USN | | | | | | | | | | |

Internal Assessment Test II – December 2024

Solution Key

Sub:	Software Engineering and Project Management					Sub Code:	BCS501	Branch:	ISE		
Date:	14-12-2024	Duration:	90 min's	Max Marks:	50	Sem/Sec:	V / A, B & C			OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a. List and briefly explain the special characteristics of a software reflecting on the importance of its quality. Solutions:	5 5	CO5	L3

ISO 9126 identifies six major external software quality characteristics:

- *functionality*, which covers the functions that a software product provides to satisfy user needs;
- *reliability*, which relates to the capability of the software to maintain its level of performance;
- *usability*, which relates to the effort needed to use the software;
- *efficiency*, which relates to the physical resources used when the software is executed;
- *maintainability*, which relates to the effort needed to the make changes to the software;
- *portability*, which relates to the ability of the software to be transferred to a different environment.

'Functionality compliance' refers to the degree to which the software adheres to application-related standards or legal requirements. Typically these could be auditing requirements. Since the original 1999 draft, a sub-characteristic called 'compliance' has been added to all six ISO external characteristics. In each case, this refers to any specific standards that might apply to the particular quality attribute.

'Maturity' refers to the frequency of failure due to faults in a software product, the implication being that the more the software has been used, the more faults will have been uncovered and removed. It is also interesting to note that 'recoverability' has been clearly distinguished from 'security' which describes the control of access to a system.

Note how 'learnability' is distinguished from 'operability'. A software tool could be easy to learn but time-consuming to use because, say, it uses a large number of nested menus. This might be fine for a package used intermittently, but not where the system is used for many hours each day. In this case 'learnability' has been incorporated at the expense of 'operability'.

'Attractiveness' is a recent addition to the sub-characteristics of usability and is especially important where users are not compelled to use a particular software product, as in the case of games and other entertainment products.

b. Use the COCOMO II model to estimate the effort required to build software for a simple ATM that produces 12 screens, 10 reports, and will require approximately 80 software components. Assume average complexity and average developer/environment maturity.

Solution:

26.7.2 The COCOMO II Model

In his classic book on "software engineering economics," Barry Boehm [Boe81] introduced a hierarchy of software estimation models bearing the name COCOMO, for *COnstructive COst MOdel*. The original COCOMO model became one of the most widely used and discussed software cost estimation models in the industry. It has evolved into a more comprehensive estimation model, called COCOMO II [Boe00]. Like its predecessor, COCOMO II is actually a hierarchy of estimation models that address the following areas:

- *Application composition model.* Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
- *Early design stage model.* Used once requirements have been stabilized and basic software architecture has been established.
- *Post-architecture-stage model.* Used during the construction of the software.

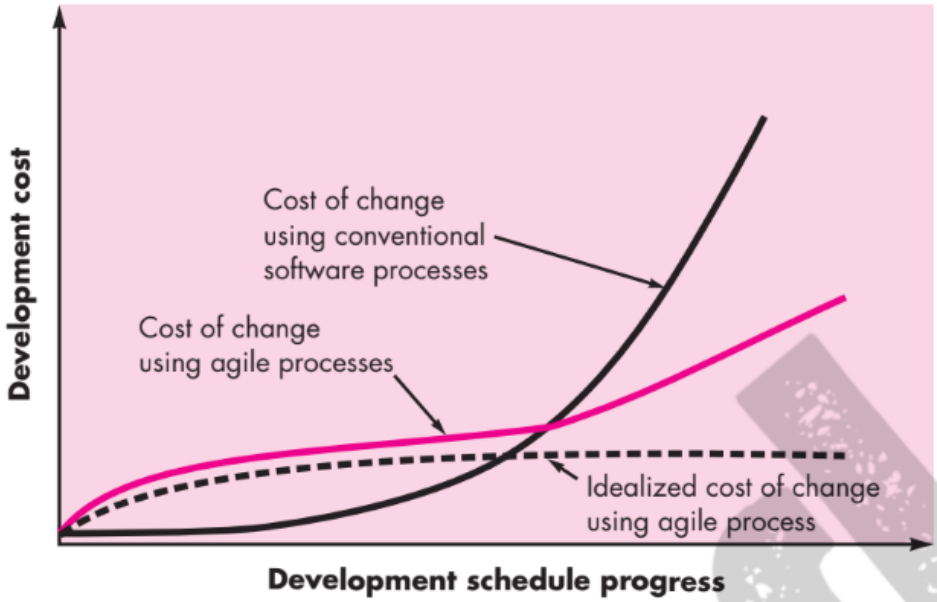
Like all estimation models for software, the COCOMO II models require sizing information. Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code.

