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

Internal Assessment Test 2 – DEC 2024

Sub:	**Software Engineering & Project Management**					Sub Code:	**BCS501**	Branch:	**CSE**
Date:	16/12/24	Duration:	90 mins	Max Marks:	50		Sem/Sec:5 A/B/C		90 mins

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	A) What is scenario-based modeling, and how is it used to capture and represent different aspects of a software system during development? B) What is class-based modeling, and how does it help in organizing and structuring the components of a software system?	[5] [5]	CO2	L3
2	What are CRC (Class-Responsibility-Collaboration) models, and how do they help in defining the roles and interactions of classes? Additionally, what is a Composite Aggregate Class, and how is it used in object-oriented design?	[10]	CO2	L3
3	What are some ways to categorize software projects, and how do these categories help in managing and executing them effectively?	[10]	CO4	L4

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

Internal Assessment Test 2 – DEC 2024

Sub:	**Software Engineering & Project Management**					Sub Code:	**BCS501**	Branch:	**CSE**
Date:	16/12/24	Duration:	90 mins	Max Marks:	50		Sem/Sec:5 A/B/C		90 mins

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	A) What is scenario-based modeling, and how is it used to capture and represent different aspects of a software system during development? B) What is class-based modeling, and how does it help in organizing and structuring the components of a software system?	[5] [5]	CO2	L3
2	What are CRC (Class-Responsibility-Collaboration) models, and how do they help in defining the roles and interactions of classes? Additionally, what is a Composite Aggregate Class, and how is it used in object-oriented design?	[10]	CO2	L3
3	What are some ways to categorize software projects, and how do these categories help in managing and executing them effectively?	[10]	CO4	L4

4	A. What are the key differences between traditional and modern project management practices, and how do they impact the management and execution of software projects? B. What are the stages of the project management life cycle, and how do they contribute to the successful completion of a project?	[5] [5]	C O4	L2
5	How does product quality management differ from process quality management, and what are their respective impacts on software development?	[10]	C O5	L5
6	What is software quality, and why is it important in software development? Provide a detailed explanation of Dromey's Quality Model, including its key components, how it evaluates software quality, and its significance in improving software systems.	[10]	CO5	L5

CI **CCI** **HOD**

4	A. What are the key differences between traditional and modern project management practices, and how do they impact the management and execution of software projects? B. What are the stages of the project management life cycle, and how do they contribute to the successful completion of a project?	[10]	C O4	L2
5	How does product quality management differ from process quality management, and what are their respective impacts on software development?	[10]	C O5	L5
6	What is software quality, and why is it important in software development? Provide a detailed explanation of Dromey's Quality Model, including its key components, how it evaluates software quality, and its significance in improving software systems.	[10]	CO5	L5

CI **CCI** **HOD**

1A.

Scenario-Based Modeling

"[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)." Ivar Jacobson

(1) What should we write about?

(2) How much should we write about it?

(3) How detailed should we make our description?

(4) How should we organize the description?

What to Write About?

- Inception and elicitation—provide you with the information you'll need to begin writing use cases.
- Requirements gathering meetings, QFD, and other requirements engineering mechanisms are used to
  - identify stakeholders
  - define the scope of the problem
  - specify overall operational goals
  - establish priorities
  - outline all known functional requirements, and
  - describe the things (objects) that will be manipulated by the system.
- To begin developing a set of use cases, list the functions or activities performed by a specific actor.

How Much to Write About?

- As further conversations with the stakeholders progress, the requirements gathering team develops use cases for each of the functions noted.
- In general, use cases are written first in an informal narrative fashion.
- If more formality is required, the same use case is rewritten using a structured format similar to the one proposed.

1B.

Class-Based Modeling

- Class-based modeling represents:
  - objects that the system will manipulate
  - operations (also called methods or services) that will be applied to the objects to effect the manipulation
  - relationships (some hierarchical) between the objects
  - collaborations that occur between the classes that are defined.
- The elements of a class-based model include classes and objects, attributes, operations, CRC models, collaboration diagrams and packages.

