**FIGURE 11.8** Repeated application of TTDSE to the layers of the system's design.

reference systems (SRIs), one of which was in "hot" standby mode for the other. Both SRIs failed when their software converted a 64-bit floating-point number to a 16-bit signed integer value. The conversion failed when the floating-point number was too large for the 16-bit signed integer, resulting in an operand error for which there was no protection. The system operated as designed when this failure occurred: the failure was indicated on the data bus, the failure context was stored in EEPROM memory, and the SRI processor was shut down.

During the design of the SRI there was a strong theme of designing to prevent random errors. In addition, a requirement had been set to limit the maximum workload of the SRI computer to 80% of its capacity. An analysis was done during the development and testing of the SRI software to determine the vulnerability of the code due to exceptions such as operand errors. Analysis of conversions from floating-point to integer numbers yielded seven variables that could cause an operand error. A deliberate decision was made to protect four of the variables. The other three, including the one that caused the SRI failure, were judged to be protected by either physical limitations or a large margin of safety. A clear trade-off decision was made in this design to risk an operand error in lieu of increasing the workload on the SRI computer.

The testing and qualification procedures set out for the flight control system of Ariane 5 consisted of four levels: equipment qualification, software qualification, stage integration, and system validation. No test was done on the SRI to examine the operational scenario associated with the countdown and flight trajectory of the Ariane 5. This scenario could not be tested with the SRI as a black box. However, the SRI could have been tested by feeding simulated acceleration signals into the SRI while the SRI was placed on a turntable to provide realistic movement. This test was not done because the SRI specification does not require the SRI to be operational after launch. The purpose of the SRI was to provide inertial reference data prior to launch. Even though the SRI served no useful purpose after launch, its operation after launch was sufficient to cause the destruction of Ariane 5 37 seconds into the flight.

Much of the Ariane 5 requirements and software were inherited from earlier versions of Ariane. Ten years earlier requirements had been established that the SRI operate 50 seconds beyond the initiation of flight mode. Flight mode started at — 9 seconds for Ariane-4; this allowed restarting the countdown without waiting for a normal alignment of the spacecraft, which takes about 45 minutes. However, Ariane 5 had a different initiation sequence that did not require the SRI being active during flight. *This is one case in which the old adage "if it ain't broke, don't fix it" caused a failure.*

The final stage at which this error could have been detected was at the Functional Simulation Facility (ISF) which tests (1) guidance, navigation, and control performance in the whole flight envelope, (2) sensor redundancy operation, and (3) flight software compliance with all equipment of the flight control electrical system. "Technically valid arguments" [Lions. 1996] were presented for not having the SRIs in the loop for the tests conducted at the ISF. As a result the SRIs were never tested for the Ariane 5 launch. The trajectory profile of Ariane 5 was sufficiently different than the profiles of previous Ariane launches that this operand error would always occur; a major requirements' failure followed by a failure of test design [Lions, 1996].

## PROBLEMS

11.1 Describe a process of establishing conceptual validity that identifies the elements of conceptual validity and links between pairs of these elements. This process should then establish characteristics such as completeness, consistency, and correctness.

11.2 Describe a process that could be used to establish requirements validity. This process should identify the elements of moving from the operational concept to the stakeholders' requirements, as discussed in Chapter 6. Additional products beyond those discussed in Chapter 6 should be identified that would enable the validation of such characteristics as completeness, consistency, and correctness when comparing the operational concept to the stakeholders' requirements. Examples of comparisons that should be involved are:

- Matching of operational concept elements to elements of the external systems diagram
- Matching of operational concept elements to input/output requirements
- Matching the objectives hierarchy to elements of the external systems diagram
- Matching the objectives hierarchy to input/output requirements
- Matching elements of the external systems diagram to input/output requirements
- Tracing input/output requirements to external items
- Matching the objectives hierarchy to system-wide requirements

11.3 Describe the types of activities (similar to those in Problem 11.2) that could be used to establish design validity. Identify intermediate products that could be used for establishing design validity. In particular, focus on developing the best definition of completeness for requirements that you can.

11.4 Develop an operational concept, external systems diagram, objectives hierarchy, and requirements for the qualification system for a traffic light system.

11.5 Develop an allocated architecture for the qualification system for a traffic light system.