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

The COCOMO II application composition model uses object points and is illustrated in the following paragraphs. It should be noted that other, more sophisticated estimation models (using FP and KLOC) are also available as part of COCOMO II.

Like function points, the *object point* is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports, and (3) components likely to be required to build the application. Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e., simple, medium, or difficult) using criteria suggested by Boehm [Boe96]. In essence, complexity is a function of the number and source of the client and server data tables that are required to generate the screen or report and the number of views or sections presented as part of the screen or report.

	When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted: NOP = (object points) × [(100 − %reuse)/100] where NOP is defined as new object points. To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived. Figure 26.7 presents the productivity rate $\text{PROD} = \frac{\text{NOP}}{\text{person-month}}$ for different levels of developer experience and development environment maturity. Once the productivity rate has been determined, an estimate of project effort is computed using $\text{Estimated effort} = \frac{\text{NOP}}{\text{PROD}}$ In more advanced COCOMO II models,[12] a variety of scale factors, cost drivers,			
2	a. What is agility? Discuss the relationship between agility and cost of change with a suitable diagram. Solution: *Agility* has become today's buzzword when describing a modern software process. Everyone is agile. An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about. Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product. Support for changes should be built-in everything we do in software, something we embrace because it is the heart and soul of software. An agile team recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project. The conventional wisdom in software development (supported by decades of experience) is that the cost of change increases nonlinearly as a project progresses (Figure 3.1, solid black curve). It is relatively easy to accommodate a change when a software team is gathering requirements (early in a project). A usage scenario might have to be modified, a list of functions may be extended, or a written specification can be edited. The costs of doing this work are minimal, and the time required will	5 5	CO3	L2



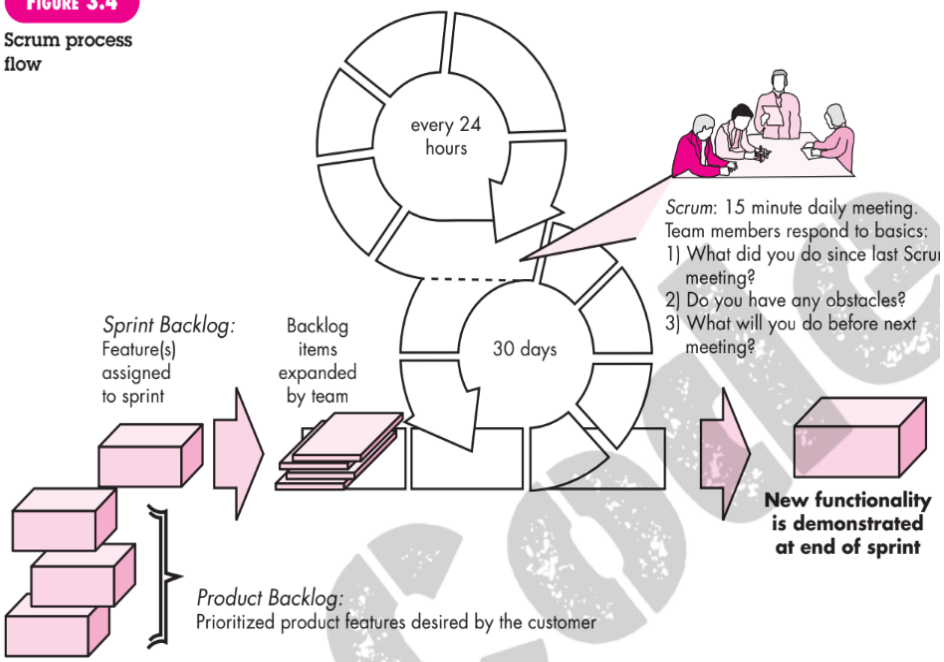
b. Explain the SCRUM process model with a suitable and neat diagram.

