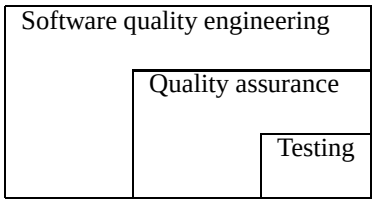

Figure 1.1. Scope and content hierarchy: Testing, quality assurance (QA), and software quality engineering

In addition, all these QA activities need to be managed in an engineering process we call the software quality engineering process, with quality goals set early in the product development, and strategies for QA selected, carried out, and monitored to achieve these preset quality goals. As part of this overall process, data collected during the QA activities, as well as from the overall development activities, can be analyzed to provide feedback to the software development process for decision making, project management, and quantifiable quality improvement. This book also provides a comprehensive coverage of these topics.

1.2 BOOK ORGANIZATION AND CHAPTER OVERVIEW

Figure 1.1. illustrates the general scope of the topics introduced above: Testing is an important subset of QA activities; and QA is an important subset of quality engineering activities. This diagram also explains our book title: "Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement". This book is organized in four major parts and 22 chapters, with the main topics outlined below.

Part I: Overview and basics

Part I gives a general introduction and overview of the topics covered in the book, and presents the basic concepts and definitions related to quality, QA, testing, quality engineering, etc.. Specific questions answered include:

- About this book: What it is? How to use it? How is it organized? In addition, what background knowledge is needed to have a thorough understanding of the technical aspects of this book? These questions are answered in Chapter 1.
- What is software quality? In particular, what are the different views of quality? Is quality a single, atomic concept, or does it consist of many different attributes or characteristics? What is the relationship between quality, correctness, and defect? Can we narrow down the definition of quality to better focus our attention on various QA activities commonly carried out during software life cycles? These questions are answered in Chapter 2.
- What is QA? The question is answered from a particular perspective in Chapter 3, representing a defect-based interpretation of quality and QA.
- What are the different QA activities and related techniques? A defect-based classification is presented, also in Chapter 3, for the major QA alternatives and techniques, such as testing, inspection, formal verification, fault tolerance, etc.

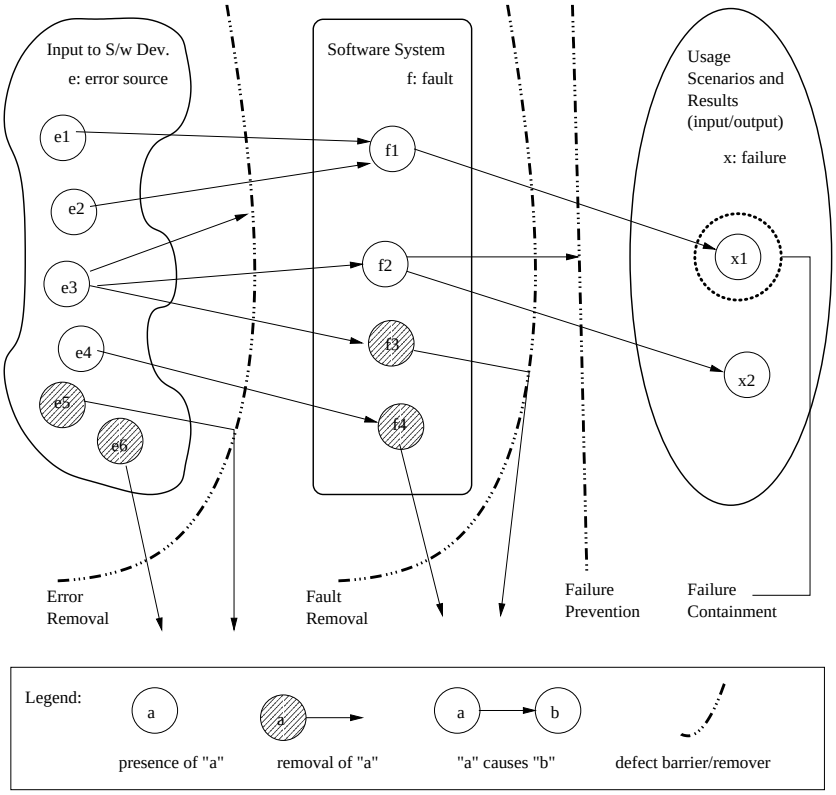

Figure 3.1. Generic ways to deal with defects

between these QA activities and related errors, faults, and failures through some specific examples, as follows:

- Some of the human conceptual errors, such as error source *e6*, are directly removed by error source removal activities, such as through better education to correct the specific human conceptual mistakes.

