Chapter 10

Software Release

Software release is an important activity that occurs once the software product is fully developed. This chapter presents various aspects related to software release.

In Chapter 10, we will learn

- What a software release is
- What a release cycle is for Waterfall model–based projects
- What a release cycle is for agile model–based projects
- What activities are performed in a software release

Software release is the stage when the software product has been fully developed and tested and is ready to be deployed at the customer's site. However, before that happens, you need to ensure that everything has been checked and there are no loose ends that need to be tied.

After a software product is developed, it needs to be released so that it can be used by end users. There are many ways a software can be released.
For software projects using Waterfall methodology, a software release happens when the software product is fully developed and tested. For these software products, there can be a final release or a final release following alpha release and beta release.

For software projects using agile methodology, there are many mini software releases. A software product vendor can align these mini releases based on market demand. It is easy to do it because software development is geared towards developing software product incrementally. The market demand is synchronized with software product development. Thus a software product can have many releases within even short span of time.

User & technical documents

Some software product being built needs to be integrated with some existing software product. If the integration requires to be done at many points, then effort required for integration will be high. If integration is only at a few points, then effort required for integration will be low.
Documentation must be prepared for the software product for both technical people and for end users. Whenever a maintenance needs to be done then the technical documents are needed. Good technical documents help in doing maintenance work better.
Good user documents are needed so that users can use the software product effectively. User documents can be in any form including screen shots, descriptive text, help videos etc.
If the software product contains any software defects, then users may not be able to use the software product effectively. Some of the transactions cannot be done due to these software defects. However, if there are alternative ways available which can be used to perform the same transactions then explanation should be provided to users. Users will be able to perform those transactions using these alternative ways. These alternative ways are known as walk arounds.

On Waterfall projects, if the software vendor realizes that a developed software product needs thorough testing before a final release of the software product could be made then the vendor can opt for an alpha release of the software product. In alpha release, the untested software product is given to customers for free. The customers use this software product and communicate to the vendor about defects found during use. The vendor fixes those software defects. If the vendor finds that the software product is now defect free then the vendor can make a final release of the software product.
If the vendor realizes that the software product needs some more testing then the vendor can make a beta release of the software product. This beta release is again distributed to customers for free. The customers

will use this software product and if any software defects are found then it is communicated to the vendor. After the beta release, all the reported defects will be rectified by the software vendor. Finally, the vendor may release the final version, which may be a paid version and will have almost no software defects (Figure 10.1). Alpha or beta release of a software product are a great way to test an untested software product quickly.

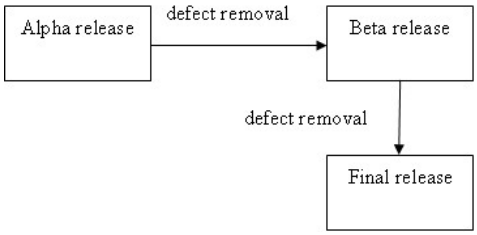

Figure 10.1 Alpha–beta–final release cycle.

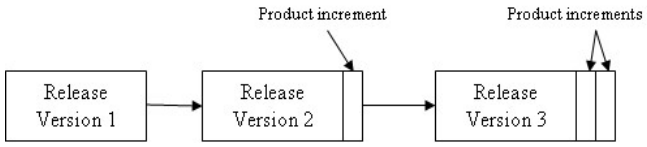

Figure 10.2 Release cycle for incremental product development.

When software products are developed incrementally, there will be a large number of releases. These releases can be both minor and major. All these minor and major releases are dictated by the market/customer demands. The marketing team or the customer gets market feedback and provides their input to the vendor about the required features of the product. The project team then develops these new features in a timeboxed (iteration) plan. The release date is fixed by the marketing team. Thus, the project team is left with a fixed time to develop the required product features. The release cycle and different versions that are built in an incremental development model are depicted in Figure 10.2.

Software release on agile projects happen differently as compared to Waterfall projects. Software release on agile projects happen after an increment happens in the software product. In each iteration, a few software product features are added to existing software product. This incremented software product is fully functional and so it is a new release of the software product. On agile projects, software product is incrementally built till all required software product features have been added. This results in many releases of the software product.

Comparison parameter	alpha – beta- final release	Incremental release
Purpose	To improve quality	To build product incrementally
Product size change	Remains constant	Increases in each release
Product quality	Improves	Remains same
Development model	Waterfall	Agile with incremental product building
Time to market	Fast	Fast
Product size	Small to even large	Small to even large
Product versions	Few	Few to many

Table 10.1 Some Comparisons between Alpha–Beta–Final Release and Incremental Release

Alpha – beta - final releases for Waterfall projects serve a different purpose as compared to incremental releases of agile projects. The table given above shows the comparison of releases of software products in Waterfall projects compared to those for agile projects.

Purpose of having alpha-beta-final release is to improve quality of the software product. Incremental releases of software products are done to build a software product incrementally.

Software product size remains constant after alpha-beta-final release. But product size increases after each incremental release.