Solution:

Scrum (the name is derived from an activity that occurs during a rugby match[13]) is an agile software development method that was conceived by Jeff Sutherland and his development team in the early 1990s. In recent years, further development on the Scrum methods has been performed by Schwaber and Beedle [Sch01a].

Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities: requirements, analysis, design, evolution, and delivery. Within each

FIGURE 3.4
Scrum process flow



3	a. What are the different types of software quality model? List the advantages of adopting a quality model for developing a software.	5 5	CO5	L3

	The need to be able to quantitatively measure the quality of a software is often felt. For example, one may want to set quantitative quality requirements for a software, or to verify whether software meets the quality requirements set of it. Unfortunately, it is hard to directly measure the quality of a software. However, it can be expressed in terms of several attributes of a software that can be directly measured. The quality models give a characterization (often hierarchical) of software quality in terms of set of characteristics of the software. The bottom level of the hierarchy can be directly measured, thereby enabling a quantitative assessment of the quality of the software. There are several well-established quality models, including McCall's, Dromey's and Boehm's. Since there was no standardization among the large number of quality models that became available, the ISO 9126 model of quality was developed. We briefly discuss Garvin's, McCall's, Dromey's and Boehm's quality models in this section and discuss ISO 9126 in the next section. **Garvin's quality dimensions:** David Garvin, a professor of Harvard Business school in his book, *Total Quality Management*, defined the quality of any product in terms of eight general attributes of the product, some of these are measurable and some are not. Garvin reasoned that sometimes users have subjective judgment of the quality of a program (perceived quality) that must be taken into account to judge its quality. • **Performance:** How well it performs the jobs. • **Features:** How well it supports the required features. • **Reliability:** Probability of a product working satisfactorily within a specific period of time. • **Conformance:** Degree to which the product meets the requirements. • **Durability:** Measure of the product life • **Serviceability:** Speed and effectiveness maintenance. • **Aesthetics:** The look and feel of the product. • **Perceived quality:** User's opinion about the product quality. b. Explain McCall's Model? Solution: *Quality Management* https://abdullahsufail.github.io/bscit **McCall' model:** McCall defined the quality of a software in terms of three broad parameters: its operation characteristics, how easy it is to fix defects and how easy it is to port it to different platforms. These three high-level quality attributes are defined based on the following eleven attributes of the software: • **Correctness:** The extent to which a software product satisfies its specifications. • **Reliability:** The probability of the software product working satisfactorily over a given duration. • **Efficiency:** The amount of computing resources required to perform the required functions. • **Integrity:** The extent to which the data of the software product remains valid. • **Usability:** The effort required to operate the software product. • **Maintainability:** The ease with which it is possible to locate and fix bugs in the software product. • **Flexibility:** The effort required to adapt the software product to changing requirements. • **Testability:** The effort required to test a software product to ensure that it performs its intended function. • **Portability:** The effort required to transfer the software product from one hardware or software system environment to another. • **Reusability:** The extent to which a software can be reused in other applications. • **Interoperability:** The effort required to integrate the software with other software.			
4	a. What is management? Explain the different activities of management.	5 5	CO4	L2