- Other incorrect actions or errors, such as some of those caused by error source *e3* and *e5*, are blocked. If an error source can be consistently blocked, such as *e5*, it is equivalent to being removed. On the other hand, if an error source is blocked sometimes, such as *e3*, additional or alternative defect prevention techniques need to be used, similar to the situation for other error sources such as *e1*, *e2*, and *e4*, where faults are likely to be injected into the software system because of these error sources.

- Some faults, such as *f4*, are detected directly, i.e., without involving the observation of failures, through inspection or other static analysis, and removed as a part of or as followup to these activities.

- Other faults, such as *f3*, are detected through testing or other execution-based QA alternatives by observing their dynamic behavior. If a failure is observed in these QA activities, the related faults are located by examining the execution record and

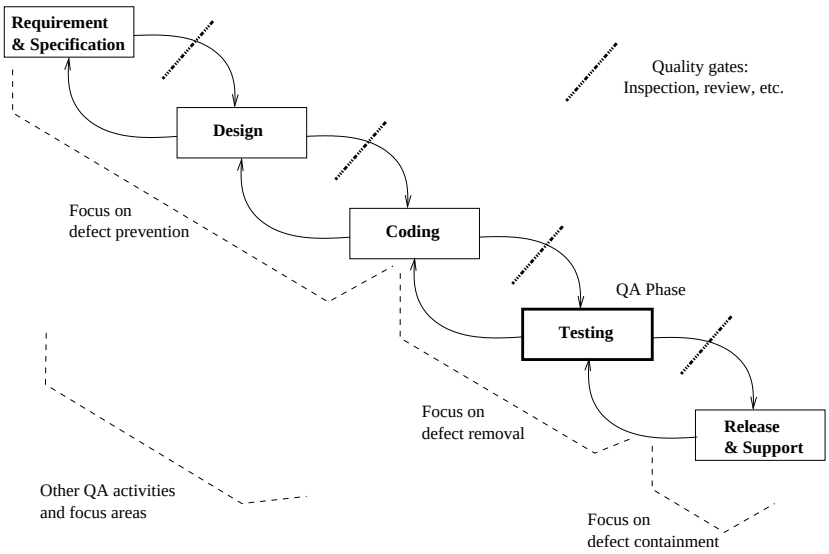

Figure 4.1. QA activities in waterfall process

Because of the possibilities of defect propagations and the increasing cost over time or successive development phases to fix defects once they are injected into the system, we need to reduce the number of faults in software systems by the combination of defect prevention and application of QA techniques that can help remove software faults early. Some defect detection and removal techniques, such as inspection, can be applied to early phases, such as inspecting requirement documents, product specifications, and different levels of product designs. On the other hand, there are practical obstacles to the early fixing of injected defects. For example, dynamic problems may only become apparent during execution; and inter-dependency only becomes apparent with the implementation of related components or modules. Because of these reasons, other fault detection and removal activities, such as testing, are typically concentrated in the middle to late phases of software development.

Finally, failure prevention and containment activities, such as fault tolerance and safety assurance, are typically the focus of operational phases. However, their planning, design, and implementation need to be carried out throughout the software development process. In some sense, they are equivalent to adding some necessary functions or features into the existing product to make them safe or fault tolerant.

Figure 4.1. illustrate how the different QA activities fit into the waterfall process. Three key characteristics of this activity distribution are illustrated:

- The phase with QA as the focus: Testing phase.
- QA activities, typically inspections and reviews, carried out at the transitions from one phase to the next are shown as barriers or gates to pass. The exception to this is between testing and release, where the reviews are typically accompanied by acceptance testing.
- Other QA activities scatter over all other development phases: The general distribution scope is shown by the dotted bracket, with a focus on defect prevention in the early phases, a focus on defect removal during coding and testing phases, and a focus on defect containment in operational support.

