The third party involved in software production is the user. The *user* is the person or persons on whose behalf the client has commissioned the product and who will utilize the software. In the insurance company example, the users may be insurance agents, who will use the software to select the most appropriate policy. In some instances, the client and the user will be the same person (for example, the accountant discussed previously).

As opposed to expensive custom software written for one client, multiple copies of software, such as word processors or spreadsheets, are sold at much lower prices to a large number of buyers. That is, the manufacturers of such software (such as Microsoft or Borland) recover the cost of developing a product by volume selling. This type of software usually is called *commercial off-the-shelf* (or COTS) software. The earlier term for this type of software was *shrink-wrapped software,* because the box containing the CD or diskettes, the manuals, and the license agreement almost always was shrink-wrapped. Nowadays, COTS software often is downloaded over the World Wide Web—there is no box to shrink wrap. For this reason, COTS software nowadays sometimes is referred to as *click-wrapped software*. COTS software is developed for "the market"; that is, there is no specific client or users until the software has been developed and is available for purchase.

In the next part of this chapter, we present the seven phases of the software life cycle and carefully analyze the role played by testing in each phase. The first phase is the requirements phase.

2.2 REQUIREMENTS PHASE

Software development is an expensive process. The development process usually begins when the client approaches a development organization with regard to a software product that, in the opinion of the client, is either essential to the profitability of his or her enterprise or somehow can be justified economically. At any stage of the process, if the client stops believing that the software will be cost effective, development will terminate immediately. Throughout this chapter the assumption is made that the client feels that the cost is justified. (In fact, the "cost" is not always purely financial. For example, military software often is built for strategic or tactical reasons. Here, the cost of the software is the potential damage that could be suffered in the absence of the weapon being developed.)

At an initial meeting between client and developers, the client outlines the product as he or she conceptualizes it. From the viewpoint of the developers, the client's description of the desired product may be vague, unreasonable, contradictory, or simply impossible to achieve. The task of the developers at this stage is to determine exactly what the client needs and to find out from the client what constraints exist. A typical constraint is the deadline. For example, the client may stipulate that the finished product must be completed within 14 months. A variety of other constraints often are present such as reliability (the product must be operational 99 percent of the time) or the size of the object code (it has to run on the client's personal computer).

The most important constraint usually is the cost. However, the client rarely tells the developers how much money is available to build the product. Instead, a common practice is that, once the specifications have been finalized, the client asks the developers to name their price for completing the project. Clients follow this bidding procedure in the hope that the amount of the developer's bid will be lower than the amount the client has budgeted for the project.

This preliminary investigation of the client's needs sometimes is called *concept exploration.* In subsequent meetings between members of the development team and the client team, the functionality of the proposed product is successively refined and analyzed for technical feasibility and financial justification.

Up to now, everything seems to be straightforward. Unfortunately, the requirements phase frequently is performed inadequately. When the product finally is delivered to the user, perhaps a year or two after the specifications have been signed off by the client, the client may say to the developers, "I know that this is what I asked for, but it isn't really what I wanted." What the client asked for and, therefore, what the developers thought the client wanted, was not what the client actually *needed.* There can be a number of reasons for this predicament. First, the client may not truly understand what is going on in his or her own organization. For example, it is no use asking the software developers for a faster operating system if the cause of the current slow turnaround is a badly designed database. Or, if the client operates an unprofitable chain of retail stores, the client may ask for a financial management information system that reflects such items as sales, salaries, accounts payable, and accounts receivable. Such a product will be of little use if the real reason for the losses is shrinkage (shoplifting, and theft by employees). If that is the case, then a stock control product rather than a financial control product is required.

But the major reason why the client frequently asks for the wrong product is that software is complex. If it is difficult for a software professional to visualize a piece of software and its functionality, the problem is far worse for a client who is barely computer literate. There are a number of ways of coping with this; one of them is rapid prototyping.

