B.Tech

TCS 042

EIGHTH SEMESTER EXAMINATION, 2007-2008

SOFTWARE PROJECT MANAGEMENT

Note : Attempt all questions.

1. Attempt any four parts of the following : 5×4=20

Q.1. (*a*) What is Software Project Management (SPM)?

What are the various activities of SPM?

Ans. Software project management encompasses the knowledge, techniques, and tools necessary to manage the development of software products. This curriculum module discusses material that managers need to create a plan for software development, using effective estimation of size and effort, and to execute that plan with attention to productivity and quality. Within this context, topics such as risk management, alternative life-cycle models, development team organization, and management of technical people are also discussed.

Management activities

Proposal writing.

Project planning and scheduling.

Project costing.

Project monitoring and reviews.

Personnel selection and evaluation.

Report writing and presentations.

Q.1. (*b*) What do you mean by software project? How does it differ from any other project?

Ans. Right about now you are probably asking, "What is a project anyway?". Clearly dictionary definitions emphasize that a project is a planned activity. If we accept that a project is planned, we must also assume that we can figure out how we are going to get the task finished before we start. Planning is thinking about doing something before you even begin to do it. Other related activities, such as routine maintenance, might be done so often that everyone will know exactly what needs to be done. In these cases, you man not think planning is needed but procedures should be documented to ensure consistency. This will greatly help any newcomers to the job.

The techniques of other types of project management apply to Software Project Management but products of software projects have characteristics that make them different or even unique. For example, if an artifact, such as a building, is being built the progress is easily seen. With software the progress isn't so clearly visible. Software Project Management Bob Hughes says is the process of making visible that which is invisible.

For the dollars spent, software products have more complexity than other engineered artifacts have. How easy software can be changed is often seen as one of its greatest strengths. This means, however, that that where the software interfaces with a physical or organizational system, the expectation is that the software will change to accommodate the other components rather than the other way around.

As you can see, from this introduction to the subject, Software Project Management is a topic you need to study in some depth. If you would rather not take a course on the topic you might consider finding a way to acquire the book Software Project Management Bob Hughes wrote. I am sure you can find it at your local library, bookstore, or

can order it online. The quality of the information will be well worth it.

Q.1. (*c*) In reference to project planning, discuss the terms 'milestones' and 'deliverables'?

Ans. Managers need information. As software is intangible, this information can only be provided as document that describes the state of the software being developed. Without this information, it is important to judge progress and cost estimates and schedules cannot be updated. When planning a project, a series of milestones should be established where a milestone is an end-point of a software process activity. At each milestone, there should be a formal output, such as a report, they can be represented to management. Milestone reports don't need large documents. They may simply be a short report of achievements in a project activity. Milestones should represent the end of a distinct, logical stage in the project. Indefinite milestones such as "Coding 80% complete" which are impossible to validate are useless for project management.

A "deliverable" is a project result that is delivered to the customer. It is usually delivered at the end of some major project phase such as specification, design, etc.

Deliverables are usually milestones but milestones need not to be deliverables. Milestones may be internal project results that are used by the major project manager to check project progress but which are not delivered to the customers.

To establish milestones, the software process must be broken down into basic activities with associated outputs.

Q.1. (*d*) What do you mean by stakeholders? Who may be the stakeholders in a software project?

Ans. The entire process of software projects is strongly stakeholder-driven. It's their wishes, fears, dreams, their stakes (hence the name) that determine the course of the projects. A stakeholder can be a project team member, an employee of the user organization, or a senior manager. Virtually, it can be anyone, as long as they have something to do with the project. The stakes are the crown jewels of the holders. They stick to them, they defend them, they are married to them. They also make up the words to formulate their expectations. The individuals will take all actions necessary to defend their stakes, or to get near the realization of them.

Stakes can be in two directions: fears or wishes. With the first there is a stake to lose, with the second there is something to gain. Either way, stakes are sacred things; anyone, including a project manager, should not try to mess with them. Again, in order to do anything with the stakes of the holders, the project manager should be the greatest negotiator he possibly can be.

Q.1. (*e*) What is meant by Software Project Estimation? Discuss.

Ans. Effective software project estimation is one of the most challenging and important activities in software development. Estimation is one of the cornerstones of effective project planning: effective project planning and control is not possible without a sound and reliable estimate. In recent SPC surveys (source: SPC Direct Insights webinar series) only 18% of software professionals identified estimation as a key strength or their organization, while 70-90% of those surveyed experienced financial impacts on their businesses, lost competitive advantage, and delays in getting benefit from their plans and initiatives due to bad estimates. Even today it is clear that the software industry in general doesn't estimate projects well and doesn't use estimates appropriately. We suffer far more than we should as a result and we need to focus some effort on improving the situation.

Under-estimating a project leads to under-staffing it (which often results in staff burnout), under-scoping the quality assurance effort (running the risk of low quality deliverables), and setting too short a schedule (resulting in loss of credibility as deadlines are missed).

For those who figure on avoiding this situation by generously padding the estimate, over-estimating a project can be just as bad for the organization! If you give a project more resources than it really needs without sufficient scope controls it will use them up! This is known as Parkinson's Law: "*Work expands so as to fill the time available for its completion*". The project is then likely to cost more than it should (a negative impact on the

bottom line), take longer to deliver than necessary (resulting in lost opportunities), and delay the use of your resources on the next project.

This paper explains the fundamental principles and practices of software project estimation and provides tips and guidance for applying estimation practices in various scenarios including "New Development" projects, Maintenance & Enhancement projects, Small projects, "New Domain" projects, and Agile projects. It also discusses the capabilities of specialized software estimation.

Q.1. (*f*) What is the requirement of having a stepwise project planning?