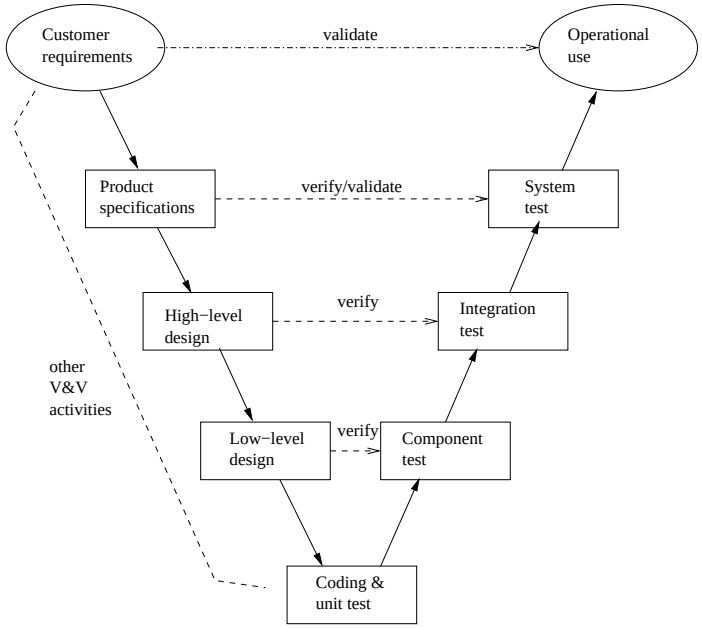

Figure 4.2. Verification and validation activities associated with the V-Model

addition, system test also *validates* the product by focusing on how the overall operations under an environment that resembles that for target customers. In a sense, the users' operational environment is captured as part of the product specification or as part of the testing model. At the bottom, coding and unit testing are typically grouped in a single phase, where the code itself specifies the expected behavior and needs to be verified through unit test. Sometimes, various other QA activities, such as inspections, reviews, walkthroughs, analyses, formal verification, etc., are also associated with the left arm of the V-model and illustrated by additional dotted lines pointed to the specific phases.

Similar to the mapping of QA activities to other process models above, validation and verification activities can be mapped into non-sequential processes such as incremental, iterative, spiral, and extreme programming processes. Typically, there is some level of user involvement in each part or iteration. Therefore, validation plays a more important role in these processes than in waterfall process or V-model.

4.4 RECONCILING THE TWO VIEWS

The above descriptions of verification and validation activities included examples of specific QA activities. These specific QA activities were also classified using our scheme according to the generic ways of dealing with defects. Through this connection and the inter-relations represented therein, we can establish the relationship and the mapping between the verification and validation (V&V) view on the one hand and our defect-centered (DC) view and classification on the other hand. In addition, we can use the process information as presented in Figure 4.1., Figure 4.2., and related discussions to help us with this mapping, as discussed below.

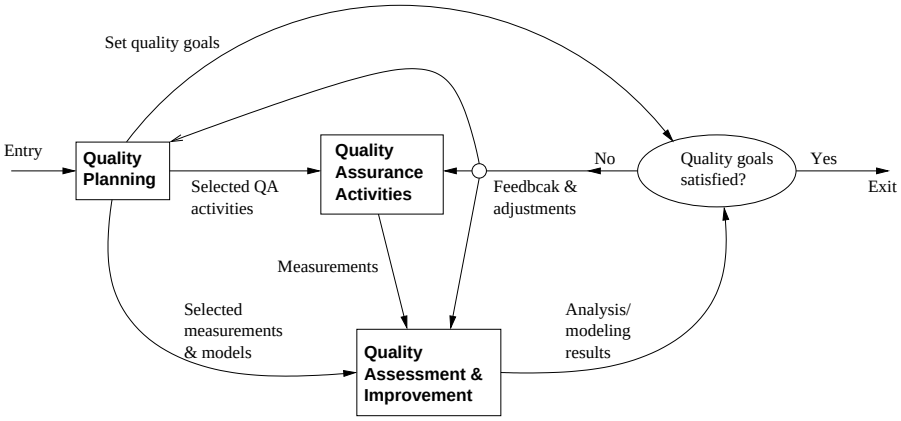

Figure 5.1. Quality engineering process

1. *Pre-QA activities: Quality planning.* These are the activities that should be carried out before carrying out the regular QA activities. There are two major types of pre-QA activities in quality planning, including:

   (a) Set specific quality goals.

   (b) Form an overall QA strategy, which includes two sub-activities:

      i. Select appropriate QA activities to perform.

      ii. Choose appropriate quality measurements and models to provide feedback, quality assessment and improvement.