Software product quality improves after alpha-beta-final release. But product quality remains the same after any incremental release.

One of the goals for introducing alpha-beta-final release is to bring the software product in the market fast by getting the software product tested by thousands of customers. By incrementally building a software product again one of the goals is to introduce the software product fast in the market.

Alpha-beta-final release is suitable for small to even large software products. In the early days of agile movement, agile methodology and incremental building of software products was suitable for small software products. Now after maturity of this methodology, even large software products are being built incrementally.

There are only 2 to 3 releases of the software product in case of alpha-beta-final release route. But in case of incremental releases, there could be many releases of the software product.

User training

User training is one of the important functions performed during release of any software product. To effectively use a software product, users must know how it works. User training ensures that users get to know all about the software product and how to use it.

User training can be conducted by the project team in live sessions. Videos with screen capture can also be used for this purpose.

Deployment

Deployment of the software product involves installing the software product at client site as well as populating data in the installed product. Once all files and supporting infrastructure is deployed then a user acceptance testing can be done to check if the software product is working as per requirement specifications.

If a software product is distributed over the internet, then deployment instructions can be published on the same website from where the software product can be downloaded.

Software migration

Commercial off the shelf (COTS) and Software as a Service (SaaS) software products also need software migration if the customer was previously using a legacy system. There is customer data in the legacy system which need to be ported in the new COTS or SaaS system. Since the format and structure of data saved in the legacy system will be different from what is expected in the COTS or SaaS system; a lot of work is involved in migrating this data in the COTS or SaaS software product.

Data migration from the legacy system to the COTS or SaaS software product happens in live environment as the customer cannot shut the legacy system and users keep using it while migration of data takes place.

Once data is fully migrated to the new system, cut over activities take place. Using a phased strategy users move to the new system one by one. The legacy system is cut off completely and all users start using the new system.

Chapter 11

Software Maintenance

Software maintenance is an important activity that occurs once the users start using the product. This chapter presents the details of that activity.

In Chapter 11, we will learn

- What software maintenance is
- What the steps in a maintenance process are
- What the strategies of software maintenance are
- What software maintenance types are
- What reverse engineering is

Once a software product is released and implemented at client site, users start using it. When any software defects are found during use, they need to be fixed. Customers also may need changes in existing functionality of the software product after some time. Software maintenance is the process of making changes in a software product which is already in use.

On waterfall projects, maintenance projects are separate from software development projects. But on agile projects, software maintenance can go hand in hand along software development. It is because, the software development team can work on building the software product incrementally and at the same time the software maintenance team can work on a maintenance project for older versions of the software product.

When a software product is being used by users then it is known as a production instance of the software product. SaaS vendors need to keep production instances of its software product up and running so that their customers and their users can use the software product. They monitor production instances on a daily basis to ensure that it is up and running and users do not face any problems in using them. If they find any problems then it is fixed immediately.

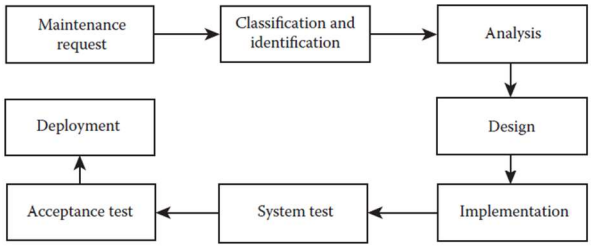

Figure 11.1 Maintenance project life cycle.

When a maintenance project is taken to fix software defects or enhance or add software product features, a maintenance process is used to do all maintenance related work.

The maintenance work is planned when end users make a request for maintenance. All the reported software defects are classified first. The project team identifies the places in the source code which need to be changed to fix a software defects. Analysis is then done to find out best way to fix those software defects. Then software design is created (if new functionality to be added in the software product) or modified (if existing functionality need to be changed). Based on the changed software design, source code will be written or changed (implement the software design). Once all source code has been written then a system test will be conducted to see if all the defects have been fixed and the software product is working fine.

Once the project team releases the software product after maintenance work, end users perform user acceptance testing to see if the software product is working fine. If the software product is working fine then it is deployed so that end users can use it.

Maintenance methodologies

Software maintenance can be done using adaptive, perfective, corrective, or preventive maintenance methodologies.

Adaptive maintenance is used when a technology on which the software product depends becomes obsolete. For example, an operating system becomes obsolete and the software product needs to run on a new operating system then the software product may need to ported to the new operating system.

Perfective maintenance is done when users want a changed functionality in the software product.

Corrective maintenance is done when users find software defects in the software product and they need to be fixed.

Preventive maintenance is used to make the software product robust so that if something goes wrong during running of the software product, the software product should still be able to run without any problems. If a user provides a wrong input on a user interface and if the resulting error can be handled in the source code then instead of the software product shutting down or crashing, it will be able to handle this exception and still will be running. Preventive maintenance is incorporated during the software construction by writing source code which can handle exceptions when it is running.