Ans. A Stepwise Project Management Method (SPMM) is developed to resolve the Project Management (PM) issues. The template of a new stepwise PM method is derived from a Project Management, Inc. (PMI) 19-Step PM method. The 19 Steps were divided into five groups by PMI: scope/requirements, Work Breakdown Structure (WBS), predecessor networks, assigning people to tasks and tracking results. As we view this from a perspective on the principles of management, there exist four basic management functions: planning, organising, leading and controlling. Having rearranged the PMI five groups into four groups of the basic management functions and further skipped the unnecessary steps, a stepwise PM method that is rearranged under the management functions is finally created. The stepwise PM method will be discussed in detail.

2. Attempt any four parts of the following : **5×4=20**

Q.2. (*a*) What do you mean by Work Breakdown Structure (WBS) in context to software project?

Ans. A Work Breakdown Structure is a results-oriented family tree that captures all the work of a project in an organized way. It is often portrayed graphically as a hierarchical tree, however, it can also be a tabular list of "element" categories and tasks or the indented task list that appears in your Gantt chart schedule.

In planning a project, it is normal to find oneself momentarily overwhelmed and confused, when one begins to grasp the details and scope of even a modest size project. This results from one person trying to understand the details of work that will be performed by a number of people over a period of time. The way to get beyond being overwhelmed and confused is to break the project into pieces, organize the pieces in a logical way using a WBS, and then get help from the rest of your project team.

The psychologists say our brains can normally comprehend around 7-9 items simultaneously. A project with thousands or even dozens of tasks goes way over our ability to grasp all at once. The solution is to divide and conquer. The WBS helps break thousands of tasks into chunks that we can understand and assimilate. Preparing and understanding a WBS for your project is a big step towards managing and mastering its inherent complexity.

The WBS is commonly used at the beginning of a project for defining project scope, organizing Gantt schedules and estimating costs. It lives on, throughout the project, in the project schedule and often is the main path for reporting project costs. On larger projects, the WBS may be used throughout the project to identify and track work packages, to organize data for *Earned Value Management (EVM)* reporting, for tracking deliverables, etc.

Q.2. (*b*) (*i*) Discuss in brief:

(*i*) Gantt Chart

(*ii*) Milestone Chart.

Ans. Gantt Charts are important tools that help you plan and manage complex projects.

They help you work out the order in which tasks need to be carried out; allow you to identify the resources needed to complete the project, along with the times when these resources will be needed; help you work out the quickest possible time in which a projeet can be completed; and help you identify the "critical path" for a project. This is the sequence of tasks that must be completed on time if you are to complete the project by a particular date.

When a project is under way, Gantt Charts help you to monitor whether the project is on schedule. If it is not, they help you to pinpoint the remedial action necessary to put it back on schedule.

These are all essential activities if you are going to manage projects successfully.

Project Management Issues and Considerations (Issacons) IAC # 1302

Milestone Chart

Graphical representation shows

milestone dates

Identifies key points in the Project's life span

The bars are not necessarily visible

Project Management Consulting Services @November, 2001 *AEW*

A milestone chart depicts key events along a timescale.

A milestone chart traditionally used triangles to depict a specific event.

A milestone on a milestone chart Gantt can depict a specific event or a culmination of events.

Milestones on a milestone chart Gantt can be shown in various colors or with markings that indicate status.

A milestone chart Gantt is usually used for top level reporting so management does not become bogged down in the minutia of the project or projects.

Q.2.(c) What are the time estimates used in PERT? Briefly discuss their significance.

Ans. Software project estimation is a tough job. If you get it right, great; but if you get it wrong - that's an issue. And, it does not matter whether your team will employ an agile methodology or the Waterfall during the implementation phase. At the estimation stage there are an enormous number of unknowns that are waiting to bury you.

For anything but the most trivial, repetitive tasks there is no simple way. Except, math and the experience of other people. In this blog post I will describe an industry-standard Program Evaluation Review Technique (PERT) that can help plan projects more realistically and even calculate expected error of estimation.

The mantra of mediocre project-managers is the infamous "most IT projects fail". It is used as a standard excuse for the projects that go months over planned deadlines and thousands of dollars over the estimated budget. Granted, the number of unknowns in a complex IT project is very large and, let's be honest, most clients do not provide requirements documents at even a reasonable level of completeness. Is that a good-enough excuse for poor planning?

What should we do? Give up? Accept defeat? Some "rational" clients go as far as simply doubling an estimate provided by a software team and using that for projections. It may be fine when a project is for $100K, but not for a multi-million project.

Q.2.(d) What do you understand by the term "Critical path"?

Ans. Critical Path Analysis and PERT are powerful tools that help you to schedule and manage complex projects. They were developed in the 1950s to control large defense projects, and have been used routinely since then.

As with **Gantt Charts,** Critical Path Analysis (CPA) helps you to plan all tasks that must be completed as part of a project. They act as the basis both for preparation of a schedule, and of resource planning. When you are managing a project, they allow you to monitor achievement of project goals, and help you to see where remedial action needs to be taken to get a project back on course.

The benefit of using CPA over Gantt Charts is that Critical Path Analysis formally identifies tasks which must be completed on time for the whole project to be completed on time (these are the tasks on the critical path), and also identifies tasks which can be delayed for a while, if resources need to be redeployed to catch up elsewhere.

The disadvantage of CPA is that the relation of tasks to time is not as immediately obvious as with Gantt Charts. This can make them more difficult to understand.

Q.2.(e) What are the ways to organize the personnel of a software development organization?