A *rapid prototype* is a piece of software hurriedly put together that incorporates much of the functionality of the target product but omits those aspects generally invisible to the client, such as file updating or error handling. The client and users then experiment with the prototype to determine whether it indeed meets their needs. The rapid prototype can be changed until the client and users are satisfied that it encapsulates the functionality they desire. Rapid prototyping and other requirements analysis techniques are discussed in detail in Chapter 10.

2.2.1 Requirements Phase Testing

Within every software development organization should be a group whose primary responsibility is to ensure that the delivered product is what the client ordered and that the product has been built correctly in every way. This group is called the software quality assurance (SQA) group. The quality of software is the extent to which it meets

its specifications. Quality and software quality assurance are described in more detail in Chapter 6, as is the role of SQA in setting up and enforcing standards.

The SQA group must play a role right from the start of the development process. In particular, it is vital that the product satisfy the client's needs. The SQA group, therefore, must verify with the client that the final version of the rapid prototype is totally satisfactory.

It is essential that the rapid prototype be carefully checked by both client and users to be certain that it reflects their current needs. Nevertheless, no matter how meticulously this is done, there always is the possibility that forces beyond the control of the development team will necessitate changes in the requirements while the product is being developed. Further development then has to be put on hold until the necessary modifications have been made to the partially completed product.

A major issue in software development is the so-called moving target problem. That is, the client changes the requirements during development. One reason this occurs is an unforeseeable change in circumstances. For example, if a company expands its operations, or is taken over by another company, then many products have to be modified, including those still under development. However, the major cause of the moving target problem is a client who keeps changing his or her mind. As explained in Section 16.4.4, nothing can be done about it if the client has sufficient clout.

2.2.2 REQUIREMENTS PHASE DOCUMENTATION

The documentation produced during the requirements phase usually includes the rapid prototype together with the records of the discussions with the client and users on the basis of which the rapid prototype was built and modified. If the team decides not to build a rapid prototype, a requirements document is drawn up that describes the needs of the client. This document needs to be checked by the client, selected users, and the development team before the SQA group scrutinizes it meticulously.

2.3 SPECIFICATION PHASE

Once the client agrees that the developers understand the requirements, the *specification document* is drawn up by the specification team. As opposed to the informal requirements phase, the specification document (or *specifications*) explicitly describes the functionality of the product—that is, precisely what the product is supposed to do—and lists any constraints that the product must satisfy. The specification document includes the inputs to the product and the required outputs. For example, if the client needs a payroll product, then the inputs include the pay scales of each employee, data from a time clock, as well as information from personnel files so that taxes can be computed correctly. The outputs are paychecks and reports such as Social Security deductions. In addition, the specification document includes stipulations that the

product must be able to handle correctly a wide range of deductions, such as medical insurance payments, union dues, and pension fund contributions.

The specification document of the product constitutes a contract. The software developers will be deemed to have completed the contract when they deliver a product that satisfies the acceptance criteria of the specification document. For this reason, the specification document should not include imprecise terms like *suitable, convenient, ample,* or *enough* or similar terms that sound exact but in practice are equally imprecise, such as *optimal* or *98 percent complete.* Whereas contract software development can lead to a lawsuit, there is no chance of the specification document forming the basis for legal action when the client and developers are from the same organization. Nevertheless, even in the case of internal software development, the specification document always should be written as if it will be used as evidence in a trial.

More important, the specification document is essential for both testing and maintenance. Unless the specification document is precise, we cannot determine whether the specifications are correct, let alone whether the implementation satisfies the specifications. And it is hard to change the specifications during the maintenance phase unless we have a document that tells us exactly what the specifications currently are.

A variety of difficulties can arise during the specification phase. One possible mistake that can be made by the specification team is that the specifications are *ambiguous*—certain sentences or sections may have more than one valid interpretation. Consider the specification, "A part record and a plant record are read from the database. If it contains the letter A directly followed by the letter Q, then compute the cost of transporting that part to that plant." To what does the *it* in the preceding sentence refer: the part record or the plant record? In fact, the *it* conceivably even could refer to the database.