   A detailed description of these pre-QA activities is presented in Section 5.2.

2. *In-QA activities: Executing planned QA activities and handling discovered defects.* In addition to performing selected QA activities, an important part of this normal execution is to deal with the discovered problems. These activities were described in the previous two chapters.

3. *Post-QA activities: Quality measurement, assessment and improvement* These are the activities that are carried out after normal QA activities have started but not as part of these normal activities. The primary purpose of these activities is to provide quality assessment and feedback so that various management decisions can be made and possible quality improvement initiatives can be carried out. These activities are described in Section 5.3.

Notice here that "post-QA" does not mean after the finish of QA activities. In fact, many of the measurement and analysis activities are carried out parallel to QA activities after they are started. In addition, pre-QA activities may overlap with the normal QA activities as well.

Pre-QA quality planning activities play a leading role in this quality engineering process, although the execution of selected QA activities usually consumes the most resources. Quality goals need to be set so that we can manage the QA activities and stop them when the quality goals are met. QA strategies need to be selected, before we can carry out specific QA activities, collect data, perform analysis, and provide feedback.

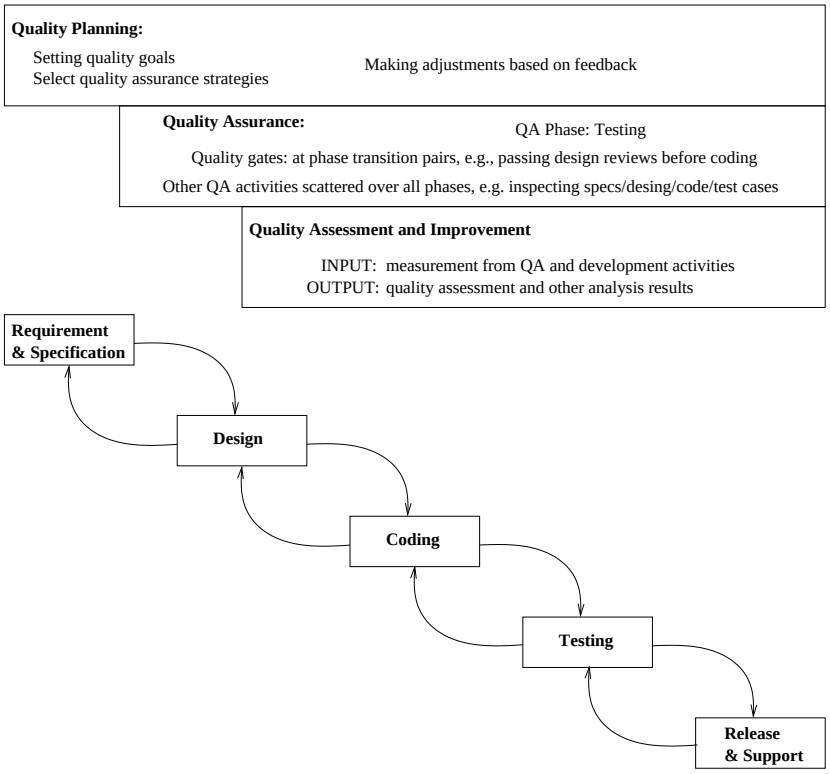

Figure 5.2. Quality engineering in the waterfall process

All these activities typically last over the whole development process, with different sub-activities carried out in different phases. This is particularly true for the QA activities, with testing in the test phase, various reviews or inspections at the transition from one phase to its successor phase, and other QA activities scattered over other phases.

Minor modifications are needed to integrate quality engineering activities into other development processes. However, the distribution of these activities and related effort is by no means uniform over the activities or over time, which is examined next.