Ans. Today, more and more people are purchasing personal organizer software to help them better their lives. However, here's the question you need to ask yourself before going out and buying one for yourself: do you really need a personal organizer to keep your life in order? First of all, when

used properly, a personal organizer can be an absolutely invaluable tool in anyone's live. Without one of these, it can be very difficult to keep track of all your responsibilities for any given day. However, manufacturer's competition for these machines is fiercer than ever. With all the model choices available, it is becoming increasingly difficult to choose the right one for you. Here are some questions to ask yourself before purchasing one. What will the organizer's main function be? Do you need to keep track of your daily activities better, track your schedule more efficiently, listen to audio books, or do self development reading? These questions are very important in helping you determine which model is right for you; most different personal organizer makes have different strengths, and you need to find the one that matches its capabilities to your needs.

Does it need to have a built in microphone and sound? Not all organizers carry this feature. If you don't need it, then don't pay the premium price for it just because it's available. What size does the organizer need to be? Does it need to be able to be able to fit into your pocket, or do you plan on using a book style cover that's capable of holding other items? What kind of software do you plan to run with it? The answer to this will determine what type of machine you need and how much power it must have.

Obviously, the less power you need, the less expensive the electronic organizer will come. Therefore, before purchasing, make sure that its' capabilities are something you absolutely need. Don't simply purchase a personal organizer because you want the most gadgets and gadgets available even if you don't really need them. Finally, what kind of memory card do you use in your digital camera? If you will need an organizer with this feature, find a personal organizer with a similar type of card.

This feature will allow out to take pictures on your camera and simply transfer them quickly and easily onto your PDA. Of course, some organizers come with a built in camera, so this may not be necessary. These are just a few simple questions to ask yourself when searching for . The bottom line is, these machines are becoming more and more popular all the time, and when used effectively can help you keep your life organized very well. However, be ca eful when making a purchase-don't simply buy a model with more features than you need. Find one that will fit your life's needs, and only then should you make your purchase.

Follow these tips, you will find the organizer that's right for you and your budget. Most importantly, your life will be in order and you will be able to reach your full effectiveness in whatever you do.

Q.2.(*f*) Differentiate between 'Process WBS' and 'Product WBS'.

Ans. A WBS for a large project will have multiple levels of detail, and the lowest WBS element will be linked to functional area cost accounts that are made up of individual work packages. Whether you need three levels or seven, work packages should add up through each WBS level to form the project total.

Product or Process Oriented?

The WBS was initially defined as a product oriented family tree, however subsequent definitions have introduced more flexibility-so a WBS can also be deliverable or process oriented. Your WBS can be built on nouns or verbs. If the results of your project are primarily verbs, then a verb based or process based WBS may make more sense. If your WBS is to be product or deliverable oriented, then you can start by thinking of the WBS as a parts list for the ultimate end-items of your project. This link will give a simple illustration of a *product or process* based WBS orientation. These differences are not shown to tell you what is the right way for your project, but just to familiarize you with the distinctions, so you can think about them and choose what's best for your project.

3. Attempt any two parts of the following : **10×2=20**

Q.3.(*a*) What do you mean by 'Code Review'? Also discuss the difference between Code Inspection and Code Walkthorugh.

Ans. Code review is systematic examination (often as *peer review*) of computer *source code* intended to find and fix *mistakes* overlooked in the

initial development phase, improving both the overall *quality of software* and the developers' skills.

Code reviews can often find and remove common *vulnerabilities* such as *format string exploits, race conditions, memory leaks* and *buffer overflows*, thereby improving software security. Online software repositories based on *Subversion* with *Trac, Mercurial, GIT* or others allow groups of individuals to collaboratively review code. Additionally, specific tools for collaborative code review can facilitate the code review process.

Automated *code reviewing software* lessens the task of reviewing large chunks of code on the *developer* by systematically checking source code for known vulnerabilities.

There are many examples of where it is claimed that adopting code reviews improved a software development project. Notable examples include:

• *Blender* (*software*), a 3D graphics design package greatly improved by the *open source* development *community*.

• *Linux kernel*, once a hobby project of *Linus Torvalds*, it is now reviewed and improved by hundreds of programmers worldwide.

These claims can be hard to evaluate because each project was implemented only once; it's not possible to know for sure how the project *would have* turned if it hadn't adopted code reviews, or if it had instead adopted other quality control measures.

A Code Walkthrough is where the "author" of the document or code is walking the "inspectors" through the code in one meeting. In my company, this method works better with requirement documents than code.

A Code Inpsection/Code Review is where the "inspectors" get a copy of the code before hand and looks for issues. After a certain period has been meet for them to have reviewed the code such as a week then they gather for a meeting. Depending upon how structured the meeting is depends upon if it is a code Inspection or Review.

In our group we use Code Reviews (Peer Reviews) mainly. We use Peer Reviews more often because they are less formal yet for us they have a sense of team building and a learning arena for newer programmers.

I would say that it is necessary for all descent sized applications to have several walkthroughs or Inspections. Of Course, one thing to remember is that Inspections/Walkthroughs do not replace testing and visa versa.

Q.3.(*b*) What is meant by 'Error Tracking'? Why is error tracking an excellent management practice?

Ans. Most programmers are rather cavalier about controlling the quality of the software they write. They bang out some code, run it through some fairly obvious ad hoc tests, and if it seems okay, they're done. While this approach may work all right for small, personal programs, it doesn't cut the mustard for professional software development. Modern software engineering practices include considerable effort directed toward software quality assurance and testing. The idea, of course, is to produce completed software systems that have a high probability of satisfying the customer's needs.

There are two ways to deliver software free of errors. The first is to prevent the introduction of errors in the first place. And the second is to identify the bugs lurking in your code, seek them out, and destroy them. Obviously, the first method is superior. A big part of software quality comes from doing a good job of defining the requirements for the system you're building and designing a software solution that will satisfy those requirements. Testing concentrates on detecting those errors that creep in despite your best efforts to keep them out.