The specifications also may be *incomplete;* that is, some relevant fact or requirement may be omitted. For instance, the specifications may not state what actions are to be taken if the input data contain errors. Moreover, the specifications may be *contradictory.* For example, in one place in the specification document for a product that controls a fermentation process, it is stated that if the pressure exceeds 35 psi, then valve M17 immediately must be shut. However, in another place, it is stated that if the pressure exceeds 35 psi, then the operator immediately must be alerted; only if the operator takes no remedial action within 30 seconds should valve M17 be shut automatically. Software development cannot proceed until such problems in the specifications have been corrected.

Once the specifications are complete, detailed planning and estimating commences. No client will authorize a software project without knowing in advance how long the project will take and how much it will cost. From the viewpoint of the developers, these two items are just as important. If the developers underestimate the cost of a project, then the client will pay the agreed fee, which may be significantly less than the actual cost to the developers. Conversely, if the developers overestimate what the project will cost, then the client may turn down the project or have the job done by other developers whose estimate is more reasonable. Similar issues arise with regard to duration estimation. If the developers underestimate how long it will take to complete a project, then the resulting late delivery of the product, at best, will result

in a loss of confidence on the part of the client. At worst, lateness penalty clauses in the contract will be invoked, causing the developers to suffer financially. Again, if the developers overestimate how long it will take for the product to be delivered, the client may well award the job to developers who promise faster delivery.

For the developers, merely estimating the duration and total cost is not enough. The developers need to assign the appropriate personnel to the various stages of the development process. For example, the coding team cannot start until the design documents have been approved by the SQA group, and the design team is not needed until the specification team has completed its task. In other words, the developers have to plan ahead. A software project management plan (SPMP) must be drawn up that reflects the separate phases of the development process and shows which members of the development organization are involved in each task, as well as the deadlines for completing each task.

The earliest that such a detailed plan can be drawn up is when the specifications have been finalized. Before that time, the project is too amorphous to undertake complete planning. Some aspects of the project certainly must be planned right from the start, but until the developers know exactly what is to be built, they cannot specify all aspects of the plan for building it.

Therefore, once the specification document has been finished and checked, preparation of the software project management plan commences. Major components of the plan are the deliverables (what the client is going to get), the milestones (when the client gets them), and the budget (how much it is going to cost).

The plan describes the software process in fullest detail. It includes aspects such as the life-cycle model to be used, the organizational structure of the development organization, project responsibilities, managerial objectives and priorities, the techniques and CASE tools to be used, and detailed schedules, budgets, and resource allocations. Underlying the entire plan are the duration and cost estimates; techniques for obtaining such estimates are described in Section 9.2.

The specification phase is described in Chapters 11 and 12: Classical techniques are described in Chapter 11, and object-oriented analysis is the subject of Chapter 12. (The term *analysis* sometimes is used to describe activities of the specification phase, hence the phrase *object-oriented analysis.*)

2.3.1 SPECIFICATION PHASE TESTING

As pointed out in Chapter 1, a major source of faults in delivered software is faults in the specification document that are not detected until the software has been installed on the client's computer and is being used by the client's organization for its intended purpose. The SQA group therefore must check the specifications carefully, looking for contradictions, ambiguities, and any signs of incompleteness. In addition, the SQA group must ensure that the specifications are feasible; for example, that any specified hardware component is fast enough or that the client's current online disk storage capacity is adequate for handling the new product. If a specification document is to be testable, then one of the properties it must have is *traceability.* It must be possible

to trace every statement in the specification document back to a statement made by the client team during the requirements phase. If the requirements have been presented methodically, properly numbered, cross-referenced, and indexed, then the SQA group should have little difficulty tracing through the specification document and ensuring that it is indeed a true reflection of the client's requirements. If rapid prototyping has been used in the requirements phase, then the relevant statements of the specification document should be traceable to the rapid prototype.

