**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**2 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - I**

***(May/June 2005)***

**1. Define software engineering paradigm.**

The software development strategy is referred as Software Engineering Paradigm. The software development strategy consists of methods, tools, and procedures. There exist various software development strategies or process models.

**2. Give the difference between “known risks” and “predictable risks”.**

**Known Risks:**

That can be not covered after careful computation of the project plan, business and technical environment in which the product is being build up.

 **Illustration: Impractical delivery rate.**

**Predictable Risks:**

 Extrapolated from the past project experience.

 **Illustration: Staff turn-over.**

***(Nov/Dec- 2007 R-2002)***

**3. List any two techniques for project scheduling.**

Project scheduling is concerned with the techniques that can be employed to manage the activities that need to be undertaken during the development of a project.

**4. What are the disadvantages of linear sequential model?**

* Once an application is in the [testing](http://istqbexamcertification.com/what-is-a-software-testing/) stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.

***(Nov/Dec- 2007 R-2005)***

**5. What is the difference between systems engineering and software engineering?**

**System engineering**

* It deals with all aspects of computer-based system development.
* It is to identify the roles of hardware, software, people, database and other system elements involved with that system which is going to be developed.
 **Software engineering**
* It is a part of system engineering.
It is to tell the practicalities of developing and delivering useful software.

***(Nov/Dec- 2009 R-2007)***

**6. Write the approaches for Software process assessment.**

The aim of process assessment is to identify the areas for improvement and suggest a plan for making that improvement. The main focus areas of process assessment are listed below.

* Obtaining guidance for improving software development and test processes
* Obtaining an independent and unbiased review of the process
* Obtaining a baseline (defined as a set of software components and documents that have been formerly reviewed and accepted; that serves as the basis for further development) for improving quality and productivity of processes.

**7. What is the objective of project planning?**

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of

* Resources
* Cost and
* Schedule

***(Nov/Dec- 2012 R-2009/2010)***

**8. Write the advantages of prototyping.**

**Reduced time and costs:** Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software.

**Improved and increased user involvement:** Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the [problem domain](http://en.wikipedia.org/wiki/Problem_domain) better than anyone on the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the users desire for look, feel and performance.

***(Aug- 2012 R-2007)***

**9. What is Software Engineering?**

**Software Engineering:**

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

**10. How does Personal Software Process(PSP) differ from Team Software Process(TSP)?**

The Personal Software Process (PSP) is an SEI technology that brings discipline to the practices of individual software engineers, dramatically improving product quality, increasing cost and schedule predictability, and reducing development cycle time for software.

The Team Software Process (TSP) is a complementary SEI technology that enables teams to develop software-intensive products more effectively. TSP shows a team of engineers how to produce quality products for planned costs and on aggressive schedules**.**

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**2 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - II**

***(May-June 2005)***

**1. Define the term abstraction in software engineering.**

The process of picking out (*abstracting*) common features of [objects](http://www.webopedia.com/TERM/O/object.html) and procedures. A programmer would use abstraction, for example, to note that two functions perform almost the same task and can be combined into a single function. Abstraction is one of the most important techniques in [software engineering](http://www.webopedia.com/TERM/S/software_engineering.html) and is closely related to two other important techniques -- [*encapsulation*](http://www.webopedia.com/TERM/E/encapsulation.html) and [*information hiding*](http://www.webopedia.com/TERM/I/information_hiding.html). All three techniques are used to reduce complexity.

**2. Give the difference between “known risks” and “predictable risks”.**

**Known Risks:**

That can be not covered after careful computation of the project plan, business and technical environment in which the product is being build up.

**Illustration: Impractical delivery rate.**

**Predictable Risks:**

 Extrapolated from the past project experience.

 **Illustration: Staff turn-over.**

**3. How are the concepts of coupling and software portability related?**

The concept of coupling embodies the reliability and dependability (strength) among all relationships betweenfunctional units.

The concept of software portability is the usability of same components/software in different environments.
 For optimal software, it is generally the case where coupling has to be minimized and portability has to be maximized (for sustainability and effectiveness in the long run). Having said that, strong software that contains low levels of coupling could easily and efficiently is portable. Similarly, if the software is portable it is probably due to the low levels of coupling that it has.