In this article we'll take a look at why the issue of software quality should be on the tip of your brain whenever you're programming, as well as discussing some tried-and-true methods for building high-quality software systems. Then we'll explore the strategies and tactics of software testing.

Let's turn our attention to software testing techniques. Testing is the second level of error control: detection. Don't confuse testing with debugging, which is the third step in error control. The goal is to find the problems, not necessarily to resolve them at this time.

Historically, testing has occupied the greatest

fraction of the time and effort associated with software development. A big reason for this is the lack of attention traditionally paid to system specification and design. Of course, you could test any program until doomsday and still not be completely sure that it will work right 100% of the time. But if we use thorough, structured test procedures, we can have more confidence that we've eradicated most of the bugs from the final product.

The testing process begins with development of a testing plan. In principle, this can be done by either the software developer (you), or by an independent testing person or group. The difference is that the developer has a conflict of interest: he wants to demonstrate that the program works properly, while the independent tester wants to find the flaws in the software. The best compromise is to have the developer handle unit testing and have somebody else address integration and acceptance testing. We'll talk more about these different test phases shortly.

The bulk of the test script should be written prior to the coding phase. This may seem silly to you, but I've found it to work very well. One of the hardest aspects of software development is for the system designer to share the customer's vision of what the end product will be and do. The person who writes the test script has to visualize how the software is to behave whenever a particular set of inputs is received or a particular action specified. This rigorous thought process helps greatly to uncover any fuzziness or errors in the design plan. It also helps to catch any oversights in going from system specification to system design. Once when I was writing a test plan, I couldn't remember writing the process narratives that would implement two functions I needed to test. Sure enough, I had overlooked those requirements. The test plan helped me catch my errors and correct them very early in the system development effort.

Your test script should include a bunch of individual tests that specify particular input values and action selections, and the expected results of the program execution under those conditions. Specify values for inputs that cover these situations: the smallest and largest allowable numbers; missing entries; values out of the legal range; random legitimate values; and incorrect data types. Design test paths that will ensure that each statement in the code is executed at least once. Make sure that your error-handling routines work right, and that your validation of input data catches all the sorts of errors that might realistically be encountered.

Similarly, devise tests for logical control flow constructs (IF/ELSE IF/ELSE, CASE/SELECT, etc.) to make sure that each conditional branching statement is executed at least once in every possible direction. You can't possibly cover every possible combination of paths through even a very small program, but you should convince yourself that control is transferred properly in each individual branching statement. Loops should be tested with conditions that will produce 0, 1, and the maximum possible number of iterations, as well as a typical number of iterations in between the extremes.

The fundamental purpose of writing a test plan is to be able to reproducibly run your software through a wringer and see how it performs. The task of executing all these tests is simplified if you can build a library of test data files and sample output for comparison. Anything you can do to automate the testing process will save time and avoid human errors. This can be difficult with programs having a strong emphasis on user interface. One possibility is to use a tool to capture keystrokes and mouse clicks and build a library of macros that simulate specific sequences of human activities.

We can think of two different aspects to testing a piece of software. First, does the software properly perform its intended functions? And second, is the structure of the code free of syntax and logic errors? This is the dichotomy of "functional" versus "structural" testing. The customer is concerned with the first case; he doesn't care what the code looks like, so long as it does the job. He thinks of the program as a black box: he supplies the inputs, and by some magical process he gets the desired outputs back. The developer is concerned about the structure and flow of his code; to test it properly, he must study the code and devise tests in

accordance with the way the program is written. These two aspects of software testing are called "black box" and "white box" testing, respectively.

White box testing typically is performed by the developer. It is a unit- or module-level testing process, in which tests are devised to evaluate the program logic and internal control flow of each individual subroutine or function in the system. Black box testing is performed by the independent testers we discussed earlier, as well as by the customers. Black box testing focuses on the proper handling of supplied inputs to generate the expected outputs.

If you review the topics that I suggested your test script should include, you'll see a mix of structural and functional aspects described. The customer is the ultimate judge of whether the software meets his stated requirements. But it's the developer's responsibility to guarantee that the code is properly written so as to trap errors, handle both valid and invalid input data, and transfer control properly.

Let's assume you're working on a software project with a few dozen separate modules. Your unit-level testing has convinced you that they are all well-behaved and properly coded. Your next problem is to assemble all these modules together into the final system, in accordance with the program architecture you devised during system design.

There are basically two ways to approach the process of system integration and integration testing. You can pull the modules all together in one fell swoop, cross your fingers, and offer sacrifices to the compiler gods. This is called "big-bang" integration, and it's almost guaranteed to fail. Or, you can use an incremental approach of joining modules into small clusters and testing the clusters as you go. This technique is superior, and it will cost you much less money for aspirin.

The incremental integration approach can use either a top-down or a bottom-up method, although in practice a combined "sandwich" method often works best. With a top-down strategy, you begin with the highest-level control modules and add one lower-level module at a time. After each module is added, you perform appropriate tests to see that the fusion went properly. The most likely source of errors is in the data interface between each pair of modules. The bottom-up technique begins by clustering your lowest-level modules (where the real work gets done) together, testing the clusters for proper behavior, and working your way up until the whole functional processing cluster is joined to the high-level control modules.

The problem with any integration strategy is that you need to simulate the presence of certain modules before you've actually incorporated them. With top-down integration, you must simulate the presence of low-level modules by using "stub" modules. These stubs are replaced one at a time by their actual counterparts. The stubs must have a certain amount of functionality built in so that they can fake out the higher-level modules into thinking that they are the real thing. Stubs can show simple trace messages to indicate when control is properly passed into the lower-level module, and they can display any passed parameters to demonstrate whether the data interface is functioning correctly. Stubs may also have to return some data, either real or simulated, to the calling module to allow execution of the cluster you're now testing to continue.

