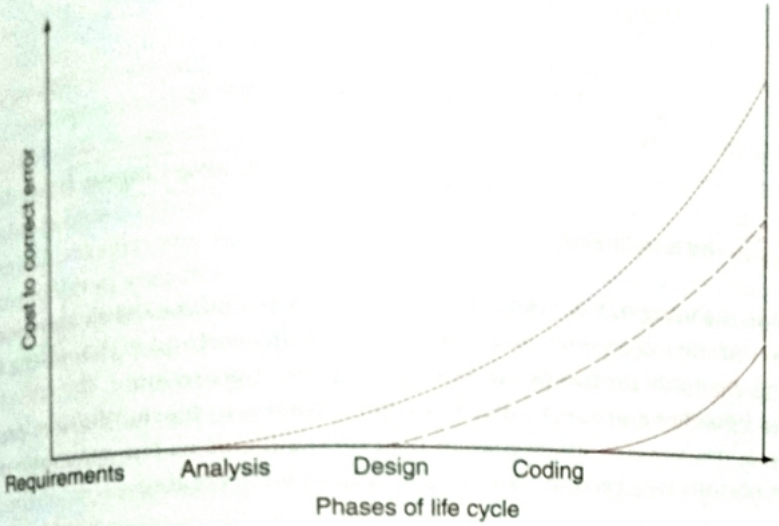

FIGURE 10.10 Cost to correct errors detected at different phases of life cycle.

ensure that the design is near-perfect and can go for coding. This approach is followed by many organisations and for large projects – with close to 500,000 software lines of code. In these cases, almost 50% of the effort within the project is dedicated to analysis and design. Second, the project managers may follow a variation of the traditional Waterfall Model with having short durations of analysis and design followed by coding and testing. The entire system is thus not analysed and designed completely, but is done in manageable parts. This approach has led to a number of alternative approaches to software development, which are described below.

10.2.3 Alternatives to the Waterfall Model

10.2.3.1 *Rapid Prototyping*

Prototyping in software development implies creating a working model of the system, not the entire system. Rapid Prototyping is a formal method by which the system is built in a sequence of cycles of the Waterfall Model. The phases of analysis, design, coding and testing are compressed into a shorter time frame. A partial system is built, and then this is used for analysing more detailed requirements for the next prototype. Detailed documentation is not done. A larger prototype is then built, again going through the phases of analysis, design, coding and testing. The working prototype that is now obtained is larger and more complex than the previous one, and is closer to the eventual system that is desired. The first prototype is now discarded, and the second one becomes the basis for the next round of prototyping (see Fig. 10.11).

The advantage of rapid prototyping is that it allows incremental development of the system, thus, reducing the amount of initial time spent on analysis and design, and giving the systems team a chance to see whether some requirements are feasible or not, and what more can be designed into the system. The prototypes allow for a

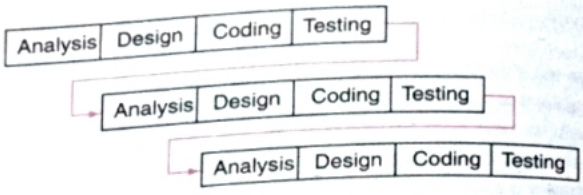

FIGURE 10.11 Rapid prototyping.

better understanding of the eventual system and also reduce the analysis and design flaws that may be introduced. A disadvantage of this method is that effort is wasted in building the initial prototypes that are discarded. Furthermore, the reduced documentation becomes a problem once the final system is in place. Without the detailed documentation, systems maintenance becomes difficult as the maintainers do not know the reasons why certain parts of the system were created.

10.2.3.2 The Spiral Model

The Spiral Model was developed in 1987 to address some of the concerns raised by system developers about the Waterfall Model. The Spiral Model uses an iterative approach, where prototypes are built successively in a spiral of expanding capabilities. It was originally designed for large projects, those that require considerable time and effort to build and would run into millions of lines of code. The model is not an alternative to the Waterfall Model as much as it is an enhancement.

The stages of the Spiral Model are depicted in Fig. 10.12. The model is depicted as a spiral that increases its diameter along the orthogonal axes. The spiral starts in the top left quadrant, and this is the first stage. The spiral is kicked off by a felt need for a system and an assessment of the alternative ways in which the need could be addressed. Objectives of the system are estimated along with estimates of the budget.