Maintenance strategies

Software maintenance can be done using any of the strategies including fix immediately, fix periodically, fix using a maintenance project, using a patch etc.

Fixing each detected software defect immediately can be done in environments such as SaaS, incrementally built websites etc. The software development team is also responsible for maintaining production instances of the software product here. Fixing each defect immediately is easy for them. Generally there will be no downtime of the production instance as fixing just one defect at a time does not require a shutdown of the production instance.

Periodic fixing of software defects is possible when the software development team is also responsible for maintaining production instances of the software product also. The project team will keep a list of reported software defects and when it is suitable to take maintenance, they will do it. Users are advised in advance about unavailability of production instance for maintenance work.

Fixing software defects using a maintenance project is done for behind the firewall installations of software products. A project team is formed by the software vendor or the service provider to take a maintenance project to fix defects.

Many software vendors have thousands of customers using their software product. Fixing defects individually for each customer is not feasible. In such cases, the software vendor creates software patches. After applying these patches, software defects are fixed in software products.

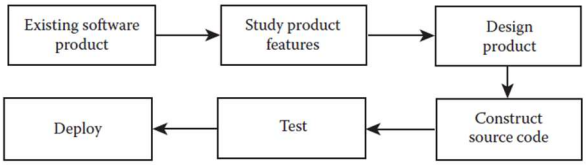

Figure 11.2 Reverse engineering process.

Reverse engineering is used for modifying a software product where source code of the software product is not available. The project team runs this software product and finds out its software features. Based on seeing how the software product features work, the project team creates a software design. This software design is then implemented by writing the source code. The completed software product is then tested. The fully tested software product is then deployed.

One area of concern regarding reverse engineering is the issue of copyrights. If there are no copyright violations, then it is not a problem. But if any copyright violations are involved then reverse engineering should be avoided.

Chapter 12

Configuration and Version Management

Configuration and version management tools are widely used in the software industry to organize several artifacts that are being developed as part of a software project. They are discussed in this chapter.

In Chapter 12, we will learn

- What software configuration management is
- What software version control is
- What a software build is
- How configuration and version control systems work
- What continuous integration of source code is

Configuration management is all about keeping and managing files containing source code and supporting documents. Version control on the other hand is all about managing various versions of different artifacts which get generated during a software project. All these artifacts are in form of electronic files and they are saved on computers.

On any software project a large number of people work as project team members. Managing these files and their various versions across computers of team members is a difficult task given the large of artifacts and their versions.

Automated configuration and version control systems (CVS) are used to avoid human errors while managing versions of artifacts. CVS also allows to have a central repository of files so that they can be configured and controlled well.

If you look at Figure 12.1, you will realize that there could be many versions of the same artifact located both on the local computer of the developer and on the server. Many versions of the same artifact are generated on almost all projects because of the incorporation of change requests into that artifact.

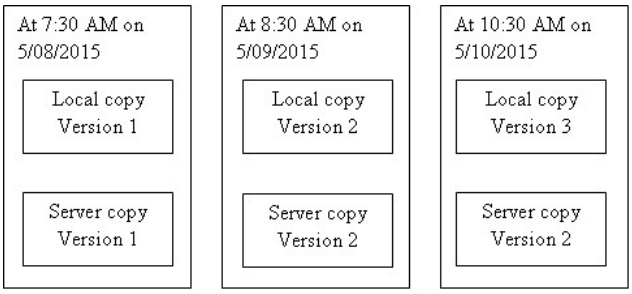

Figure 12.1 Many copies of an artifact on a local computer and on the server.

Files containing artifacts are created by project team members. First a team member will save the file on his/her local computer. Then the team member checks in the file on the server. Later when a change is needed in the artifact then the team member will make the change in the file and save it on his/her local computer. The team member will save this file with a different name to distinguish it from the original file. Thus there are more than one version of the file on the local computer. When this new version of the file is checked in the server then the server will also have more than one version of the file.

It is also possible that there are more than one copy of a file on a computer. This copy file may or may not have the same content as the original file. Nevertheless, many copies of a file on a computer system can create problems. In the figure given above, you can see that by mistake wrong version of an artifact can be saved on the server. This type of human error can lead to rework and other issues.

Limitations of file systems

File system of any operating system can manage files, directories and various versions of a file well. But on software projects, a configuration and version control system (CVS) system is used because of following reasons:

- A file system of an operating system can manage files only on one computer. On software projects, since many team members work, it is difficult to manage all the files residing on computers of team members. A central system is needed which can do this job.
- A file system of an operating system does not have configuration management capability. CVS systems provide this capability.
- File synchronization between the local computer (computer of a team member) and the server is difficult in file system of an operating system.

Version management system, its branches and files

Creating a new branch in a CVS system is very similar to creating a new directory in the file system of an operating system.