Similarly, the clusters you're testing in a bottom-up approach aren't designed to function without the higher-level modules attached, so you have to use temporary "driver" modules to get the program to execute. The drivers can simply invoke the top-level module in the cluster you're now testing, or it can pass in some parameters both to test the data interface and to allow the cluster to do its thing and return some results to the driver.

Neither top-down nor bottom-up integration is perfect, so you may want to select some combination strategy for your individual projects. The method you choose may be based on where you expect to have the most problems. If control is complex and processing simple, top-down offers the advantage of testing the control modules first. If the data interface between your system and the outside environment is a big issue, bottom-up may be the best approach. In general, drivers are more complex than stubs, so you may have to do a little more work if you use a straight bottom-up

method.

Testing isn't complete just because you think you've integrated all your modules properly. It's very important to perform an overall system test to look for additional errors that didn't crop up during the unit testing or integration process. Also, acceptance testing by the users is essential to see if the final product does in fact match the stated requirements. For more discussion about these and other aspects of software testing, see the books listed in the references.

Q.3.(*c*) What do you understand by 'Earned Value Analysis'? Discuss the following progress indicators :

(*i*) Cost Variance (CV)

(*ii*) Schedule Performance Index (SPI)

(*iii*) Cost performance Index (CPI)

Ans. Earned Value Management (EVM) is a *project management* technique for measuring project progress in an objective manner. EVM has the unique ability to combine measurements of scope, schedule, and cost in a single integrated system. When properly applied, EVM provides an early warning of performance problems. Additionally, EVM promises to improve the definition of *project scope*, prevent *scope creep*, communicate objective progress to *stakeholders*, and keep the *project team* focused on achieving progress.

Essential features of any EVM implementation include

1. a *project plan* that identifies work to be accomplished,

2. a valuation of planned work, called Planned Value **(PV)** or *Budgeted Cost of Work Scheduled* **(BCWS)**, and

3. pre-defined "earning rules" (also called metrics) to quantify the accomplishment of work, called Earned Value **(EV)** or *Budgeted Cost of Work Performed* **(BCWP)**,.

EVM implementations for large or complex projects include many more features, such *as* indicators and forecasts of cost performance (over budget or under budget) and schedule performance (behind schedule or ahead of schedule). The most basic requirement of an EVM system, however. is that it quantifies progress using PV and EV.

Advanced Implementations (integrating cost, technical and schedule performance)

In addition to managing technical and schedule performance, large and complex projects require that cost performance is monitored and reviewed at regular intervals. To measure cost performance, planned value (or BCWS - *Budgeted Cost of Work Scheduled*) and earned value (or BCWP-Budgeted Cost of Work Performed) must be in units of currency (the same units that actual costs are measured.) In large implementations, the planned value curve is commonly called a Performance Measurement Baseline **(PMB)** and may be arranged in control accounts, summary-level planning packages, planning packages and work packages. In large projects, establishing control accounts is the primary method of delegating responsibility and authority to various parts of the performing organization. Control accounts are cells of a *responsibility assignment matrix*, which is intersection of the project WBS and the *organizational breakdown structure* **(OBS)**. Control accounts are assigned to Control Account Managers **(CAMs)**. Large projects require more elaborate processes for controlling baseline revisions, more thorough integration with subcontractor EVM systems, and more elaborate management of procured materials.

In the United States, the primary standard for full-featured EVM systems is the ANSI EIA-748A standard, published in May 1998 and reaffirmed in August 2002. The standard defines 32 criteria for full-featured EVM system compliance. As of the year 2007, a draft of EIA-748B, a revision to the original is available from ANSI. Other countries have established similar standards.

The following EVM formulas are for schedule management, and do not require accumulation of actual cost (AC). This is important because it is not uncommon in small and intermediate size projects for true costs to be unknown or unavailable.

Schedule Variance (SV)

EV-PV greater than 0 is good (ahead of schedule)

Schedule Performance Index (SPI)

EV/PV greater than 1 is good (ahead of schedule)

In addition to using BCWS and BCWP, prior to 1998 implementations often use the term Actual Cost of Work Performed (**ACWP**) instead of AC. Additional acronyms and formulas include:

Budget at Completion (BAC)

The total planned value (PV or BCWS) at the end of the project. If a project has a Management Reserve (MR), it is typically in addition to the BAC.

Cost Variance (CV)

EV-AC, greater than 0 is good (under budget)

Cost Performance Index (CPI)

EV/AC, greater than 1 is good (under budget)

< 1 means that the cost of completing the work is higher than planned (bad)

= 1 means that the cost of completing the work is right on plan (good)

> 1 means that the cost of completing the work is less than planned (good or sometimes bad).

Having a CPI that is very high (in some cases, very high is only 1.2) may mean that the plan was too conservative, and thus a very high number may in fact not be good, since the CPI is being measured against a poor baseline. Management or the customer may be upset with the planners since an overly conservative baseline ties up available funds for other purposes, and the baseline is also used for manpower planning.

4. Attempt any two parts of the following : **10×2=20**

Q.4.(*a*) How would you define 'Software Quality'? What do you mean by software quality assurance? Also discuss the various software quality assurance.

Ans. In the context of *software engineering*, **software quality** measures how well *software* is designed (*quality of design*), and how well the software conforms to that design (*quality of conformance*)[1], although there are several different definitions.

Whereas *quality of conformance* is concerned with implementation (see *Software Quality Assurance*), *quality of design* measures how valid the design and *requirements* are in creating a worthwhile *product*[2].