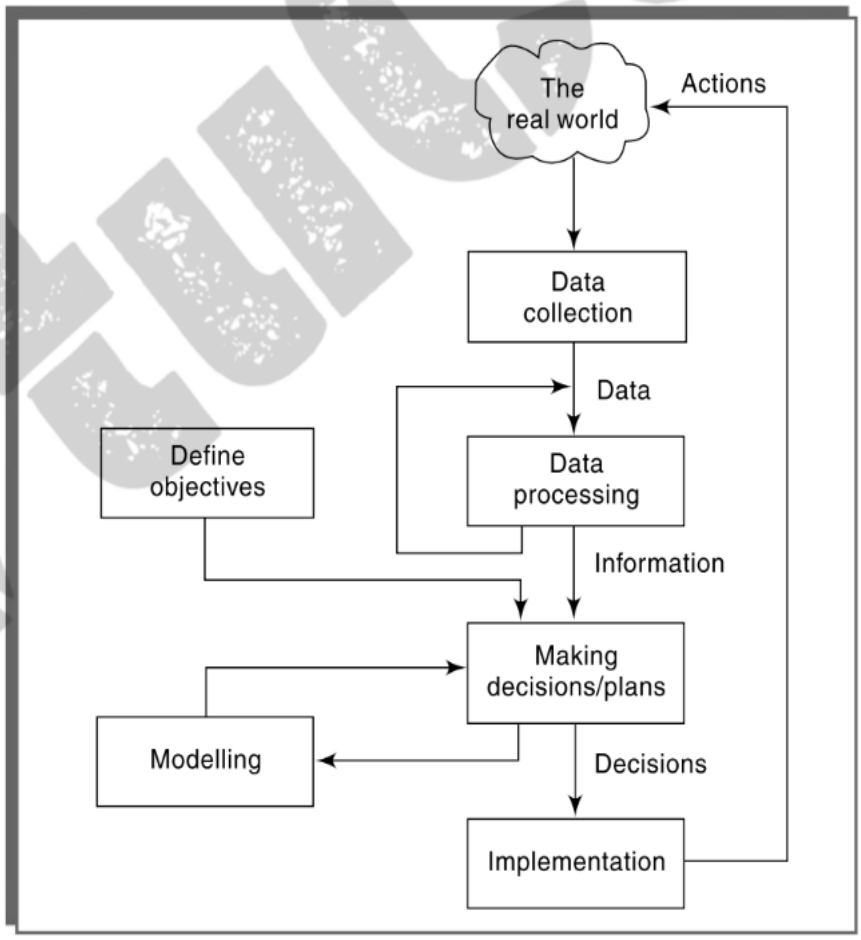
Solution:

1.13 What is Management?

We have explored some of the special characteristics of software. We now look at the 'management' aspect of software project management. It has been suggested that management involves the following activities:

- planning – deciding what is to be done;
- organizing – making arrangements;
- staffing – selecting the right people for the job etc.;
- directing – giving instructions;
- monitoring – checking on progress;
- controlling – taking action to remedy hold-ups;
- innovating – coming up with new solutions;
- representing – liaising with clients, users, developer, suppliers and other stakeholders.

b. Explain the project management control life cycle with a suitable diagram.



In our example, the project management might examine the 'estimated completion date' for completing data transfer for each branch. These can be checked against the overall target date for completion of this phase of the project. In effect they are comparing actual performance with one aspect of the overall project objectives. They might find that one or two branches will fail to complete the transfer of details in time. They would then need to consider what to do (this is represented in Figure 1.5 by the box *Making decisions/ plans*). One possibility would be to move staff temporarily from one branch to another. If this is done, there is always the danger that while the completion date for the one branch is pulled back to before the overall target date, the date for the branch from which staff are being moved is pushed forward beyond that date. The project manager would need to calculate carefully what the impact would be in moving staff from particular branches. This is *modelling* the consequences of a potential solution. Several different proposals could be modelled in this way before one was chosen for *implementation.*

5	a. Explain about activities covered by software project management with suitable diagrams?	5 5	CO4	L2

1.6 Activities Covered by Software Project Management

Chapter 4 on project analysis and technical planning looks at some alternative life cycles.

A software project is not only concerned with the actual writing of software. In fact, where a software application is bought 'off the shelf', there may be no software writing as such, but this is still fundamentally a software project because so many of the other activities associated with software will still be present.

Usually there are three successive processes that bring a new system into being – see Figure 1.2.

1. **The feasibility study** assesses whether a project is worth starting – that it has a valid *business case*. Information is gathered about the requirements of the proposed application. Requirements elicitation can, at least initially, be complex and difficult. The stakeholders may know the aims they wish to

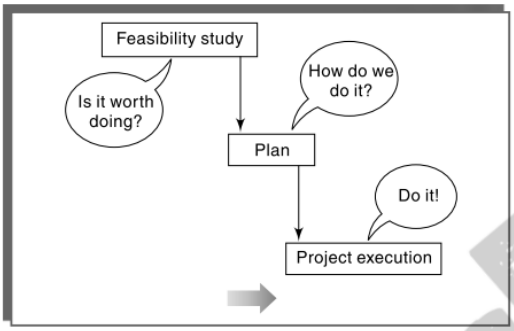


FIGURE 1.2 The feasibility study/plan/execution cycle

b. Define project and write the characteristics of projects.