A version control system is similar to a file system of any operating system. The difference is that while a file system displays files and directories of one computer, a CVS system displays files and branches created inside the version control system. There are directories (branches) and files. You can create a branch to keep relevant files in one place. You can have many branches and sub branches in the CVS system. The files (inside this directory and subdirectories) are like leaves. A CVS system with all its branches, subbranches, and files forms a similar tree structure.

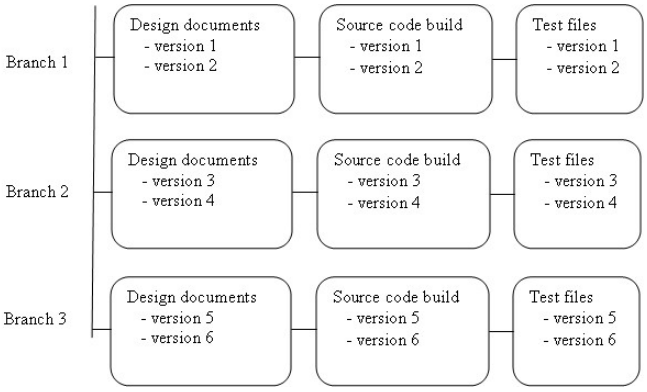

Figure 12.2 Each branch represents one or more versions of all the artifacts in CVS.

In Figure 12.2 you can see this tree structure. There is the main trunk on which the branches (Branches 1–3) are located (the trunk is the vertical straight line, shown at the left-hand side of this figure, to which these branches are attached). On each of these branches, there are files or documents (Design documents version 1, version 2, etc.). These documents are like the leaves (i.e., leaf nodes) of that tree.

Software build

During software development, software developers write source code and save it in source code files. There are many software developers on any software project. All the source code files created by these software developers need to be integrated to create the complete source code for the software product. The integrated files of a software product is known as a software build.

Integration problems can happen if interfaces of components defined in the source code are not defined properly.

To ensure that there are no integration problems, the software build should always be in compiled state. Compilation of the software build should always be done after a file is integrated with the software build. If a file is tried to integrate with the software build and due to integration problems the compilation fails, then the entire software build itself fails and is termed as in failed state. If another file is tried to be integrated with this failed software build, then even if this new file has no integration problem still the software build will be in failed state. So if a software build fails after a new file is integrated then this new file should be rectified of integration errors and should be integrated again. If there are no compilation errors this time, then the integration was successful.

Configuration & version control system with client and server parts

The earliest CVS systems used to have a central repository on a central server where project team members used to upload files from their local computers. This arrangement used to create a problem. It was never known in advance if a file containing source code created by a software developer will be integrated successfully with the software build on the server.

Later distributed CVS systems were developed. These systems consist of one server part and many clients can have access to this server part. The complete file repository can be replicated from the server part to each client part. The compilation can be done both on the server part as well as on any client part. A developer can compile his/her source code file on the client part (installed on his/her computer). If there are integration errors then the developer can fix integration problems in his/her file. Once compilation is successful on the client part then the developer can upload his/her file to the server part. Since the client part of the software build is exact replica of the server part, if integration of a file does not create any problems on the client CVS, it will also not create any integration problems on the server CVS.

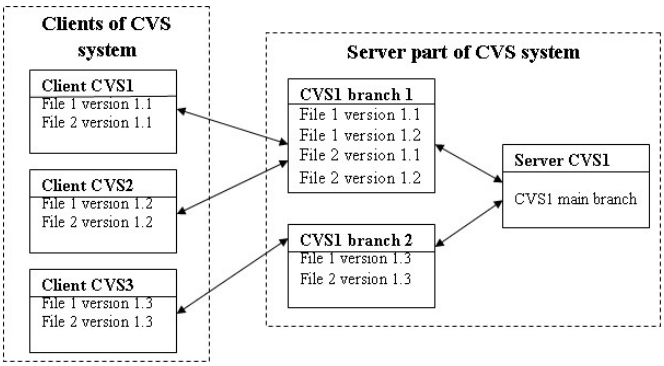

Figure 12.3 Centralized CVS system (client and server) with branching.

Figure 12.3 depicts a centralized CVS system. This figure also shows the server part of the CVS system and its branches as well as the client CVS systems connected to the server CVS system. How the file synchronization happens between a branch of a server CVS system and a client CVS system will be explained later. A branch can contain many versions of the same file or a single version of each file. It depends entirely on the need of the project.

Merging of files in a CVS system

Merging and synchronization of files between client and server part of CVS systems is a very common activity. When a project team member creates and saves a file on his/her computer (client part of CVS), he/she also uploads this file on the server part of CVS.

If a change is made in a file in the client CVS and is uploaded to the server CVS, then if it is uploaded on the same location where a file with the same name already exists (uploaded previously) then the CVS system will display a message that the file already exists. In this situation, either the changes can be merged with the existing file or a new version of the file can be created with a different file name.