Identifying Analysis Classes

- Analysis classes are abstract representations of key concepts or entities within the system's problem domain.
- These classes help bridge the gap between raw requirements and a more structured design.
- Analysis classes typically capture essential attributes and behaviors that are significant for understanding the system and its requirements.
- **Characteristics of Analysis Classes**
  - **Abstraction**: Analysis classes represent abstract concepts rather than concrete implementations.
  - **Attributes**: Key properties or data elements associated with the class.
  - **Operations:** Important functions that the class can perform.
  - **Associations:** Relationships between analysis classes, indicating how they interact or depend on each other.

Manifestations of Analysis Classes

- *Analysis classes* manifest themselves in one of the following ways:
  - *External entities* (e.g., other systems, devices, people) that produce or consume information
  - *Things* (e.g, reports, displays, letters, signals) that are part of the information domain for the problem
  - *Occurrences or events* (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation
  - *Roles* (e.g., manager, engineer, salesperson) played by people who interact with the system
  - *Organizational units* (e.g., division, group, team) that are relevant to an application
  - *Places* (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function
  - *Structures* (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects

Potential Classes

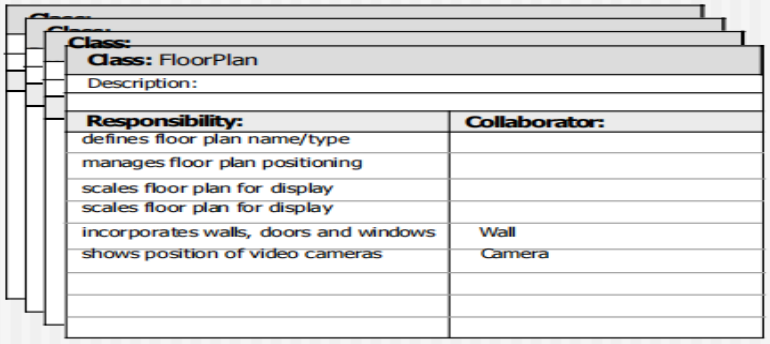
Potential Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity

2.

CRC Models

- *Class-responsibility-collaborator (CRC) modeling* [Wir90] provides a simple means for identifying and organizing the classes that are relevant to system or product requirements. Ambler [Amb95] describes CRC modeling in the following way:
  - A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card you write the name of the class. In the body of the card you list the class responsibilities on the left and the collaborators on the right.

CRC Modeling



Class Types

- *Entity classes*, also called *model* or *business* classes, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).
- *Boundary classes* are used to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.
- *Controller classes* manage a "unit of work" [UML03] from start to finish. That is, controller classes can be designed to manage
  - the creation or update of entity objects;
  - the instantiation of boundary objects as they obtain information from entity objects;
  - complex communication between sets of objects;
  - validation of data communicated between objects or between the user and the application.

Responsibilities

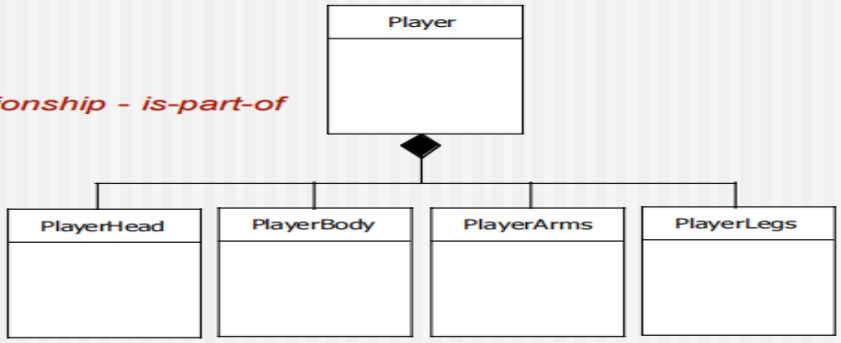
- System intelligence should be distributed across classes to best address the needs of the problem
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes, when appropriate.