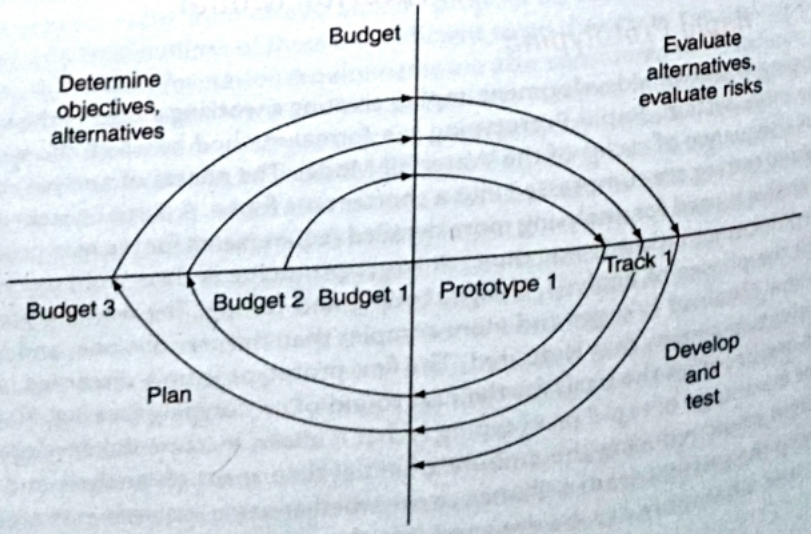

FIGURE 10.12 The spiral model of software development.

In the next stage, as depicted in the top right quadrant, the alternatives are evaluated. For each alternative, the evaluation involves estimating the risk associated with the development. For example, if the two alternatives for a system for setting up a database are – using a proprietary software or using an open source alternative, then these two choices would be assessed for their risks. Risk assessment involves assigning estimates to the construction, use, value and eventual success of the two alternatives. It involves identifying uncertainties in the project and manner in which this could be dealt with. This stage may include benchmarking the project against other similar projects, interviewing developers and users regarding the effort, and simulating or visualising the conditions under which the software would be used. Once a risk profile has been created, the prototype is developed.

At the next stage, represented by the lower right quadrant in the figure, the prototype is tested and evaluated. This provides a detailed idea of the requirements of the system, now that the prototype can be tested, as also the features that are missing. Using the detailed requirements as well as the risk profile determined in the previous stage, the next stage is started. This stage is marked by the lower left quadrant in the figure. Here plans for the next iteration in the spiral are made. These plans are based on the lessons learned from the first prototype and its evaluation. The plan includes the life cycle of the product, including all stages of analysis, design, testing and coding.

As the next cycle of the spiral is initiated, the activities of setting objectives, determining alternatives and setting a budget are initiated again, followed by the next stage of determining the risks for the alternatives. A new prototype is built, followed by its evaluation and a new set of plans for the next iteration.

The Spiral Model of iterations ends when the risk analysis shows that the system is well enough understood to go into the final round of analysis, design, coding and testing. The requirements are known well enough now, and so the focus is on the design of the eventual system and the testing.

The Spiral Model is a risk-driven model that explicitly models risk and uncertainty in the development process. The entire focus is on understanding and controlling risks to reduce the chances of eventual failure of the system.

10.2.4 Agile Methods

The late 1990s saw the evolution of software development methods known collectively as the *Agile Methods*. Drawing inspiration from manufacturing quality processes, these methods relied on short analysis–design–coding–testing cycles, with close user interaction and flexible team compositions to build software. The underlying emphasis in the Agile methods is on reducing and controlling risk. Shorter life cycles ensured that bugs and design flaws were detected early, and the user involvement throughout the project ensured the requirements were addressed adequately.

The Agile manifesto, a document outlining the beliefs of the developers who invented the Agile techniques, defines the fundamental propositions of these methods. The Agile methods rely on creating programs quickly and with close interaction with the customer. Requirements are never frozen and change is welcomed to ensure that the customers' needs are attended to right through the end of the project. Developers are given autonomy and responsibility to work on their own and seek out

quality measures for the project. The Agile methods also require suitable working conditions like an open working environment and a limit on the number of hours that can be worked on in a week so that developers remain motivated and productive.

Different agile methods have different means by which they address the objectives of the manifesto. Two of the methods that have gained popularity are Extreme Programming and Scrum. These methods are described below.