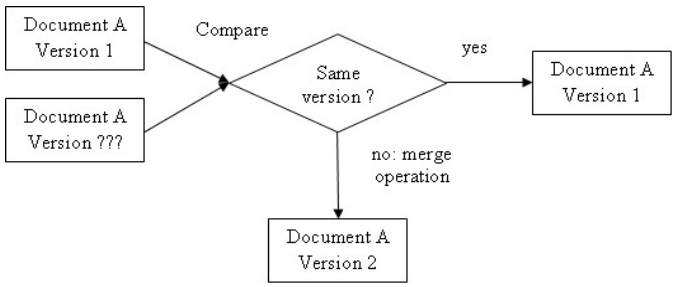

Figure 12.4 Document comparison and synchronization in a decentralized CVS system.

Figure 12.4 depicts the file synchronization process. If a file needs to be submitted at a location in the CVS system where there is already a file of the same name, then a file comparison will take place. If both files have

the same content, then it can be said that both files are synchronized. On the other hand, if contents inside these files are different, then the contents will be merged.

Continuous integration – smoke test in a configuration management system

Continuous integration of software build is the best integration strategy for any software project. Integration of files consisting of a software build is a real concern as many integration problems occur when files are tried to get integrated with a software build. Continuous integration minimizes this risk by integrating one file at a time with a software build.

Generally, a software build system is divided into two parts: client software build and server software build. First, a developer checks in his or her source code on the client software build that is installed on his or her local computer. If the client software build compiles and integrates the piece of the newly developed source code without any error, then the developer will check in his or her source code on the server software build.

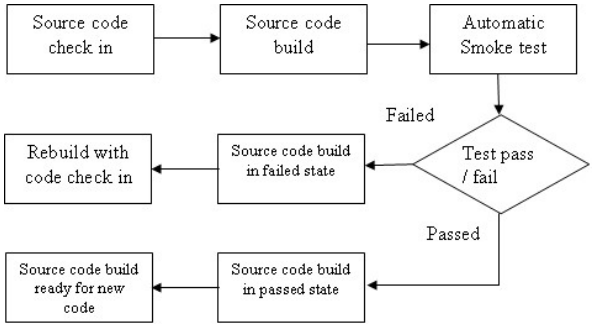

Figure 12.5 Continuous integration with automatic smoke test.

Smoke tests are useful with continuous integration. In Figure 12.5, we can see the smoke test process for software build. The smoke test process starts when a developer checks in his/her file or source code on the server software build. The moment the file is integrated with software build; the automatic smoke testing tool will start. If integration failed, then the software build will be in failed state. The software developer will have to rectify the error and check in the file again on the software build. If the smoke testing tool finds that the file integrates with the software build, then the software build will be in passed state. When a software build is in passed state then it is ready to accept new files for integration.

Chapter 13

Software Project Management

Software project management is all about planning, execution, and control of the tasks involved in software product development. This chapter presents the software project management activities.

In Chapter 13, we will learn

- What software project management is
- Project management in Waterfall model–based projects
- What a project schedule is
- What are the techniques to plan, monitor, and control software projects
- Project management in agile model–based projects
- What team management is on software projects
- CPM/PERT, Gantt chart, and earned value management in project management
- What customer management is for software projects

A project is a process through which an objective is achieved by using a series of predetermined tasks. There is a start and end date inside which the project must be started, executed and finally should come to an end. A project uses a finite amount of resources and time.

Project management consists of project planning, monitoring and controlling so that the project objectives can be accomplished in a planned manner.

Software project management is the use of project management processes to build a software product. Software project management uses software engineering processes inside the project management processes for the purpose of building a software product.

Some software projects use Waterfall based software engineering methodology while some other software projects use Agile based software engineering methodologies.

Waterfall methodology is a truly a plan driven approach to building software products. Unfortunately, a plan driven approach is not suitable for building all kinds of software products. Some of the reasons include ambiguous software requirement specifications, long waiting period and changing software requirements.

Agile methodology evolved to take care of the challenges which could not be tackled by Waterfall methodology. Agile methodology is based on incremental software development to tackle ambiguous software requirement specifications, long waiting period and changing software requirements. In Agile projects incremental approach and quick response to market demand define the approach to software development.

Project scope

Project planning in Waterfall projects take into account fixing of scope for the project first. The scope is fixed by finding the size of the software product to be created. The size of the software product to be developed will determine the amount of work to be done on the project. If for any reason the scope is changed then amount of work to be done on the project will increase (as shown in Figure 13.1).

The other aspect about project scope is the quality level to be achieved for the software product. Higher product quality will lead to higher amount of work for the project and thus additional scope.

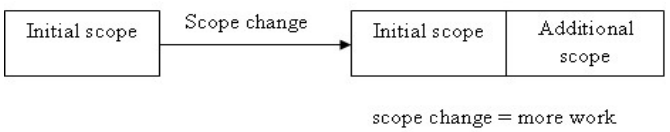

Figure 13.1 Scope change results in more work.

Work breakdown structure & project schedule

Once project scope is fixed for a project then project tasks needs to be defined. It is done using work breakdown structure (WBS). WBS is a complete listing of all tasks which will be planned and executed on a project. The project itself can be considered a large task. This task need to be broken down into manageable and meaningful tasks.