Collaborations

- Classes fulfill their responsibilities in one of two ways:
  - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
    - a class can collaborate with other classes.
- Collaborations identify relationships between classes
- Collaborations are identified by determining whether a class can fulfill each responsibility itself
- three different generic relationships between classes
  - *is-part-of* relationship
  - the *has-knowledge-of* relationship
  - the *depends-upon* relationship

Composite Aggregate Class

- *Relationship - is-part-of*



Associations and Dependencies

- Two analysis classes are often related to one another in some fashion
  - In UML these relationships are called *associations*
  - Associations can be refined by indicating *multiplicity* (the term *cardinality* is used in data modeling)
- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class in some way and a *dependency relationship* is established

3.

1.8 Some Ways of Categorizing Software Projects

Projects may differ because of the different technical products to be created. Thus we need to identify the characteristics of a project which could affect the way in which it should be planned and managed. Other factors are discussed below.

Changes to the characteristics of software projects

Over the last few decades, the characteristics of software projects have undergone drastic changes. For example, in the initial years of software development, almost every software was being entirely written from scratch. A possible reason for this could be that in those early days of software development, not many programs were available from which code could be reused in a new software development project. As a result, the entire code for every project had to be written from scratch. Further inhibiting code reuse was the fact that the programming paradigms that existed in those days hardly provided any support for code reuse. In contrast, at present, almost every programming language supports several ways of reusing existing code, including elegant ways of customizing and extending existing code, efficiently and dynamically linking library routines and support for frameworks.

A large number of projects that are being undertaken now are essentially customization of some existing software or development for a new release of an existing software. In these projects, much of the code of the developed software comes from reused code and only a small part of the code is actually written by the development team. Consequently, project durations have now shrunk to only a few months compared to multi-year projects that were being undertaken till about a few decades back. Further, in the past, customer participation in software projects was largely restricted to only the initial interactions for requirements gathering, and specification and later, on completion of the project, taking delivery of the developed software. In contrast, at present, customer participation in almost every aspect of a project is being actively encouraged and often a few customer representatives are being included in the development team.

Compulsory versus voluntary users

In workplaces, there are systems that staff have to use if they want to do something, such as recording a sale. However, use of a system is increasingly voluntary, as in the case of computer games. Here it is difficult to elicit precise requirements from potential users as we could with a business system. What the game will do will thus depend much on the informed ingenuity of the developers, along with techniques such as market surveys, focus groups and prototype evaluation.

Information systems versus embedded systems

A traditional distinction has been between *information systems* which enable staff to carry out office processes and *embedded systems* which control machines. A stock control system would be an information system. An embedded, or process control, system might control the air conditioning equipment in a building. Some systems may have elements of both where, for example, the stock control system also controls an automated warehouse.

Embedded systems are also called real-time or industrial systems.

Exercise 1.5

Would an operating system on a computer be an information system or an embedded system?

Software products *versus* services

All types of software projects can broadly be classified into software product development projects and software services projects. These two broad classes of software projects can be further classified into subclasses, as shown in Figure 1.4. A software product development project concerns developing the software by keeping the requirements of the general customers in mind and the developed software is usually sold off-the-shelf to a large number of customers. A software product development project can further be classified depending on whether it concerns developing a generic product or a domain-specific product. A generic software product is sold to a broad spectrum of customers and is said to have a horizontal market. Examples of generic software products are Microsoft's Windows operating system and Oracle Corporation's Oracle 8i database management software. On the other hand, domain-specific software products are sold to specific categories of customers and are said to have a vertical market. Domain-specific software products target specific segments of customers (called verticals), such as, banking, telecommunication, finance and accounts and medical. Examples of domain-specific software products are BANCS from TCS and FINACLE from Infosys in the banking domain and AspenPlus from Aspen corporation in the chemical process simulation.