One of the problems with Software Quality is that "everyone feels they understand it."[3] In addition to the definition above by Dr. Scott Pressman, other software engineering experts have given several definitions.

A definition in Steve McConnell's *Code Complete* similarly divides software into two pieces: **internal and external quality characteristics.** External quality characteristics are those parts of a product that face its users, where internal quality characteristics are those that do not.[4]

Another definition by *Dr. Tom DeMarco* says "a product's quality is a function of how much it changes the world for the better."[5] This can be interpreted as meaning that user satisfaction is more important than anything in determining software quality.[1]

Another definition, coined by *Gerald Weinberg* in *Quality Software Management: Systems Thinking*, is "Quality is value to some person." This definition stresses that quality is inherently subjective - different people will experience the quality of the same software very differently. One strength of this definition is the questions it invites software teams to consider, such as "Who are the people we want to value our software?" and "What will be valuable to *them*?".

Software quality assurance (SQA) consists of a means of monitoring the *software engineering* processes and methods used to ensure quality. It does this by means of audits of the *quality management system* under which the software system is created. These audits are backed by one or more standards, usually *ISO 9000* or *CMMI*.

Software quality assurance is distinct from *software quality control* which includes reviewing *requirements* documents, and *software testing*. SQA encompasses the entire *software development* process, which includes processes such as *software design, coding, source code control, code reviews, change management, configuration management, and release management*. Whereas software quality control is a control of products, software

quality assurance is a control of processes.

Software quality assurance is related to the practice of quality assurance in product *manufacturing*. There are, however, some notable differences between software and a manufactured product. These differences stem from the fact that the manufactured product is physical and can be seen whereas the software product is not visible. Therefore its function, benefit and costs are not as easily measured. What's more, when a manufactured product rolls off the assembly line, it is essentially a complete, finished product, whereas software is never finished.[*citation needed*] Software lives, grows, evolves, and metamorphoses, unlike its tangible counterparts. Therefore, the processes and methods to manage, monitor, and measure its ongoing quality are as fluid and sometimes elusive as are the defects that they are meant to keep in check.

SQA Methodology

Software testing is as much an art as a science. In large, complex applications, such as operating systems, it is practically impossible to iron out every single bug before releasing it both from a difficulty point of view and due to time constraints. Different software applications require different approaches when it comes to testing, but some of the most common tasks in software QA include:

- PPQA audits: Process and Product Quality Assurance is the activity of ensuring that the process and work product conform to the agreed upon process.

The following quality control activities are often confused as quality assurance activities:

- Peer Reviews: Peer reviews of a project's work products are the most efficient defect removal (quality control) activity.
- Validation testing: Validation testing is the act of entering data that the tester knows to be erroneous into an application. For instance, typing "Hello" into an edit box that is expecting to receive a numeric entry.
- Data comparison: Comparing the output of an application with specific parameters to a previously created set of data with the same parameters that is known to be accurate.
- Stress testing: A stress test is when the software is used as heavily as possible for a period of time to see whether it copes with high levels of load. Often used for server software that will have multiple users connected to it simultaneously. Also known as Destruction testing. *Fault injection* is especially useful for any software system with exposed interfaces, e.g., protocol implementations.
- Conformance testing: Confirms that the software implementation complies with established standards.
- Load testing: Establishes the maximum amount of traffic that a target can accept, usually measured in packets-per-second (pps) or calls-per-second or some other rate-oriented metric. Such devices often have very fast hardware-optimized data plane processing, with software-based control and management plane functions. The SQA process is designed to ensure that the software portions of the system properly support the hardware functions.
- Usability testing: Sometimes getting users who are unfamiliar with the software to try it for a while and offer feedback to the developers about what they found difficult to do is the best way of making improvements to a user interface.
- Robustness testing: Software systems are presented with invalid or unexpected inputs to determine whether they have robust error-handling or input validation. Such systems pass the test(s) if they can tolerate a wide variety of invalid or unexpected inputs across the entire protocol interface specification. Commercially available robustness testing products exist, including *Mu Dynamics' Service Analyzer*.

Q.4.(*b*) Discuss the SEI Capability Maturity Model in detail.

Ans. In 1991 the Software Engineering Institute (SEI) at Carnegie Mellon University introduced the Capability Maturity Model for Software (CMM). This event marked a major milestone in the evolution

of software process management because, for the first time, the software community had a comprehensive description of how software organizations "mature" or improve, in their ability to develop software.

The CMM defines five levels of process capability, each of which represents an evolutionary plateau towards a disciplined, measured, and continuously improving software process.

The Initial Level

At the Initial level (Level 1) few, if any, organized processes exist. Each developer utilizes whatever methods or techniques strike his or her fancy. The situation is sometimes described as chaotic and ad hoc. Software quality is more a matter of chance, and is highly dependent on the capabilities of specific individuals within the organization.

The Repeatable Level

To reach Level 2, a software development organization must put into place basic project management practices. This includes the capability to estimate the size of the software to be produced, estimate resources to execute the project, and track progress against these estimates. Also included is the implementation of software configuration management and quality assurance practices, the capability to effectively manage the requirements definition process, and the capability to manage subcontractors (if applicable). This level is referred to as the Repeatable level; the organization has mastered tasks previously learned. The organization is still highly dependent on individuals for the success of a project. In times of stress, the organization tends to revert back to behaving as a Level 1 organization.

The Defined Level

Level 3 is characterized as the Defined level. At this level, the organization has defined and established the software development and maintenance practices specific to the types of applications they produce. They have put into place a set of standards and procedures to codify these practices, and the organization follows them consistently. Training in these practices is provided. Peer reviews are performed as in-process evaluations of product quality. Integrated project management exists. The organization is no longer highly dependent on key individuals; the process belongs to the organization, not to individuals. At times of stress, the Level 3 practices are not abandoned.