5.4.2 Effort profile

Among the three major types of activities in the quality engineering process, the execution of specific QA activities is central to dealing with defects and assuring quality for the software products. Therefore, they should and normally do consume the most resources in terms of human effort as well as utilization of computing and other related resources. However, the effort distribution among the three is not constant over time because of the process characteristics described above and the shifting focus over time. Some key factors that affect and characterize the effort profile, or the effort distribution over time, include:

- Quality planning drives and should precede the other two groups of activities. Therefore, at the beginning part of product development, quality planning should be the dominant part of quality engineering activities. Thereafter, occasional adjustments

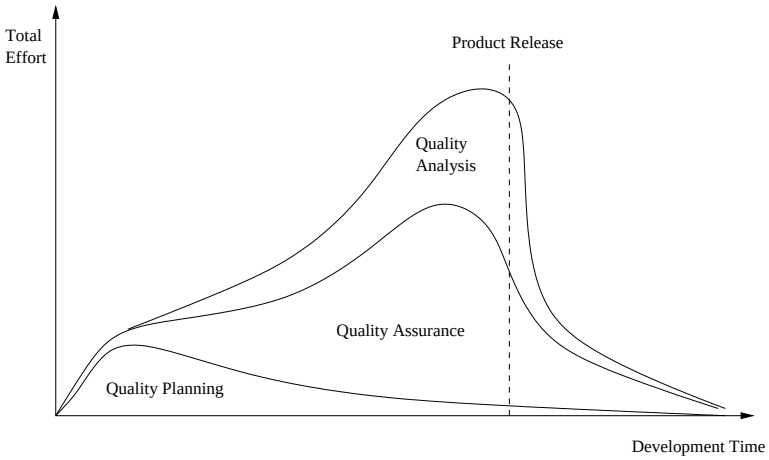

Figure 5.3. Quality engineering effort profile: The share of different activities as part of the total effort

In addition, the general shape and pattern of the profile such as in Figure 5.3. should preserve regardless of the specific development process used. Waterfall process would see more dominance of quality planning in the beginning, and dominance of testing near product release, and measurement and quality assessment activities peak right before product release.

Other development processes, such as incremental, iterative, spiral, and extreme programming processes, would be associated with curves that vary less between the peaks and valleys. QA is spread out more evenly in these processes than in the waterfall process, although it is still expected to peak a little bit before product release. Similarly, measurement and analysis activities are also spread out more evenly to monitor and assess each part or increment, with the cumulative modeling results used in product release decisions. There are also more adjustments and small scale planning activities involved in quality planning, which also makes the corresponding profiles less variable as well.

5.5 CONCLUDING REMARKS

To manage the quality assurance (QA) activities and to provide realistic opportunities of quantifiable quality improvement, we need to go beyond QA to perform the following:

- *Quality planning* before specific QA activities are carried out, in the so-called pre-QA activities in software quality engineering. We need to set the overall quality goal by managing customer's quality expectations under the project cost and budgetary constraints. We also need to select specific QA alternatives and techniques to implement as well as measurement and models to provide project monitoring and qualitative feedback.

- *Quality quantification and improvement* through measurement, analysis, feedback, and followup activities. These activities need to be carried out after the start of specific QA activities, in the so-called post-QA activities in software quality engineering. The analyses would provide us with quantitative assessment of product quality, and

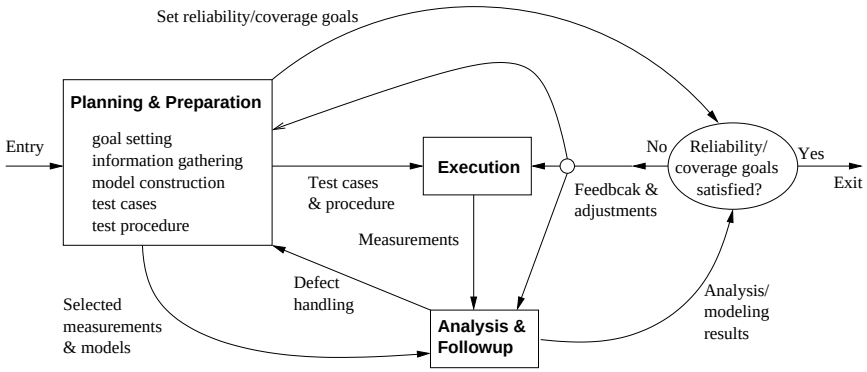

Figure 6.1. Generic testing process

observations. In fact, many forms of informal testing include just this middle group of activities related to test execution, with some informal ways to communicate the results and fix the defects, but without much planning and preparation. However, as we will see in the rest of Part II, in all forms of systematic testing, the other two activity groups, particularly test planning and preparation activities, play a much more important role in the overall testing process and activities.