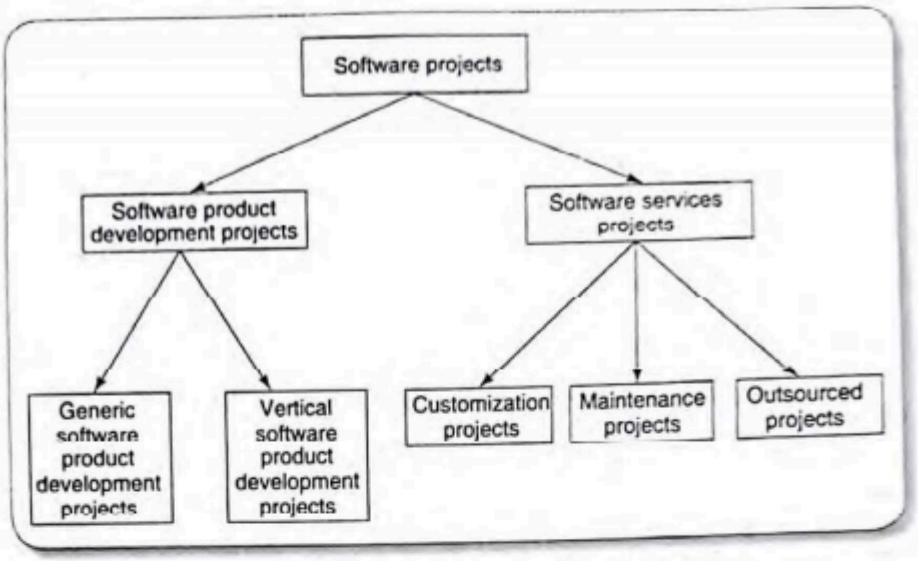


FIGURE 1.4 A classification of software projects

Software services on the other hand, cover a large gamut of software projects such as customization, outsourcing, maintenance, testing and consultancy. At present, there is a rapid growth in the number of software services projects that are being undertaken world-wide and software services are poised to become the dominant type of software projects. One of the reasons behind this situation is the steep growth in the available code base. Over the past few decades, a large number of programs have already been developed. Available programs can therefore be modified to quickly fulfil the specific requirements of any customer. At present, there is hardly any software project in which the program code is written from scratch, and software is being mostly developed by customizing some existing software. For example, to develop a software to automate the payroll generation activities of an educational institute, the vendor may customize an existing software that might have been developed earlier for a different client's educational institute. Due to heavy reuse of code, it has now become possible to develop even large software systems in rather short period of time. Therefore, typical project durations are at present only a couple of months and multi-year projects have become very rare.

Outsourced projects

While developing a large project, sometimes, it makes good commercial sense for a company to outsource some parts of its work to other companies. There can be several reasons behind such a decision. For example, a company may consider outsourcing as a good option, if it feels that it does not have sufficient expertise to develop some specific parts of the product or if it determines that some parts can be developed cost-effectively by another company. Since an outsourced project is a small part of some project, it is usually small in size and needs to be completed within a few months. Considering these differences between an outsourced project and a conventional project, managing an outsourced project entails special challenges.

Indian software companies excel in executing outsourced software projects and have earned a fine reputation in this field all over the world. Of late, the Indian companies have slowly begun to focus on product development as well.

4 A.

1.17 Traditional versus Modern Project Management Practices

Over the last two decades, the basic approach taken by the software industry to develop software has undergone a radical change. Hardly any software is being developed from scratch any more. Software development projects are increasingly being based on either tailoring some existing product or reusing certain pre-built libraries. In either case, two important goals of recent life cycle models are maximization of code reuse and compression of project durations. Other goals include facilitating and accommodating client feedbacks and customer participation in project development work, and incremental delivery of the product with evolving functionalities. Change requests from customers are encouraged, rather than circumvented. Clients on the other hand, are demanding further reductions in product delivery times and costs. These recent developments have changed project management practices in many significant ways. In the following section, we will discuss some important differences between modern project management practices and traditional practices.

- *Planning Incremental Delivery* Few decades ago, projects were much simpler and therefore more predictable than the present day projects. In those days, projects were planned with sufficient detail, much before the actual project execution started. After the project initiation, monitoring and control activities were carried out to ensure that the project execution proceeded as per plan. Now, projects are required to be completed over a much shorter duration, and rapid application development and deployment are considered key strategies. The traditional long-term planning has given way to adaptive short-term planning. Instead of making a long-term project completion plan, the project manager now plans all incremental deliveries with evolving functionalities. This type of project management is often called extreme project management. Extreme project management is a highly flexible approach to project management that concentrates on the human side of project management (e.g., managing project stakeholders), rather than formal and complex planning and monitoring techniques.
- *Quality Management* Of late, customer awareness about product quality has increased significantly. Tasks associated with quality management have become an important responsibility of the project manager. The key responsibilities of a project manager now include assessment of project progress and tracking the quality of all intermediate artifacts. We will discuss quality management issues in Chapter 13.
- *Change Management* Earlier, when the requirements were signed off by the customer, any changes to the requirements were rarely entertained. Customer suggestions are now actively being solicited and incorporated throughout the development process. To facilitate customer feedback, incremental delivery models are popularly being used. Product development is being carried out through a series of product versions implementing increasingly greater functionalities. Also customer feedback is solicited on each version for incorporation. This has made it necessary for an organization to keep track of the various versions and revisions through which the product develops. Another reason for the increased importance of keeping track of the versions and revisions is the following. Application development through customization has become a popular business model. Therefore, existence of a large number of versions of a product and the need to support these by a development organization has become common.