***(Nov/Dec- 2007 R-2002)***

**4. Define Coupling.**

Coupling is the manner and degree of interdependence between software modules; a measure of how closely connected two routines or modules are;[[1]](https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29#cite_note-ISO_24765-1)the strength of the relationships between modules

**5. What are the characteristics of a good design.**

**Understanding:**Do I know what to do, how to do it, and is the system response clear.

 **Meaning:** Are the functions of the system meaningful to me, my life and to my cultural context.
 **Value :** While this feels close to meaning, value implies an economy that meaning doesn't require. Even if time & effort are the only currency of the system, there needs to be a perception of value foran interactiondesign to be good.
**Engagement :** This has many components to it, but in the end the system has to be one brings me in on both a cognitive and an emotional level. But for me it is best described in having an element that pulls your attention, and can even increase your motivation for participation.
 **Fit :** While Meaning is very contextual, things that are meaningful don't always fit against the other behaviours in our lives. It is important to fit on many levels, or if you are to disrupt, the disruption needs to use novelty in order to create a new "fit". If you think to yourself, I can't imagine ever not having done things this way before, it's a good fit.
**Emotion :** How you feel is going to make all the difference. Do I know what to do, how to do it, and is the system response clear.

***(Nov/Dec- 2007 R-2005)***

**6. Enumerate the fundamental software design concepts**

The design process comprises a set of principles, concepts and practices, which allow a software engineer to model the system or product that is to be built. This model, known as design model, is assessed for quality and reviewed before a code is generated and tests are conducted. The design model provides details about software data structures, architecture, interfaces and components which are required to implement the system.

***(Nov/Dec- 2009 R-2007)***

**7. What are use-cases?**

A use case in software engineering is a description of a system’s behaviour as it responds to a request that originates from outside of that system. In other words, a use case describes “who” can do “what” with the system in question. The use case technique is used to capture a system’s behavioural requirements by detailing scenario-driven threads through the functional requirements.

**8. What is the major goal of Architectural design?**

Software design is the process by which an [agent](https://en.wikipedia.org/wiki/Agency_%28philosophy%29) creates a specification of a [software artifact](https://en.wikipedia.org/wiki/Artifact_%28software_development%29), intended to accomplish [goals](https://en.wikipedia.org/wiki/Goal), using a set of primitive components and subject to [constraints](https://en.wikipedia.org/wiki/Constraint_%28mathematics%29).

 [[1]](https://en.wikipedia.org/wiki/Software_design#cite_note-1) Software design may refer to either "all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" or "the activity following [requirements](https://en.wikipedia.org/wiki/Software_requirements) specification and before [programming](https://en.wikipedia.org/wiki/Computer_programming), as ... [in] a stylized software engineering process."

 [[2]](https://en.wikipedia.org/wiki/Software_design#cite_note-2) Software design usually involves problem solving and planning a [software](https://en.wikipedia.org/wiki/Software)solution. This includes both low-level component and [algorithm design](https://en.wikipedia.org/wiki/Algorithm_design) and high-level, [architecture](https://en.wikipedia.org/wiki/Software_architecture) design.

**9. Give the importance of Modularity in software design.**

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of ‘divide and conquer’ problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

***(Nov/Dec- 2012 R-2009/2010)***

**10. Define abstraction and modularity.**

**Modularity:**

In software engineering, modularity refers to the extent to which a software/Web application may be divided into smaller modules. Software modularity indicates that the number of application modules is capable of serving a specified business domain.

Modularity is successful because developers use prewritten code, which saves resources. Overall, modularity provides greater software development manageability.

**Abstraction:**

Abstraction is the act of representing essential features without including the background details or explanations. In the computer science and software engineering domain, the abstraction principle is used to reduce complexity and allow efficient design and implementation of complex software systems.