Solutions:

1.3 What is a Project?

Dictionary definitions of 'project' include: 'A specific plan or design' 'A planned undertaking' 'A large undertaking: e.g. a public works scheme', *Longman Concise English Dictionary*, 1982.

The dictionary definitions put a clear emphasis on the project being a *planne* activity.

The emphasis on being planned assumes we can determine how to carry out a tas before we start. Yet with exploratory projects this might be difficult. Planning is i essence thinking carefully about something before you do it – even with uncertai projects this is worth doing as long as the resulting plans are seen as provisional. Othe activities, such as routine maintenance, will have been performed so many times tha everyone knows exactly what to do. In these cases, planning hardly seems necessary although procedures might be documented to ensure consistency and to help newcomers.

	The following characteristics distinguish projects: • non-routine tasks are involved; • planning is required; • specific objectives are to be met or a specified product is to be created; • the project has a predetermined time span; • work is carried out for someone other than yourself; • work involves several specialisms; • people are formed into a temporary work group to carry out the task; • work is carried out in several phases; • the resources that are available for use on the project are constrained; • the project is large or complex. The more any of these factors apply to a task, the more difficult that task will be. Project size is particularly important. The project that employs 20 developers is likely to be disproportionately more difficult than one with only 10 staff because of the need for additional coordination. The examples and exercises used in this book usually relate to smaller projects in order to make the techniques easier to grasp. However, the techniques and issues discussed are of equal relevance to larger projects.			
6	a. List and explain the principles that guide the process elements of software development **4.2.1 Principles That Guide Process** In Part 1 of this book I discussed the importance of the software process and described the many different process models that have been proposed for software engineering work. Regardless of whether a model is linear or iterative, prescriptive or agile, it can be characterized using the generic process framework that is applicable for all process models. The following set of core principles can be applied to the framework, and by extension, to every software process. **Principle 1. *Be agile.*** Whether the process model you choose is prescriptive or agile, the basic tenets of agile development should govern your approach. Every aspect of the work you do should emphasize economy of action—keep your technical approach as simple as possible, keep the work products you produce as concise as possible, and make decisions locally whenever possible. **Principle 2. *Focus on quality at every step.*** The exit condition for every process activity, action, and task should focus on the quality of the work product that has been produced. **Principle 3. *Be ready to adapt.*** Process is not a religious experience, and dogma has no place in it. When necessary, adapt your approach to constraints imposed by the problem, the people, and the project itself. **Principle 4. *Build an effective team.*** Software engineering process and practice are important, but the bottom line is people. Build a self-organizing team that has mutual trust and respect. b.Explain feature driven development with suitable diagram?	5+5	CO3	L3

Solution:

3.5.5 Feature Driven Development (FDD)

Feature Driven Development (FDD) was originally conceived by Peter Coad and his colleagues [Coa99] as a practical process model for object-oriented software engineering. Stephen Palmer and John Felsing [Pal02] have extended and improved Coad's work, describing an adaptive, agile process that can be applied to moderately sized and larger software projects.

Like other agile approaches, FDD adopts a philosophy that (1) emphasizes collaboration among people on an FDD team; (2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments, and (3) communication of technical detail using verbal, graphical, and text-based means. FDD emphasizes software quality assurance activities by encouraging an incremental development strategy, the use of design and code inspections, the application of software quality assurance audits (Chapter 16), the collection of metrics, and the use of patterns (for analysis, design, and construction).

In the context of FDD, a *feature* "is a client-valued function that can be implemented in two weeks or less" [Coa99]. The emphasis on the definition of features provides the following benefits:

In the context of FDD, a *feature* "is a client-valued function that can be implemented in two weeks or less" [Coa99]. The emphasis on the definition of features provides the following benefits:

- Because features are small blocks of deliverable functionality, users can describe them more easily; understand how they relate to one another more readily; and better review them for ambiguity, error, or omissions.
- Features can be organized into a hierarchical business-related grouping.
- Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.
- Because features are small, their design and code representations are easier to inspect effectively.
- Project planning, scheduling, and tracking are driven by the feature hierarchy, rather than an arbitrarily adopted software engineering task set.