https://abdullahsurati.github.io/bsdit

In this context, the project manager plays a key role in product base lining and version control. This has made change management a crucial responsibility of the project manager. Change management is also known as configuration management. We will discuss change management in Chapter 9.

- *Requirements Management* In modern software development practices, there is a conscious effort to develop software such that it would, to a large extent, meet the actual requirements of the customer. A basic premise of these modern development methodologies is that at the start of a project the customers are often unable to fully visualize their exact needs and are only able to determine their actual requirements after they start using the software. From this view point, modern software development practices advocate delivery of software in increments as and when the increments are completed by the development team, and actively soliciting change requests from the customer as they use the increments of the software delivered to them. In fact, a few customer representatives are included in the development team to foster close every day interactions with the customers. Contrast this with the practice followed in older development methodologies, where the requirements had to be identified upfront and these were then 'signed off' by the customer and 'frozen' before the development could start. Change requests from the customer after the start of the project were discouraged. Consequently, at present in most projects, the requirements change frequently during the development cycle. It has, therefore, become necessary to properly manage the requirements, so that as and when there is any change in requirements, the latest and up-to-date requirements become available to all. Requirements management has therefore become a systematic process of controlling changes, documenting, analysing, tracing, prioritizing requirements and then communicating the changes to the relevant stakeholders. By the term controlling changes, we mean that every change request is well managed, and problems such as accidental overwriting of a newer document with an older document are avoided.

- *Release Management* Release management concerns planning, prioritizing and controlling the different releases of a software. For almost every software, multiple releases are made during its life cycle. Starting with an initial release, releases are made each time the code changes. There are several reasons as to why the code needs to change. These reasons include functionality enhancements, bug fixes and improved execution speed. Further, modern development processes such as the agile development processes advocate frequent and regular releases of the software to be made to the customer during the software development. Starting with the release of the basic or core functionalities of the software, more complete functionalities are made available to the customers every couple of weeks. In this context, effective release management has become important.
- *Risk Management* In modern software project management practices, effective risk management is considered very important to the success of a project. A risk is any negative situation that may arise as the project progresses and may threaten the success of the project. Every project is susceptible to a host of risks that could usually be attributed to factors such as technology, personnel and customer. Unless proper risk management is practised, the progress of the project may get adversely affected. Risk management involves identification of risks, assessment of the impacts of various risks, prioritization of the risks and preparation of risk-containment plans. We discuss various aspects of project risk management in Chapter 7.
- *Scope Management* Once a project gets underway, many requirement change requests usually arise. Some of these can be attributed to the customers and the others to the development team. As we have already mentioned, modern development practices encourage the customer to come up with change requests. While all essential changes must be carried out, the superfluous and ornamental changes must be scrupulously avoided. However, while accepting change requests, it must be remembered that the three critical project parameters: scope, schedule and project cost are interdependent and are very

intricately related. If the scope is allowed to change extensively, while strictly maintaining the schedule and cost, then the quality of the work would be the major casualty. Therefore, for every scope change request, the project managers examine whether the change request is really necessary and whether the budget and time schedule would permit it. Often, the scope change requests are superfluous. For example, an overenthusiastic project team member may suggest to add features that are not required by the customer. Such scope change requests originated by the overenthusiastic team members are called *goldplating* and should be discouraged if the project is to succeed. The customer may also initiate scope change requests that are more ornamental or at best nonessential. These serve only to jeopardize the success of the project, while not adding any perceptible value to the delivered software. Such avoidable scope change requests originated by the customer are termed as scope creep. To ensure the success of the project, the project manager needs to guard against both gold plating and scope creep.