**11. What are the characteristics of a good software design.**

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

* Operational
* Transitional
* Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

* Budget
* Usability
* Efficiency
* Correctness
* Functionality
* Dependability
* Security
* Safety

Transitional

This aspect is important when the software is moved from one platform to another:

* Portability
* Interoperability
* Reusability
* Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

* Modularity
* Maintainability
* Flexibility
* Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

***(Aug- 2012 R-2007)***

**12. What is the significance of DFD in software design?**

The DFD aims to capture the transformations that take place within a system to the input data so that eventually the output data is produced. The agent that performs the transformation of data from one state to another is called a process (or a bubble). Thus, a DFD shows the movement of data through the different transformations or processes in the system. The processes are shown by named circles and data flows are represented by named arrows entering or leaving the bubbles. A rectangle represents a source or sink and is a net originator or consumer of data.

**13. Define Modularity. What is the advantage of modularity in design?**

In software engineering, modularity refers to the extent to which a software/Web application may be divided into smaller modules. Software modularity indicates that the number of application modules is capable of serving a specified business domain.

Modularity is successful because developers use prewritten code, which saves resources. Overall, modularity provides greater software development manageability.

**14. Define Coupling.**

Coupling is the act of joining two things together.  In software development, coupling refers to the degree to which software components are dependent upon each other.  For instance, in a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled.  In a [loosely-coupled](http://searchnetworking.techtarget.com/definition/loose-coupling) architecture, components can remain autonomous and allow [middleware](http://searchsoa.techtarget.com/definition/middleware) software to manage communication between them. In a decoupled architecture, the components can operate completely separately and independently.

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**2 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - III**

***(May-June 2005)***

**1. What are the three types of maintenance? Which one of them consumes more effort? Why?**

There are four types of maintenance, namely, **corrective, adaptive, perfective, and preventive.**

Corrective maintenance is concerned with fixing errors that are observed when the software is in use. Adaptive maintenance is concerned with the change in the software that takes place to make the software adaptable to new environment such as to run the software on a new [operating system](http://ecomputernotes.com/fundamental/disk-operating-system/what-is-operating-system). Perfective maintenance is concerned with the change in the software that occurs while adding new functionalities in the software. Preventive maintenance involves implementing changes to prevent the occurrence of errors. The distribution of types of maintenance by type and by percentage of time consumed.

***(Nov/Dec- 2007 R-2002)***

**2. Why testing is important in software development?**

The major role of software testing involves that there should be no discrepancy in the software development process. According to one survey software errors costs U.S economy 0.6 percent of the gross domestic product and about 80% of the software development costs of a project are spent on identifying and fixing errors

**3. What is beta testing?**

A beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. (Beta is the second letter of the Greek alphabet.) Originally, the term alpha test meant the first phase of testing in a software development process. The first phase includes unit testing, component testing, and system testing. Beta testing can be considered "pre-release testing." Beta test versions of software are now distributed to a wide audience on the Web partly to give the program a "real-world" test and partly to provide a preview of the next release.

***(Nov/Dec- 2007 R-2005)***

**4. Define testability.**

Software testability has historically referred to the ease with which inputs can be selected to satisfy a specific structural testing criterion (e.g., branch coverage). With this definition, if it were extremely difficult to find inputs that satisfied a particular structural coverage criterion for a given program, then the program would be labeled as having “low testability.”

**5. Differentiate between Errors and Defects.**

**An error**:
A human action that produces an incorrect result.
Programmatically mistake leads to error.
bug:
An informal word describing any of the above.
Deviation from the expected result.
A software bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from working as intended, or produces an incorrect result. Bugs arise from mistakes and errors, made by people, in either a program's source code or its design. It is said that there are bugs in all useful computer programs, but well-written programs contain relatively few bugs, and these bugs typically do not prevent the program from performing its task. A program that contains a large number of bugs, and/or bugs that seriously interfere with its functionality, is said to be buggy. Reports about bugs in a program are referred to as bug reports, also called PRs (problem reports), trouble reports, CRs (change requests), and so forth.
Defect:
Problem in algorithm leads to failure.
A defect is for something that normally works, but it has something out-of-spec.