10.2.4.1 Extreme Programming

Extreme Programming has gained wide popularity among the Agile methods. It embodies most of the objectives of the Agile manifesto and has been demonstrated to be effective for small- to medium-scale projects. As a method, Extreme Programming has a few core ideas.

1. **Stories:** All activities are initiated by stories that users tell about what they want from the system, how they go about doing their work and what processes they follow among others. These stories are then written on index cards and the writing is done rapidly. The stories constitute the initial analysis for the system.
2. **Short releases:** Modules for the software are released quickly and often. A module is determined by the users and the developers jointly, based on the highest priority of features wanted. Here, users have a priority as to what they want to see first, and the developers guide this choice based on cost and resources available. The releases are made as frequently as every day.
3. **Testing:** Releases of code are tested as soon as they are created. In fact, while planning for the module, the test sets are created and a module is considered completed after the tests are satisfied. This manner of testing is known as *unit* testing.
4. **Iteration:** The iterations in Extreme Programming are based on the features needed in the releases. The system is put into production the moment the release is made, or it is placed in the central repository (a repository is a central collection of code). The units are integrated and the whole package is tested. If it fails then the latest changes are reversed. If the changes are accepted then this starts a new iteration with another story and another set of features.
5. **Refactoring:** The design of the system evolves and changes as the system is built up from the modules. Refactoring, or changes, is done as the integrated system needs them.
6. **Pair programming:** All the coding is done by a pair of developers. One writes the code and the other 'looks over his/her shoulder' to see how the code fits in with the rest of the system and how it can be thoroughly tested. Programmers have to be highly skilled in doing their task and they have to work rapidly.
7. **Work environment:** The entire team is located in one room, with the programmer pairs at a centre table/computer. At least one customer is always available for evolving the features and to write the stories. In total, 40-h work weeks are desired. However, occasional overtime is permitted, but not for successive weeks.
8. **Metaphors:** Both the developers and the users work with metaphors to visualise their system. Metaphors are words or phrases that resemble some desired

feature or attribute, but are not exact. Metaphors help seek out different meanings that the users may have about what they want, and explore different possible ways of achieving the same.

Extreme Programming is successful because it allows the developers autonomy in understanding the problem clearly and designing the system accordingly. The close interaction with the customers also enables continuous dialogue and an understanding of implicit and tacit codes that are often missed in formal meetings. Furthermore, its emphasis on continuous testing and integration ensures that the evolving system is always working and meets the customers' needs.

10.2.4.2 Scrum

The Scrum method relies on the idea of evolving the requirements for the project and the needed activities on the basis of work completed rather than on plans. The project proceeds in a series of *sprints*, which are fixed durations of 2 or 4 weeks. In each sprint particular and defined objectives are met; a tested system is built for that portion alone, and what further needs to be done is determined on the basis of the completed work.

Scrum involves defining a number of roles for the project members and the creation of a number of artefacts.

1. **Product owner:** He/she is a customer or client who determines what the system should be and how it should behave. He/she is a part of the team and is always involved.
2. **ScrumMaster:** He/she is a leader for the Scrum team, who acts as a project coordinator.
3. **Team member:** He/she is a member of the team, who participates in building the software.
4. **Product backlog and sprint backlog:** These are requirement documents that state what has to be achieved for the project and for each sprint. Each backlog is essentially a high-level requirement specification that is fleshed out and detailed during the sprints.
5. **Burn down chart:** This is a record of backlogs that have been completed. For a sprint, the burn down chart shows what has been completed on a daily basis, giving the team a focus for what needs to be achieved.

During a sprint, meetings to review the progress and create the burn down logs are highly structured and controlled (see Fig. 10.13). A daily meeting is held for exactly 15 min in the same place and at the same time (first thing in the morning). All may attend the meeting but only the 'pigs' are permitted to speak. Pigs are the product owner, the ScrumMaster, and the team members, or all those who are committed to the project; commitment implies they are accountable to the progress of the project. Others are not permitted to speak, and are termed 'chicken' as they are only involved with the project but do not have a commitment. Meetings determine what has been achieved during the previous day's sprint and what will be achieved today. The ScrumMaster determines if there are any obstructions to the progress, and if so, these are resolved. After this meeting, the team members meet to discuss various areas in which collaboration is required.

