As described in Chapter 4, a software unit is the smallest separately testable element in the design of a software system. They cannot be subdivided into other components. A software component is composed of software units. It is a system element offering a predefined service and is able to communicate with other software components. Software components may be tested in isolation or in combination with other components. The former is called component level testing, whereas the latter refers to integration level testing. Integration level testing is the phase of software testing in which software components are combined and tested as a group.

During the development of complex software systems, software components are often in different development states. Whereas one component may be in its implementation phase, another component may be ready for integration with other components. Different development states cannot be avoided. They are the consequence of parallel development in several groups of developers, different size and complexities of the software components and project priorities.

Component and integration level testing is therefore often confronted with the problem that the service provided by the component or a group of components under test (the SUT) requires functionality of components which are not ready for integration. Delays of the component and integration level testing process can be avoided by the development of emulators for the missing functionality and an elaborated project plan which considers the integration order of the components.

Due to the similarities of component and integration level testing, we cover both in this chapter. To ease the reading, we will mainly use the term integration level testing. Only where necessary, we will refer to component level testing.

In this chapter, we will discuss some basics on integration level testing and show how the UML Testing Profile (UTP) can be utilized for this kind of testing.
5 Component and Integration Level Testing

5.1 Integration Strategies and Integration Level Testing

The objective of integration level testing is to test the smooth interoperation of components. Some properties may be tested statically, whereas errors related to the semantics of the communication can only be tested dynamically. For example, a compiler can statically check the correct usage of interfaces provided by a component, whereas the exchange of syntactically correct but semantically incorrect information can only be detected dynamically.

Integration level testing depends heavily on the integration strategy used for assembling the whole system. Well-known integration strategies are big-bang, bottom-up, top-down, and adhoc integration:

- The integration of the whole system in one step is called big-bang integration. In this case, integration level testing starts after the finalization of all components. The problems with this approach are that the integration level testing starts very late in the development process and that all the integration problems appear at once. Testing may become complicated, and it can be very difficult to identify the source of errors observed during the test.
- A bottom-up integration strategy requires a hierarchical system structure. Software components with basic functionality are assembled to yield more complex components, which are integrated to produce even more complex subsystems, etc., until the complete system is produced. Testing is done in all steps of integration. The advantage of this strategy is that only test drivers, but no emulators, are needed for testing because the availability of already tested lower-level components is a prerequisite for the integration of the next level of hierarchy. The problem with this strategy is that the test of the overall functionality of the system starts very late, and therefore, design and efficiency problems related to the basic functions of the system may be detected only at the end of the integration level testing.
- Top-down integration also requires a hierarchical system structure. It is the opposite of bottom-up integration. Integration starts from the top, and step by step lower-level functionality is added until the complete system is realized. For integration level testing, the main disadvantage of this strategy is that the lower-level functionality is not available and has to be emulated. This is normally done by implementing emulators for the missing components.
- Bottom-up integration and top-down integration require a hierarchical system structure and predictable finalization dates for the different components. In most cases, these prerequisites cannot be guaranteed. Therefore, the most popular integration strategy is adhoc integration, that is, components are integrated whenever they are finalized and whenever it is reasonable. Integration testing with an adhoc integration strategy also requires the emulation of missing functionality.