Table 13.1 shows a project schedule. The columns WBS no., Summary task, and Subtask are part of the WBS. All the dates are in month/day/year format. The remaining columns in this table are added to create a project schedule. Note that these columns are not part of the WBS. They are in fact part of the project schedule. We will learn about project schedules later.

Table 13.1 Project Schedule Table

WBS no.	Summary task	Sub task	Start date	End date	Resource	Dependency
1	Design		10/10/2015	12/12/2015		
2		Module 1 design	10/10/2015	11/11/2015	Andy	
3		Module 2 design	10/12/2015	12/12/2015	Rachel	
4	Source code		11/12/2015	02/05/2016		
5		Module 1 build	11/12/2015	01/05/2016	Larry	2
6		Module 2 build	12/13/2015	02/05/2016	Ashley	3

Some tasks can also be broken down in such a way that they can be performed in parallel. This will reduce time and thus will lead to shorter project duration. A software design task can easily be broken down into 2 or more sub-tasks. These tasks can then easily to completed into parallel. This will ensure that the entire software design work is completed in less time.

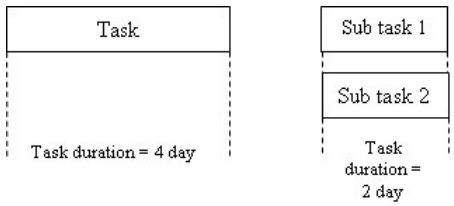

Figure 13.2 Task duration for a complete task and the task duration after division of that task.

Figure 13.2 shows the duration of a complete task and the reduction in the duration of that task after dividing it into two subtasks. In the first case, when the task was not divided, only one person was working on it. In the second case, after dividing that task into two subtasks, one person worked on each of these subtasks.

Gantt Chart showing project schedule

The project schedule can also be plotted in the form of a Gantt chart. Figure 13.3 depicts a Gantt chart.

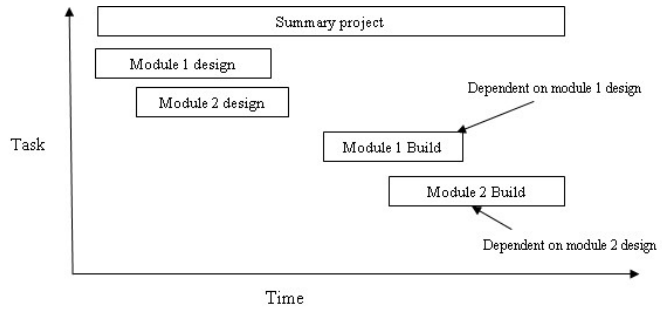

Figure 13.3 Gantt chart for project scheduling.

Figure 13.3 contains a Gantt chart in which the tasks are plotted against time. This Gantt chart is an alternative way of showing a project schedule. The benefit of a Gantt chart is that it is graphical; thus, understanding a project schedule is much easier with it. Figure 13.3 shows the four project tasks for a sample project.

The WBS is used to create the project schedule. The project schedule is created by considering effort required for completing a task, skills matching & resource allocation, productivity of a resource, resource availability and dependency between tasks.

If productivity of a resource is to write 20 lines of source code in a day and the task is estimated to be a size of 100 lines of source code, then the task will run for 5 days. Suitability of skills also need to be considered as not all people have the same level of skills and experience. Allocation of a resource to a task also depends on economic considerations. If a task has no dependency and there is enough time to complete it then a lower

skilled person can be assigned to this task as it will be cheaper. But critical tasks should always be assigned to best available person.

If a task needs to be completed before another starts, then scheduling should consider this dependency between these 2 tasks. Similarly, if a resource is available for a certain period then the task on which this resource is assigned needs to be scheduled inside this time window.
When all schedule constraints are resolved then the project schedule can be drawn. A project schedule can be drawn either in form of a schedule table or a Gantt Chart. A Gantt Chart is more useful as it depicts all tasks in a graphical manner which is easier to understand.

Finally, the project cost can be calculated by adding costs for all task. The labor rate and the time a resource works on the task will give the cost for each task.

Baseline project schedule

In a project schedule, some buffers are also placed to take care of emergencies or project risks.

Critical path Method / Programme Evaluation & Review Technique (CPM/PERT) is used to create the project schedule. In this method all tasks are placed in a line as per their start and end dates. The shortest line from start date of the earliest task to the end of the latest task becomes the project duration.

Once a project schedule is finalized then it needs to be frozen. A frozen project schedule is known as a baseline project schedule.

When a project is executed, it is tracked against this baseline project schedule.

Project planning for Agile projects

All projects have to deal with time, cost, scope, and quality constraints. In traditional projects, all the terms related to time, cost, scope, and quality are considered and agreed upon before the project starts. However, in agile projects, things are a bit different. The cost, quality, and time are fixed. However, the scope is kept open. This is because the agile product development is supposed to be responsive to the customer needs.