An excellent way of checking the specification document is by review. Representatives of the specification team and of the client are present. The meeting usually is chaired by a member of the SQA group. The aim of the review is to determine whether the specifications are correct. The reviewers go through the specification document, ensuring that there are no misunderstandings about the document. Walkthroughs and inspections are two types of reviews, and they are described in Section 6.2.

Consider now the checking of the detailed planning and estimating that takes place once the client has signed off the specifications. Whereas it is essential that every aspect of the SPMP be meticulously checked by the SQA group, particular attention must be paid to the plan's duration and cost estimates. One way to do this is for management to obtain two (or more) independent estimates of both duration and cost at the start of the planning phase, then to reconcile any significant differences. With regard to the SPMP document, an excellent way to verify it is by a review similar to the review of the specification document. If the duration and cost estimates are satisfactory, then the client will give permission for the project to proceed.

2.3.2 Specification Phase Documentation

The specification phase has two primary outputs. The first is the specification document (specifications). Chapters 11 and 12 describe how the specifications are drawn up. The second output is the software project management plan. An explanation of how to draw up the SPMP is given in Sections 9.3 though 9.5.

The next stage is to design the product.

2.4 Design Phase

The specifications of a product spell out *what* the product is to do. The aim of the design phase is to determine *how* the product is to do it. Starting with the specifications, the design team determines the internal structure of the product. The designers decompose the product into *modules,* independent pieces of code with well-defined interfaces to the rest of the product. (An object is a specific type of module.) The interface of each module, that is, the arguments passed to the module and the arguments returned by the module, must be specified in detail. For example, a module might measure the water level in a nuclear reactor and cause an alarm to sound if the level is too low. A method in an object of an avionics product might take as input two or more sets of

coordinates of an incoming enemy missile, compute its trajectory, and send a message to another object to advise the pilot as to possible evasive action.

Once the team has completed the decomposition into modules (the *architectural design*), the *detailed design* is performed. For each module, algorithms are selected and data structures chosen.

While the decomposition into modules is being performed, the design team must keep a careful record of the design decisions that are made. This information is essential for two reasons. First, while the product is being designed, there will be times when a dead end is reached and the design team feels the need to backtrack and redesign certain pieces. Having a written record of why specific decisions were made assists the team when this occurs and helps it get back on track.

The second reason for keeping the design decisions concerns maintenance. Ideally, the design of the product should be open-ended, meaning future enhancements can be done by adding new modules or replacing existing modules without affecting the design as a whole. Of course, in practice, this ideal is difficult to achieve. Deadline constraints in the real world are such that designers struggle against the clock to complete a design that satisfies the original specification document, without worrying about any later enhancements. If future enhancements (to be added after the product has been delivered to the client) are included in the specification document, then these must be allowed for in the design, but this situation is extremely rare. In general, the specification document, and hence the design, deals with only present requirements. In addition, there is no way to determine, while the product is still in the design phase, all possible future enhancements. And finally, if the design has to take *all* future possibilities into account, at best it will be unwieldy; at worst, it will be so complicated that implementation is impossible. So the designers have to compromise, putting together a design that can be extended in many reasonable ways without the need for total redesign. But, in a product that undergoes major enhancement, the time will come when the design simply cannot handle further changes. When this stage is reached, the product must be redesigned as a whole. A redesign team with a record of the reasons for all the original design decisions has an easier job.

2.4.1 DESIGN PHASE TESTING

As mentioned in Section 2.3.1, a critical aspect of testability is *traceability*. In the case of the design, this means that every part of the design can be linked to a statement in the specification document. A suitably cross-referenced design gives the SQA group a powerful tool for checking whether the design agrees with the specification document and whether every statement of the specification document is reflected in some part of the design.