The Managed Level

At Level 3 and below, the primary focus is on product quality. At Level 4 and above, the primary focus shifts to process quality (although some amount of attention is paid to process quality below Level 4). To reach Level 4, the Managed level, the organization focuses on establishing a set of process measures and uses them to initiate corrective actions. Once these measures have been established, the organization is ready to begin to use them to implement continuous process improvement.

The Optimizing Level

At Level 5, the Optimizing level, these measures are not only being used to improve existing processes, but also to evaluate candidate new processes. They are also being used as the basis for determining the efficacy of introducing new technologies into the organization.

Using the CMM

How can the CMM help your organization? There are three key roles the CMM plays. First, the CMM helps build an understanding of software process by describing the practices that contribute to a level of process maturity.

The second role of the CMM is to provide a consistent basis for conducting appraisals of software processes. The CMM defines a scale for measuring process maturity, thus allowing an organization to accurately compare its process capability to that of another organization. ISO is using the CMM in its efforts to develop international standards for software process assessments.

The CMM's third key role is to serve as a blueprint for software process improvement. The CMM can help an organization focus on the areas it must address in order to advance to the next level of maturity.

Today, leading software organizations are adopting the CMM as their core strategy for improving quality and productivity

Q.4.(c) Distinguish between software verification and software verification. When, during the software life cycle are verification and validation performed? Can one be used in place of the other?

Ans. In *software project management*, software testing, and *software engineering*, **Verification and Validation (V&V)** is the process of checking that a software system meets specifications and that it fulfils its intended purpose. It is normally part of the *software testing* process of a project. In pharmaceutical industry, verification involves testing the suitability of well established procedures or (compendial) methods, whereas validation varies from *Cross validation. Empirical validation*, periodic partial validation, internal/external validation, competence validation by nature, and *Cleaning validation. Process validation. Equipment validation*, or *Documentation validation* by tasks.

Definitions

Also known as *software quality control*

Validation checks that the product design satisfies or fits the intended usage (high-level checking) — *i.e.*, you built the right product. This is done through *dynamic testing* and other forms of review.

According to the *Capability Maturity Model* (CMMI-SW v1.1), "Validation - The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610] Verification- The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610]."

In other words, validation ensures that the product actually meets the user's needs, and that the specifications were correct in the first place, while verification is ensuring that the product has been built according to the requirements and design specifications. Validation ensures that 'you built the right thing'. Verification ensures that 'you built it right'. Validation confirms that the product, as provided, will fulfill its intended use.

Within the *modeling and simulation* community, the definitions of validation, verification and accreditation are similar:

- *Validation* The process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s).[1]
- *Accreditation* is the formal certification that a model or simulation is acceptable to be used for a specific purpose. [2]
- *Verification*: The process of determining that a *computer model*, simulation, or federation of models and simulations implementations and their associated data accurately represents the developer's conceptual description and specifications. [3]

Related concepts

Both verification and validation are related to the concepts of quality and *of software quality assurance*. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required.

Classification of methods

In *mission-critical* systems where flawless performance is absolutely necessary, *formal methods* can be used to ensure the correct operation of a system. However, often for non-mission-critical systems, formal methods prove to be very costly and an alternative method of V&V must be sought out. In this case, *syntactic methods* are often used.

Test cases

A test case is a tool used in the V&V process.

The QA team prepares test cases for verification-to determine if the process that was followed to develop the final product is right.

The QC team uses a test case for validation-if the product is built according to the requirements of the user. Other methods, such as reviews, when used early in the Software Development Life Cycle provide for validation.

Verification can be called a part of validation process.

Independent Verification and Validation

Verification and validation often is carried out by a separate group from the development team; in this case, the process is called "***Independent Verification and Validation***", or ***IV&V***.

5. Attempt any two parts of the following : **10×2=20**

Q.5.(*a*) What is meant by Software Configuration Management? What are the principal activities involved in SCM? Why is SCM crucial to the success of a large software project?

Ans. In *software engineering*, **software configuration management** (SCM) is the task of tracking and controlling changes in the software. Configuration management practices include *revision control* and the establishment of baselines.

SCM concerns itself with answering the question "Somebody did something, how can one reproduce it?" Often the problem involves not reproducing "it" identically, but with controlled, incremental changes. Answering the question thus becomes a matter of comparing different results and of analysing their differences. Traditional configuration management typically focused on controlled creation of relatively simple products. Now, implementers of SCM face the challenge of dealing with relatively minor increments under their own control, in the context of the complex system being developed.

Terminology

The *history* and terminology of SCM (which often varies) has given rise to controversy. *Roger Pressman*, in his book *Software Engineering: A Practitioner's Approach*, states that SCM is a "set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made."

Source configuration management is a related practise often used to indicate that a variety of artifacts may be managed and versioned, including software code, documents, design models, and even the directory structure itself.

Atria (later *Rational Software*, now a part of IBM), used "SCM" to mean "software configuration management". Gartner uses the term *software change and configuration management*.

Purposes

The goals of SCM are generally:[*citation needed*]

- Configuration identification-What code are we working with?
- Configuration control-Controlling the release of a product and its changes.
- Status accounting-Recording and reporting the status of components.
- Review-Ensuring completeness and consistency among components.
- *Build management*-Managing the process and tools used for builds.
- Process management-Ensuring adherence to the organization's development process.
- Environment management-Managing the software and hardware that host our system.
- Teamwork-Facilitate team interactions related to the process.
- Defect tracking-Making sure every defect has traceability back to the source

Q.5.(*b*) List the common types of risks that a typical software project might suffer from. If you are the project manager of a large software development project, what steps would you follow to manage risks in your software project.