4 B.

1.16 Project Management Life Cycle

The different phases of the project management life cycle are shown in Figure 1.8. In the following, we discuss the main activities that are carried out in each phase.

Project Initiation

As shown in Figure 1.8, the software project management life cycle starts with project initiation. The project initiation phase usually starts with project concept development. During concept development the different characteristics of the software to be developed are thoroughly understood. The different aspects of the project that are investigated and understood include: the scope of the project, project constraints, the cost that would be incurred and the benefits that would accrue. Based on this understanding, a feasibility study is undertaken to determine whether the project would be financially and technically feasible. This is true for all types of projects, including the in-house product development projects as well as the outsourced projects. For example, an organization might feel a need for a software to automate some of its activities, possibly for more efficient operation. Based on the feasibility study, the business case is developed. Once the top management agrees to the business case, the project manager is appointed, the project charter is written, and finally the project team is formed. This sets the ground for the manager to start the project planning phase.

As has already been pointed out, during the project initiation phase it is crucial for the champions of the project to develop a thorough understanding of the important characteristics of the project. In his W5HH principle, Barry Boehm summarized the questions that need to be asked and answered in order to have an understanding of these project characteristics.

W5HH Principle: Boehm suggested that during project initiation, the project champions should have comprehensive answers to a set of key questions pertaining to the project. The answers to these questions would lead to the definition of key project characteristics. The name of this principle (W5HH) is an acronym constructed from the first letter of each question. This set of seven questions is the following:

- Why is the software being built?
- What will be done?
- When will it be done?
- Who is responsible for a function?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource is needed?

Project bidding: Once an organization's top management is convinced by the business case, the project charter is developed. For some categories of projects, it may be necessary to have a formal bidding process to select a suitable vendor based on some cost-performance criteria. If the project involves automating some activities of an organization, the organization may either decide to develop it in-house or may get various software vendors to bid for the project. We briefly mention the different types of bidding techniques and their implications and applicability.

- *Request for quotation (RFQ)* An organization advertises an RFQ if it has good understanding of the project and the possible solutions. While publishing the RFQ, the organization would have to mention the scope of the work in a statement of work (SOW) document. Based on the RFQ different vendors can submit their quotations. The RFQ issuing organization can select a vendor based on the price quoted as

well as the competency of the vendor. In government organizations, the term request for tender (RFT) is usually used in place of RFQ. RFT is similar to RFQ; however, in RFT the bidder needs to deposit a tender fee in order to participate in the bidding process.

- *Request for proposal (RFP)* Many times it so happens that an organization has reasonable understanding of the problem to be solved, however it does not have a good grasp of the solution aspects. That is, the organization may not have sufficient knowledge about the different features that are to be implemented, and may lack familiarity with the possible choices of the implementation environment, such as, databases, operating systems, client-server deployment, etc. In this case, the organization may solicit solution proposals from vendors. The vendors may submit a few alternative solutions and the approximate costs for each solution. In order to develop a better understanding, the requesting organization may ask the vendors to explain or demonstrate their solutions. It needs to be understood that the purpose of RFP is to get an understanding of the alternative solutions possible that can be deployed and not vendor selection. Based on the RFP process, the requesting organization can form a clear idea of the project solutions required, based on which it can form a statement work (SOW) for requesting RFQ from the vendors.
- *Request for Information (RFI)* An organization soliciting bids may publish an RFI. Based on the vendor response to the RFI, the organization can assess the competencies of the vendors and shortlist the vendors who can bid for the work. However, it must be noted that vendor selection is seldom done based on RFI, but the RFI response from the vendors may be used in conjunction with RFP and RFQ responses for vendor selection.

Project planning

An important outcome of the project initiation phase is the project charter. During the project planning phase, the project manager carries out several processes and creates the following documents:

- *Project plan* This document identifies the project tasks, and a schedule for the project tasks that assigns project resources and time frames to the tasks.
- *Resource plan* It lists the resources, manpower and equipment that would be required to execute the project.
- *Financial plan* It documents the plan for manpower, equipment and other costs.
- *Quality plan* Plan of quality targets and control plans are included in this document.
- *Risk plan* This document lists the identification of the potential risks, their prioritization and a plan for the actions that would be taken to contain the different risks.