Design reviews are similar to the reviews that the specifications undergo. However, in view of the technical nature of most design documents, the client usually is not present. Members of the design team and the SQA group work through the design as a whole as well as through each separate module, ensuring that the design is correct. The types of faults to look for include logic faults, interface faults, lack of exception handling (processing of error conditions), and most important, nonconformance to the

specifications. In addition, the review team always should be aware of the possibility that some specification faults were not detected during the previous phase. A detailed description of the review process is given in Section 6.2.

2.4.2 Design Phase Documentation

The major output from the design phase is the *design* itself, which has two parts: the *architectural design,* a description of the product in terms of its modules, and the *detailed design,* a description of each module. The detailed designs are given to the programmers for implementation. Chapter 7 is devoted to the theory of design in general and the design of objects in particular. Design techniques, including object-oriented design, are described in Chapter 13, together with ways of describing the design, such as graphics and pseudocode.

2.5 Implementation Phase

During the implementation phase, the various component modules of the design are coded. Implementation is discussed in detail in Chapters 14 and 15.

2.5.1 Implementation Phase Testing

The modules should be tested while they are being implemented (*desk checking*), and after they have been implemented, they are run against test cases. This informal testing is done by the programmer. Thereafter, the quality assurance group tests the modules methodically. A variety of module testing techniques are described in Chapter 14.

In addition to running test cases, a code review is a powerful, successful technique for detecting programming faults. Here, the programmer guides the members of the review team through the listing of the module. The review team must include an SQA representative. The procedure is similar to reviews of specifications and designs described previously. As in all the other phases, a record of the activities of the SQA group are kept.

2.5.2 Implementation Phase Documentation

The major documentation associated with implementation is the source code of each module, with suitable comments. But the programmers should provide additional documentation to assist in maintenance, including all test cases against which the code was tested, the expected results, and the actual output. These documents are used in regression testing, as explained in Section 2.7.1.

2.6 INTEGRATION PHASE

The next stage is to combine the modules and determine whether the product as a whole functions correctly. The way in which the modules are integrated (all at once or one at a time) and the specific order (from top to bottom in the module interconnection diagram or bottom to top) can have a critical influence on the quality of the resulting product. For example, suppose the product is integrated bottom up. A major design fault, if present, will show up late, necessitating an expensive rewrite. Conversely, if the modules are integrated top down, then the lower-level modules usually will not receive as thorough a testing as would be the case if the product were integrated bottom up. These and other problems are discussed in detail in Chapter 15. A careful explanation is given in that chapter as to why implementation and integration must be performed in parallel.

2.6.1 INTEGRATION PHASE TESTING

The purpose of *integration testing* is to check that the modules combine together correctly to achieve a product that satisfies its specifications. During integration testing, particular care must be paid to testing the module interfaces. It is important that the number, order, and types of formal arguments match the number, order, and types of actual arguments. This strong type checking [van Wijngaarden et al., 1975] is best performed by the compiler and linker. However, many languages are not strongly typed. When such a language is used, checking the interfaces must be done by members of the SQA group.

When the integration testing has been completed, the SQA group performs *product testing*. The functionality of the product as a whole is checked against the specifications. In particular, the constraints listed in the specification document must be tested. A typical example is whether the response time is short enough. Because the aim of product testing is to determine whether the specifications have been implemented correctly, many of the test cases can be drawn up once the specification document is complete.

Not only must the correctness of the product be tested but also its robustness. That is, intentionally erroneous input data are submitted to determine whether the product will crash or whether its error-handling capabilities are adequate for dealing with bad data. If the product is to be run together with the client's currently installed software, then tests also must be performed to check that the new product will have no adverse effect on the client's existing computer operations. Finally, a check must be made as to whether the source code and all other types of documentation are complete and internally consistent. Product testing is discussed in Section 15.4.