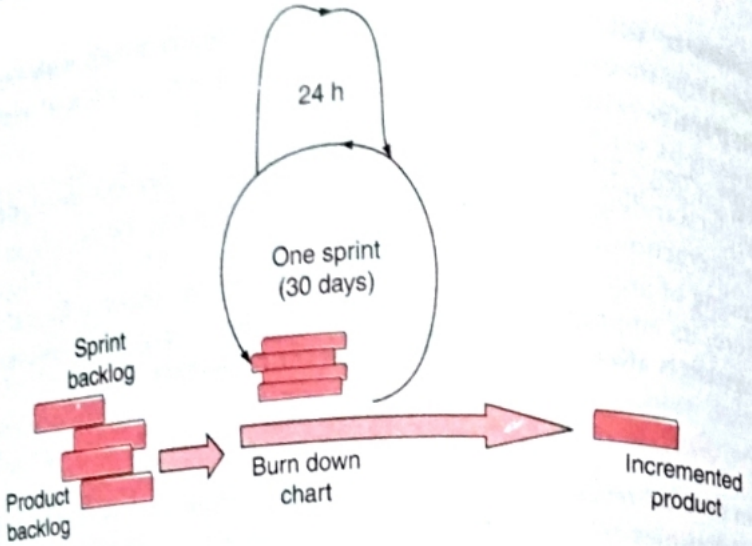

FIGURE 10.13 The scrum process.

Scrum meetings are also held after each sprint to establish a new set of backlogs based on what has already been achieved. A Scrum ends after it is realised that the system has met all the objectives stated by the product owner. Each sprint produces a working version of the software, albeit incompletely. As the sprints progress, the software grows and begins to meet all the requirements.

The difference between Scrum and Extreme Programming lies mainly in the diffuse manner in which Scrum proceeds. Requirements are specified at the high level and get focused and tightened only as the development progresses. Furthermore, requirement changes are considered to be inherent in the product evolution process, and after each sprint changes can be quite significant.

10.3 SOFTWARE PROJECT MANAGEMENT

The frameworks or methodologies for software development present a structured manner in which software can be constructed. However, the frameworks do not include a large number of issues that arise as a project unfolds. Some of these issues are common to projects of all kinds, whether for software development or for some other activity, and some are widely different. For software project management, the most substantial issues have to do with finding requirements, managing a team, ensuring the final product works, and trying to stay within budgets of time and money.

The project management issues arise in all contexts of software development, regardless of the particular methodology followed.

10.3.1 Top Management Support

Possibly one of the most important issues for software project management in the current organisational environment is that of obtaining and retaining top management support. Top management buy-in to the project is required from the start, even while the project is being conceptualised. The requisite resources have to be allocated and then supported for the duration of the project.

The distinct advantages of having the top management involved are many. Large and medium-sized projects, in particular, require integration with other systems within the organisation, which the top management can facilitate. The vision and high-level requirements for the software are provided with assistance from the top management, ensuring that the software integrates well with the larger goals of the organisation. Vendor involvement and selection are facilitated with negotiations at the highest levels of the organisation. Also, for the organisation to accept the new system, the top management is indispensable for providing the motive and rationale for the new system, ensuring personnel training and reassignments.

10.3.2 Team Composition

The team that is involved with the overall design, implementation and maintenance of the system has to be constituted carefully. Teams have to have a balance of skills and experience in project management. Ideally, teams should last the duration of the entire project cycle, however long the cycle may be. Rules for team composition are hard to define, as they are so dependent on context. However, some avoidable issues can be highlighted as they are known to be serious problems.

1. The project leader should be an individual who commits to the entire duration of the project. A replacement should be clearly identified and groomed, in case the leader has to be away for unavoidable circumstances.
2. Skills of team members should be carefully documented and made known. This will ensure that others know who can do what, and so they can call upon them when needed.
3. Team members should be comfortable working with each other. More importantly, if there are known problems with individuals, they should be addressed early in the project by the management.
4. The leader should have overall accountability and responsibility for the project. He/she should be the person one is to turn to if things are not going well. The leader should also be the person to be held responsible if the project fails.

In modern, large organisations, it is often the case that teams are distributed across the globe and work jointly using electronic networks for communication. System building assumes a more difficult form in such situations, although the above critical concerns have to be addressed.