Ans. ***What is Risk?***

Risk is defined as "The possibility of suffering harm or loss; danger." Even if we're not familiar with the formal definition, most of us have an innate sense of risk. We are aware of the potential dangers that permeate even simple daily activities, from getting injured when crossing the street to having a heart attack because our cholesterol level is too high. Although we prefer not to dwell on the myriad of hazards that surround us, these risks shape many of our behaviors. Experience (or a parent) has taught us to look both ways before stepping off the curb

and most of us at least think twice before ordering a steak. Indeed, we manage personal risks every day.

Risks in Software Project Management

Unlike the hazards of daily living, the dangers in the young and emerging field of software engineering must often be learned without the benefit of lifelong exposure. A more deliberate approach is required. Such an approach involves studying the experiences of successful project managers as well as keeping up with the leading writers and thinkers in the field. One such writer in the area of risk is Dr. Barry W. Boehm. In his article "Software Risk Management: Principles and Practices" he lists the following top 10 software risk items:

1. Personnel Shortfalls
2. Unrealistic schedules and budgets
3. Developing the wrong functions and properties
4. Developing the wrong user interface
5. Gold-plating
6. Continuing stream of requirements changes
7. Shortfalls in externally furnished components
8. Shortfalls in externally performed tasks
9. Real-time performance shortfalls
10. Straining computer-science capabilities

How to Manage

In the same article, Dr. Boehm describes risk management as being comprised of the following activities:

- *Risk Assessment* (figuring out what the risks are and what to focus on)
  - o - making a list of all of the potential dangers that will affect the project
  - o - assessing the probability of occurence and potential loss of each item listed
  - o - ranking the items (from most to least dangerous)
- *Risk Control* (doing something about them)
  - o - coming up with techniques and strategies to mitigate the highest ordered risks
  - o - implementing the strategies to resolve the high order risks factors
  - o - monitoring the effectiveness of the strategies and the changing levels of risk throughout the project

Q.5.(c) What do you understand by the term "CASE" Tools? What are the main advantages of using CASE tools? Also discuss and draw the architecture of a CASE environment.

Ans. CASE tools

CASE tools are a class of software that automates many of the activities involved in various *life cycle* phases. For example, when establishing the *functional requirements* of a proposed application, *prototyping* tools can be used to develop graphic models of application screens to assist end users to visualize how an application will look after development. Subsequently, system designers can use automated design tools to transform the prototyped functional requirements into detailed design documents. Programmers can then use automated code generators to convert the design documents into code. Automated tools can be used collectively, as mentioned, or individually. For example, prototyping tools could be used to define application requirements that get passed to design technicians who convert the requirements into detailed designs in a traditional manner using flowcharts and narrative documents, without the assistance of automated design software.[6]

Classification of CASE Tools

Existing CASE Environments can be classified along 4 different dimensions :

1. Life-Cycle Support
2. Integration Dimension
3. Construction Dimension
4. Knowledge Based CASE dimension [7]

Let us take the meaning of these dimensions along with their examples one by one :

[edit] Life-Cycle Based CASE Tools

This dimension classifies CASE Tools on the basis of the activities they support in the information systems life cycle. They can be classified as Upper or Lower CASE tools.

- Upper CASE Tools: support strategic, planning and construction of conceptual level product and ignore the design aspect. They support traditional diagrammatic languages such as *ER diagrams. Data flow diagram, Structure charts* etc.
- Lower CASE Tools : concentrate on the back end activities of the software life cycle and hence support activities like physical design, debugging, construction, testing, integration of software components, maintenance, reengineering and reverse engineering activities.

Integration Dimension

Three main CASE Integration dimension have been proposed : [8]

1. CASE Framework
2. ICASE Tools
3. Integrated Project Support Environment (IPSE)

Applications

All aspects of the software development life cycle can be supported by software tools, and so the use of tools from across the spectrum can, arguably, be described as CASE: from project management software through tools for business and functional analysis, system design, code storage, *compilers*, translation tools, test software, and so on.

However, it is the tools that are concerned with analysis and design, and with using design information to create parts (or all) of the software product, that are most frequently thought of as CASE tools. CASE applied, for instance, to a database software product, might normally involve:

- Modelling business / real world processes and data flow
- Development of data models in the form of entity-relationship diagrams
- Development of process and function descriptions
- Production of database creation SQL and stored procedures

Risks and associated controls

Common CASE risks and associated controls include:

- *Inadequate Standardization* : Linking CASE tools from different vendors (design tool from Company X, programming tool from Company Y) may be difficult if the products do not use standardized code structures and data classifications. File formats can be converted, but usually not economically. Controls include using tools from the same vendor, or using tools based on standard protocols and insisting on demonstrated compatibility. Additionally, if organizations obtain tools for only a portion of the development process, they should consider acquiring them from a vendor that has a full line of products to ensure future compatibility if they add more tools. [6]
- *Unrealistic Expectations* : organizations often implement CASE technologies to reduce development costs. Implementing CASE strategies usually involves high start-up costs. Generally, management must be willing to accept a long-term *payback period*. Controls include requiring senior managers to define their purpose and strategies for implementing CASE technologies. [6]
- *Quick Implementation* : Implementing CASE technologies can involve a significant change from traditional development environments. Typically, organizations should not use CASE tools the first time on critical projects or projects with short deadlines because of the lengthy training process. Additionally, organizations should consider using the tools on smaller, less complex projects and gradually implementing the tools to allow more training time. [6]
- *Weak Repository Controls* : Failure to adequately control access to CASE repositories may result in security breaches or damage to the work documents, system designs, or code modules stored in the repository. Controls include protecting the repositories with appropriate access, version, and backup controls. [6]