The final aspect of integration testing is *acceptance testing*. The software is delivered to the client, who tests it on the actual hardware, using actual data as opposed to test data. No matter how careful the development team or the SQA group might be, there is a significant difference between test cases, which by their very nature are artificial, and actual data. A software product cannot be considered to satisfy its

specifications until the product has passed its acceptance tests. More details about acceptance testing are given in Section 15.5.

In the case of COTS software (Section 2.1), as soon as product testing is complete, versions of the complete product are supplied to selected possible future clients for testing on site. The first such version is termed the *alpha version.* The corrected alpha version is called the *beta version;* in general, the beta version is intended to be close to the final version.

Faults in COTS software usually result in poor sales of the product and huge losses for the development company. For as many faults as possible to come to light as early as possible, developers of COTS software frequently give alpha or beta versions to selected companies, in the expectation that on-site tests will uncover any latent faults. In return, the alpha and beta sites frequently are promised free copies of the delivered version of the software. There are risks involved for a company participating in alpha or beta testing. In particular, alpha test versions can be fault laden, resulting in frustration, wasted time, and possible damage to databases. However, the company gets a head start in using the new COTS software, which can give it an advantage over its competitors. A problem occurs sometimes when software organizations use alpha testing by potential clients in place of thorough product testing by the SQA group. Although alpha testing at a number of different sites usually brings to light a large variety of faults, there is no substitute for the methodical testing that the SQA group can provide.

2.6.2 Integration Phase Documentation

The documentation produced during this phase consists of the commented source code for the project as a whole, the test cases for the project as a whole, and the user manual, operator manual, database manual, and other manuals.

2.7 Maintenance Phase

Once the product has been accepted by the client, any changes constitute maintenance. Maintenance is not an activity grudgingly carried out after the product has been installed on the client's computer. On the contrary, it is an integral part of the software process that must be planned for from the beginning. As explained in Section 2.4, the design, as far as is feasible, should take future enhancements into account. Coding must be performed with future maintenance kept in mind. After all, as pointed out in Section 1.3, more money is spent on maintenance than on all other software activities combined. It therefore is a vital aspect of software production. Maintenance must never be treated as an afterthought. Instead, the entire software development effort must be carried out in such a way as to minimize the impact of the inevitable future maintenance.

A common problem with maintenance is documentation, or rather a lack of it. In the course of developing software against a time deadline, the original specification and design documents frequently are not updated and, consequently, are almost useless to the maintenance team. Other documentation such as the database manual or the operating manual may never be written, because management decided that delivering the product to the client on time was more important than developing the documentation in parallel with the software. In many instances, the source code is the only documentation available to the maintainer. The high rate of personnel turnover in the software industry exacerbates the maintenance situation in that none of the original developers may work for the organization at the time when maintenance is performed.

Maintenance frequently is the most challenging phase of software production for the reasons stated previously and for the additional reasons given in Chapter 16.

2.7.1 Maintenance Phase Testing

There are two aspects to testing changes to a product during the maintenance phase. The first is checking that the required changes have been implemented correctly. The second aspect is ensuring that, in the course of making the required changes to the product, no other inadvertent changes were made. Therefore, once the programmer has determined that the desired changes have been implemented, the product must be tested against previous test cases to make certain that the functionality of the rest of the product has not been compromised. This procedure is called *regression testing.* To assist in regression testing, it is necessary that all previous test cases be retained, together with the results of running those test cases. Testing during the maintenance phase is discussed in greater detail in Chapter 16.

2.7.2 Maintenance Phase Documentation

A major aspect of the maintenance phase is a record of all the changes made, together with the reason for each change. When software is changed, it has to be regression tested. Therefore, the regression test cases are a central form of documentation for this phase.

2.8 Retirement

The final phase in the software life cycle is retirement. After many years of service, a stage is reached when further maintenance no longer is cost effective.

1. Sometimes, the proposed changes are so drastic that the design as a whole would have to be changed. In such a case, it is less expensive to redesign and recode the entire product.