Even though Agile projects are not plan driven, nevertheless some planning is needed to execute these projects. On Agile projects, planning has shifted from project level to lower levels. On Agile projects a project plan is divided into major release plans. A major release plan is then divided into minor release plans (also known as Timeboxes or iterations). Then each minor release plan is divided into daily plans.

Detail and firmness of a plan depends on at what level it is located. A major release plan is fuzzy and tentative. Thus a major release plan date is not firm. The timebox plans are firm as far as dates are concerned. But scope for a timebox is not firm. Daily plans are made for achieving tasks to be completing for a day. The daily plans are thus firmed in all respects.

Timeboxing is a concept to pack all project tasks inside a tight and fixed schedule. The scope for a timebox is not firmed. All project tasks are given a priority level. High priority tasks are done first and completed inside the timebox. If time permits, then less priority tasks can be taken. If time does not permit, then these less priority tasks can be taken in the next timebox.

Traditional project planning techniques like Gantt Chart, PERT/CPM, Goldratt's Critical Chain Method etc. do not work on Agile projects.

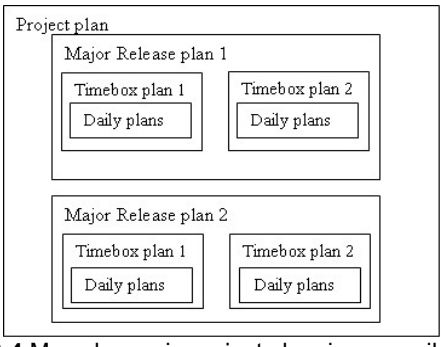

Figure 13.4 Many layers in project planning on agile projects.

In Figure 13.4, we have depicted many layers of project planning on agile projects. The topmost-level planning, "Project plan," is the planning for the entire project. In reality, we never do project planning in agile projects because we always carry out planning at the lower levels. The next level of planning is known as major release planning. Major release planning is done to ensure that the product reaches the customers as per the agreed upon time frame.

Gantt Chart with baselined & actual project task progress

The Gantt chart can provide information for task progress in terms of its schedule. Using a Gantt chart, you can maintain the baseline dates and actual dates for any task.

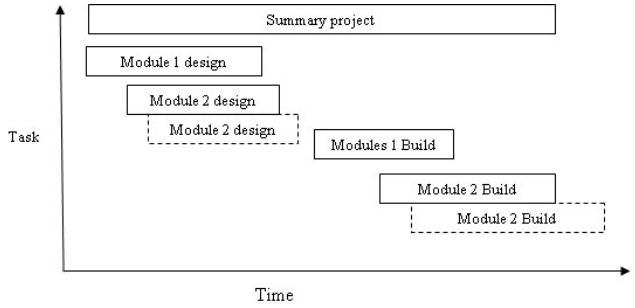

Figure 13.5 Gantt chart with project progress measurement.

In Figure 13.5, a Gantt chart is shown. The tasks with solid lines are the baselined tasks. They have baseline start dates and end dates. The tasks with actual start dates and end dates are shown with dotted lines. Gantt charts also show the percentage progress of any task.

Project monitoring & control for Waterfall projects

Project monitoring & control is needed when a project starts executing. Tools like Gantt Chart, Earned Value Management (EVM), CPM/PERT, Goldratt's Critical Chain Method etc. are used for project monitoring & control on Waterfall projects.

On a Gantt Chart, if a task is slipped from its schedule then it is shown in dotted lines. The actual start date and end date are also shown.

Goldratt's Critical Chain Method uses buffers for critical tasks. A critical task is a task which if slipped will result in slipping of the project schedule. For these critical tasks, the buffers are monitored carefully. If a critical task slips and consumes more than 50% of its related buffer, then it is considered that the project schedule is in danger of getting delayed and some action need to be taken to ensure that the project schedule does not slip.

Time buffer between critical tasks

When a project plan is created, time buffers are introduced in the plan. These time buffers are provided after critical tasks for providing cushion if a critical task clips. Some amount of these time buffers are consumed when such a critical task slips. Thus even though a critical task slips, still the project plan is still on track; thanks to these time buffers.

In fact, in Goldratt's Critical Chain Method, critical tasks progress is not even measured. It is the time buffer provided for these tasks which is measured. If such a time buffer is consumed more than a predefined value, then the project manager must act and take some action to keep the project schedule on track.

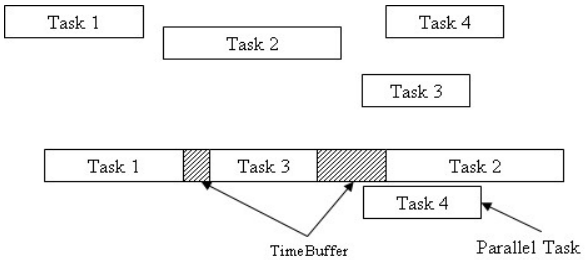

Figure 13.6 Project tasks aligned for creating a critical path.