The execution of a specific test case, or a sub-division of the overall test execution sequence for some systems that require continuous operation, is often referred to as a "test run". One of the key component to effective test execution is the handling of problems to ensure that failed runs will not block the executions of other test cases. This is particularly important for systems that require continuous operation. To many people, defect fixing is not considered to be a part of testing, but rather a part of the development activities. However, re-verification of problem fixes is considered as a part of testing. In this book, we consider all of these activities as a part of testing.

Data captured during execution and other related measurements can be used to locate and fix the underlying faults that led to the observed failures. After we have determined if a test run is a success or failure, appropriate actions can be initiated for failed runs to locate and fix the underlying faults. In addition, further analyses can be performed to provide valuable feedback to the testing process and to the overall development process in general. These analysis results provide us with assessments of the current status with respect to progress, effort, defect, and product quality, so that decisions, such as when to stop testing, can be made based on facts instead of on people's gut feelings. In addition, some analyses can also help us identify opportunities for long term product quality improvement. Therefore, various other activities, such as measurement, analysis, and followup activities, also need to be supported.

Sub-activities in test planning and preparation

Because of the increasing size and complexity of today's software products, informal testing without much planning and preparation becomes inadequate. Important functions, features, and related software components and implementation details could be easily overlooked in such informal testing. Therefore, there is a strong need for planned, monitored, managed and optimized testing strategies based on systematic considerations for quality, formal models, and related techniques. Test cases can be planned and prepared using such testing

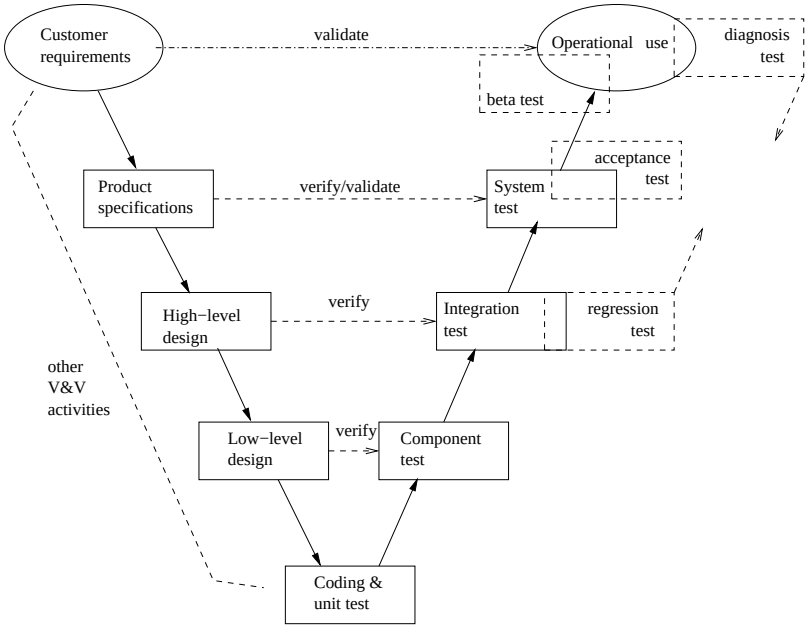

Figure 12.1. Testing sub-phases associated with the V-Model

Testing sub-phases

Figure 12.1. illustrates the testing sub-phases through the use of V-model, a variation of the commonly used waterfall process with an emphasis on verification and validation activities. It shows an annotated V-model with additional information about the specific testing sub-phases. All the sub-phases not included in the original V-model are shown in dashed boxes, with their relationship to the other sub-phases also illustrated. Specific information about these sub-phases is described below:

- When problems are reported by customers during operational use, *diagnosis testing* can be used to recreate and diagnose the problems. Diagnosis testing can also be used for other sub-phases of testing for the same purpose as well, as illustrated by the downward arrow in Figure 12.1..

- Controlled product release and operational use by limited customers lead to *beta testing*. Beta testing can be considered as an additional testing sub-phase closely linked to operational use. It often directly precedes operational use or is carried out at the very beginning of it, as depicted accordingly in Figure 12.1..

- A special testing sub-phases, *acceptance testing*, is attached to the end of system testing, because it is typically performed right after system testing to determine if the product should be released. Sometimes, the late part of system testing is used as acceptance test instead of a dedicated acceptance testing sub-phases. Therefore, we also show possible overlap between the two in Figure 12.1..

- As a direct division of the testing phase in the waterfall process, several specific sub-phases of testing, such as *system testing*, *integration testing*, and *component testing*