10.3.3 Vendor Selection

Organisations nowadays use vendors extensively to build and help maintain their systems. Vendors are often hard to identify, particularly those with special skills. Industry associations provide directories that list software vendors and their skills. In India, the National Association of Software and Services Companies (NASSCOM) is a well-known association that maintains a database of vendors. This is a good

starting point. Most client firms will make it a point to evaluate a vendor thoroughly before they are selected. Some points for evaluation are:

1. The number of projects the vendor has completed of a similar nature in the past. This should also include the size of the projects, the time taken for completion, the kind of technology used and whether the clients were satisfied with the project.
2. The skills base of the vendor depends on: (a) How many employees they have with particular skills, (b) how many have been with the firm for long, (c) what has been the training imparted to them and (d) how have they gained experience.
3. A site visit to the vendor's premises to see the working conditions as well as the assets the vendor has to use for the work. This would indicate how deeply the vendor can commit to the project.
4. For long-duration and large projects, the financial position of the vendor should also be evaluated. Some firms demand bank guarantees from vendors against successful completion of the project. Only financially sound vendors are able to produce such guarantees. As small vendors cannot produce such guarantees, the client organisation has to rely on financial records to assess the vendor.

It is also important that tenders and call-for-proposals floated to ask for bids from vendors should not be biased to favour one or the other vendor. The wording of the tenders has to be vendor-neutral, particularly avoiding words or phrases that identify with a particular vendor.

10.3.4 Project Estimation

At the initiation phase of projects, one of the most difficult problems managers face is that of estimating the size of a project. This entails estimating the resources that will be required by the project and the effort involved. The entire planning for the project is based on these estimates and if they are wrong, the project success is jeopardised.

Project estimates can be over or under. If they are overestimated, it means that the project will eventually take less resources than planned for. If underestimated the projects will take more resources than planned for. Some argue that it is better to overestimate than to underestimate, as excess resources are not a problem but less resources are. For vendors who are bidding for a project, overestimation leads to a high cost estimate for the project thus making their bid potentially uncompetitive. If their bid is an underestimate, which they also win then they will incur higher costs for the project than they had bid for, thus cutting into their profits. For client organisations, overestimation leads to committing resources more than required, thus leading to higher costs and management problems; underestimation leads to scarce resources that have to be procured, possibly at a high expense.

10.3.5 Function Point Analysis

This is a technique to estimate the size and effort required to build an information system. The estimate is given by one measure, called the *function point*, which shows how complex the project is. For example, a project with a function point of 200 can be estimated to be built in a certain number of person hours. This estimate is independent

of the platform and software tools used. The function point estimates are based on two broad categories of computing requirements:

1. **Data functions:** It indicate the variety of data input and output needs and how they have to be built.
2. **Transaction functions:** It indicate the processing required within the system. Transactions are categorised into various sorts such as communication, manipulation, writing, etc., and the efforts required for these are then estimated based on some industry benchmarks.

Function point analysis has been shown to be quite effective in providing an estimate of the size and complexity of the system. However, it cannot account for project-based specificities such as changes in the project scope and requirements, skill levels of developers, availability of platforms and so on.

Chapter Glossary

Business process Any business activity that involves movement of information between people, groups or departments.

Business analyst A person who has the responsibility of understanding and documenting business processes.

Flow diagrams A high-level illustration of tasks and the movement of data.

Flowcharts A diagram that depicts the logic and flow control of a computer program.

Data flow diagrams A high-level diagram that illustrates data flows between people, processes and data storage.

Use cases A technique for conducting business process analysis that involves descriptions of tasks, people and information that flows between them.

Prototype A scaled down version of a software program that is used for analysis of business processes.

Stub testing A way of testing software that involves examining how responses from independent modules will impact the complete package.

Beta test Testing of software that is nearly complete, has all the features and can be conducted under conditions similar to the real environment for which it has been created.

Parallel cutover Releasing a new software in an environment where the old methods of doing tasks are also kept alive.

Scope creep Increase in the requirements of a software module, much after the requirements phase is over.

Unit testing A method of software testing that involves testing independent modules extensively.

Review Questions

1. What is a business process? Give two examples of business processes.
2. Review the elements of a flowchart. Then draw a simple flowchart for the following problem: A customer sends an e-mail requesting some information. If the request is about a product, send a product brochure to the customer by e-mail. If the request is for something else, refer to the request to a representative by forwarding the e-mail.
3. Review the elements of a data flow diagram. How are they different from a Flowchart?
4. Draw a data flow diagram for the following problem:

   A system at a library, called the Library Information System, allows a user to check out a book.