Project execution

In this phase the tasks are executed as per the project plan developed during the planning phase. A series of management processes are undertaken to ensure that the tasks are executed as per plan. Monitoring and control processes are executed to ensure that the tasks are executed as per plan and corrective actions are initiated whenever any deviations from the plan are noticed. However, the project plan may have to be revised periodically to accommodate any changes to the project plan that may arise on account of change requests, risks and various events that occur during the project execution. Quality of the deliverables is ensured through execution of proper processes. Once all the deliverables are produced and accepted by the customer, the project execution phase completes and the project closure phase starts.

5.

13.8 Product versus Process Quality Management

The measurements described above relate to *products*. With a product-based approach to planning and control, as advocated by the PRINCE2 project management method, this focus on products is convenient. However, we saw that it is often easier to measure these product qualities in a completed computer application rather than during its development. Trying to use the attributes of intermediate products created at earlier stages to predict the quality of the final application is difficult. An alternative approach is to scrutinize the quality of the *processes* used to develop software product.

The system development process comprises a number of activities linked so that the output from one activity is the input to the next (Figure 13.4). Errors can enter the process at any stage. They can be caused either by defects in a process, as when software developers make mistakes in the logic of their software, or by information not passing clearly and accurately between development stages.

Note that Extreme Programming advocates suggest that the extra effort needed to amend software at later stages can be exaggerated and is, in any case, often justified as adding value to the software.

Errors not removed at early stages become more expensive to correct at later stages. Each development step that passes before the error is found increases the amount of rework needed. An error in the specification found in testing will mean rework at all the stages between specification and testing. Each successive step of development is also more detailed and less able to absorb change.

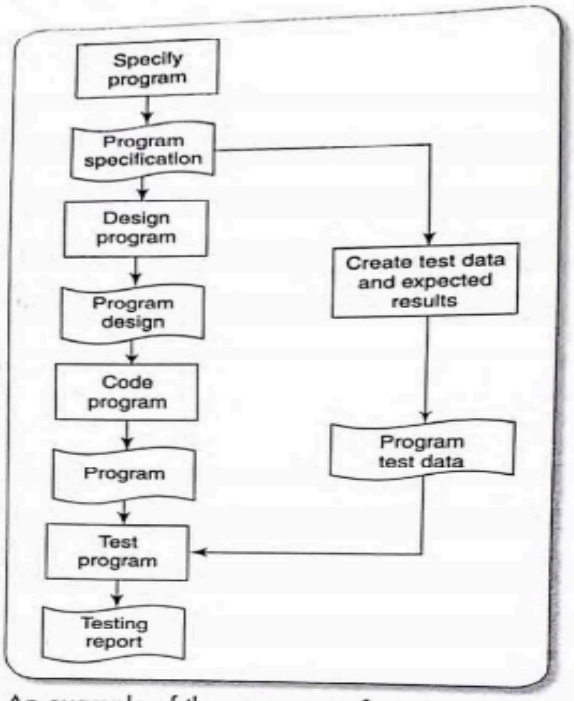


FIGURE 13.4 An example of the sequence of processes and deliverables

Errors should therefore be eradicated by careful examination of the deliverables of each step before they are passed on. One way of doing this is by having the following *process requirements* for each step.

- *Entry requirements*, which have to be in place before an activity can start. An example would be that a comprehensive set of test data and expected results be prepared and approved before program testing can commence.

These requirements may be laid out in installation standards, or a *Software Quality Plan* may be drawn up for the specific project if it is a major one.

- *Implementation requirements*, which define how the process is to be conducted. In the testing phase, for example, it could be laid down that whenever an error is found and corrected, all test runs must be repeated, even those that have previously been found to run correctly.
- *Exit requirements*, which have to be fulfilled before an activity is deemed to have been completed. For example, for the testing phase to be recognized as being completed, all tests will have to have been run successfully with no outstanding errors.

6.