In Figure 13.6, you can see the time buffers provided between Tasks 1 and 3 as well as between Tasks 3 and 2. Time buffers for critical tasks allow the project team to finish the project within the schedule even when some critical tasks get delayed to some extent. Thus, the time buffers provide a mechanism in controlling a project.

Earned Value Management (EVM)

Earned Value Management (EVM) is one of the popular methods for monitoring & controlling projects. In EVM both schedule and cost are tracked for projects.

There are 2 concepts of schedule of work and cost of work which help in understanding how EVM works. Schedule of work is related to the actual amount of work done versus the planned amount of work in a time period. Cost of work is the actual amount of cost incurred versus the planned budget in a time period. Related to these 2 concepts are schedule variance and cost variance when the planned work or cost is different from the actual results.

These 2 concepts need to be related to each other to understand EVM. It can happen that actual cost incurred in a time period is more than planned budget but the amount of work done in the same period is less than planned.

To understand how EVM works, you need to understand the concept of percentage amount of work performed on a task in terms of both time elapsed and budget spent. We discuss the concept of EVM in Figures 13.7 through 13.9.

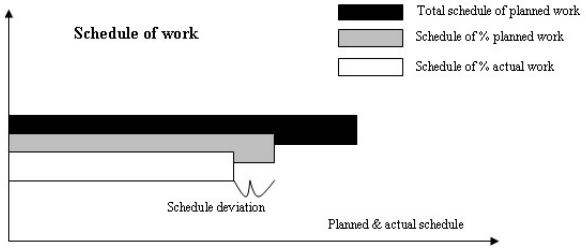

Figure 13.7 Amount of work completed in terms of time (schedule) consumed.

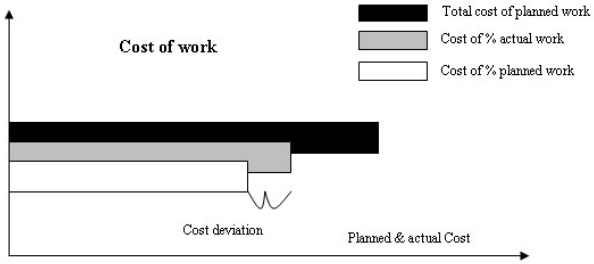

Figure 13.8 Amount of work completed in terms of budget consumed.

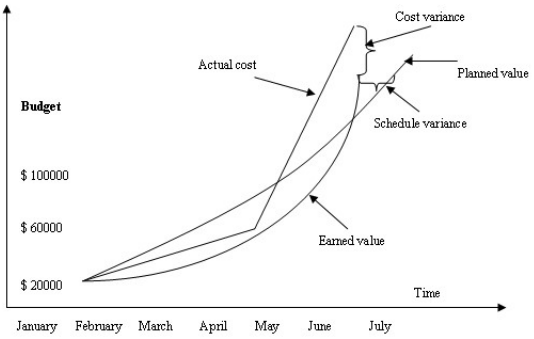

Figure 13.9 EVM showing the budget and schedule variance.

As we discussed EVM is essentially concerned with finding schedule and cost deviation from baseline project budget and schedule. If any deviations are found, then they need to be corrected by taking appropriate measures.

Project monitoring & control for Agile projects

Agile projects are not plan driven. Thus there is no baseline against which these projects can be tracked or monitored or controlled.

However, some planning is done at timebox (iteration) level. So monitoring and control is possible at this level. If speed at which the project team implements user stories can be established, then this speed can be tracked for each timebox. This speed is known as velocity. For example, if it is established that a project team can implement 5 user stories in 10 days long timeboxes then for each timebox, amount of project work can be

planned and tracked. This project work can also be monitored and controlled as the baseline for this work can be easily fixed. Any user story can be allotted a number of user story points based on the size and complexity of the user story.

Project team management

Apart from schedule and cost, a project manager also needs to manage the project team. Project team members performance on the project is crucial for the success of the project. A motivated and success driven project team makes all the difference on the project. If performance of a team member deteriorates then the entire project will get impacted as a delayed task will lead to delay in the entire project work. Thus the project manager must keep tracking each project task and also keep a track of performance of each team member.

Customer management

A project manager is also responsible to keep a good rapport with the customer. Meeting customer expectation is important. Customers want timely delivery, visibility into the project and good quality of the software product. If something is going wrong on the project, then the project manager should be able to explain why it is so and how it will be corrected. A satisfied customer will make the job of managing the project easier for the project manager.

Supplier management

If a project has suppliers (or service providers) on the project, then the project manager also needs to manage them. A good service level agreement should be in place so that the service providers are bound by this agreement to deliver the expected services at the right time. A part of the project will be delivered by the supplier. The project manager needs to work with the supplier to ensure that this part of the project work is delivered by the supplier on time. A good arrangement with a supplier should be that there is an open communication channel with the supplier and the project team gets updates from the supplier on daily basis. This will ensure that there are no misunderstandings or communications gaps which may result in rework on any project work.