13.3 Importance of Software Quality

We would expect quality to be a concern of all producers of goods and services. However, the special characteristics of software create special demands.

- *Increasing criticality of software* The final customer or user is naturally anxious about the general quality of software, especially its reliability. This is increasingly so as organizations rely more on their computer systems and software is used in more safety-critical applications, for example to control aircraft.
- *The intangibility of software* can make it difficult to know that a project task was completed satisfactorily. Task outcomes can be made tangible by demanding that the developer produce 'deliverables' that can be examined for quality.
- *Accumulating errors during software development* As computer system development comprises steps where the output from one step is the input to the next, the errors in the later deliverables will be added to those in the earlier steps, leading to an accumulating detrimental effect. In general, the later in a project that an error is found the more expensive it will be to fix. In addition, because the number of errors in the system is unknown, the debugging phases of a project are particularly difficult to control.

For these reasons quality management is an essential part of effective overall project management.

13.4 Defining Software Quality

In Chapter 1 we noted that a system has functional, quality and resource requirements. Functional requirements define what the system is to do, the resource requirements specify allowable costs and the quality requirements state how well this system is to operate.

Exercise 13.1

At Brightmouth College, Brigette has to select the best off-the-shelf payroll package for the college. How should she go about this in a methodical manner?

One element of the approach could be the identification of criteria against which payroll packages are to be judged. What might these criteria be? How could you check the extent to which packages match these criteria?

Some qualities of a software product reflect the external view of software held by *users*, as in the case of usability. These *external qualities* have to be mapped to *internal factors* of which the *developers* would be aware. It could be argued, for example, that well-structured code is likely to have fewer errors and thus improve reliability.

Defining quality is not enough. If we are to judge whether a system meets our requirements we need to be able to measure its qualities.

https://abdullahsurati.github.io/bscit

The BS ISO/IEC 15939:2007 standard *Systems and software engineering – measurement process* has codified many of the practices discussed in this section.

A good measure must relate the number of units to the maximum possible. The maximum number of faults in a program, for example, is related to the size of the program, so a measure of *faults per thousand lines of code* is more helpful than total *faults in a program*.

Trying to find measures for a particular quality helps to clarify and communicate what that quality really is. What is being asked is, in effect, 'how do we know when we have been successful?'

The measures may be *direct*, where we can measure the quality directly, or *indirect*, where the thing being measured is not the quality itself but an indicator that the quality is present. For example, the number of enquiries by users received by a help desk about how one operates a particular software application might be an indirect measurement of its usability.

Dromey's model: Dromey proposed that software product quality depends on four major high-level properties of the software: Correctness, internal characteristics, contextual characteristics and certain descriptive properties. Each of these high-level properties of a software product, in turn, depends on several lower-level quality attributes of the software. Dromey's hierarchical quality model is shown in Figure 13.2.

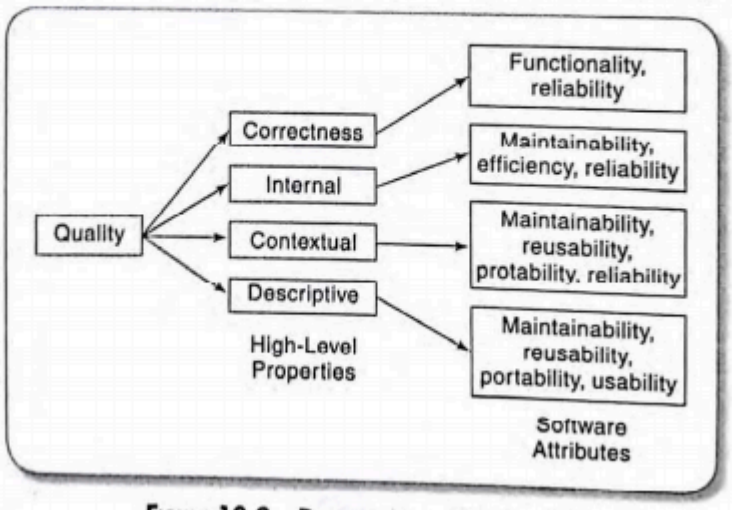


FIGURE 13.2 Dromey's quality model