***(Nov/Dec- 2009 R-2007)***

**6. Give the checklist for while validating the requirements**.

**7. Bring out the difference between Alpha testing and Beta testing.**

|  |  |
| --- | --- |
| **Alpha Testing** | **Beta Testing** |
| Alpha testing performed by Testers  who are usually internal employees of the organization | Beta testing is performed by Clients or End Users who are not employees of the organization |
| Alpha Testing performed at developer's site  | Beta testing is performed at client location or end user of the product  |
| Reliability and security testing are not performed  in-depth Alpha Testing  | Reliability, Security, Robustness are checked during Beta Testing  |
| Alpha testing involves both the white box and black box techniques | Beta Testing typically uses black box testing  |
| Alpha testing requires lab environment or testing environment  | Beta testing doesn't require any lab environment or testing environment. Software is made available to the public and  is said to be real time environment  |
| Long execution cycle may be required for Alpha testing  | Only few weeks of execution are required for Beta testing  |
| Critical issues or fixes can be addressed by developers immediately in Alpha testing  | Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product  |
| Alpha testing is to ensure the quality of the product before moving to Beta testing | Beta testing also concentrates on quality of the product, but gathers users input on the product and ensures that the product is ready for real time users. |

***(Nov/Dec- 2012 R-2009/2010)***

**8. Define: White Box Testing.**

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing [software](https://en.wikipedia.org/wiki/Software) that tests internal structures or workings of an application, as opposed to its functionality (i.e. [black-box testing](https://en.wikipedia.org/wiki/Black-box_testing)). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. [in-circuit testing](https://en.wikipedia.org/wiki/In-circuit_test) (ICT).

White-box testing can be applied at the [unit](https://en.wikipedia.org/wiki/Unit_testing), [integration](https://en.wikipedia.org/wiki/Integration_testing) and [system](https://en.wikipedia.org/wiki/System_testing) levels of the [software testing](https://en.wikipedia.org/wiki/Software_testing) process.

**9. What is Test Case Management?**

***(Aug- 2012 R-2007)***

**10. What are the different levels at which testing is done?**

**11. Differentiate between alpha and beta testing.**

**Alpha testing** performed by Testers  who are usually internal employees of the organization

**Beta testing** is performed by Clients or End Users who are not employees of the organization

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**2 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - IV**

***(May-June 2005)***

**1.What is the need for cyclomatic complexity?**

Cyclomatic complexity defines the number of independent paths in the basis set of a program and provides you with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

Cyclomatic complexity has a foundation in graph theory and provides you with an extremely useful software metric. Complexity is computed in one of three ways:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.

2. Cyclomatic complexity V(G) for a flow graph G is defined as V(G) \_ E \_ N \_ 2

where E is the number of flow graph edges and N is the number of flow graph nodes.

3. Cyclomatic complexity V(G) for a flow graph G is also defined as V(G) \_ P \_ 1

where P is the number of predicate nodes contained in the flow graph G.

***(Nov/Dec- 2007 R-2002)***

**2. What do you understand by direct measure and indirect measure with respect to software?**

**Direct measures** of the software engineering process include cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.

**Indirect measures** of the product include functionality, quality, complexity, efficiency, reliability, maintainability, and many other "–abilities"

**3. Define software reliability.**

Software reliability is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time"

***(Nov/Dec- 2007 R-2005)***

**4. How do you define cyclomatic complexity?**

Complexity is computed in one of three ways:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.

2. Cyclomatic complexity V(G) for a flow graph G is defined as V(G) \_ E \_ N \_ 2

where E is the number of flow graph edges and N is the number of flow graph nodes.

3. Cyclomatic complexity V(G) for a flow graph G is also defined as V(G) \_ P \_ 1

where P is the number of predicate nodes contained in the flow graph G.

**5. Name the measures for reliability in software. Refer Q.No:3**

***(Nov/Dec- 2009 R-2007)***

**6. How do you define Cyclomatic complexity? Refer Q.No:4**

**7. Why we need integration testing?**

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.

***(Nov/Dec- 2012 R-2009/2010)***

**8. What are direct and indirect measures? Refer Q.No:2**

***(Aug- 2012 R-2007)***

**9. Define: Software Quality.**

Software quality is a complex mix of factors that will vary across different applications and the customers who request them.

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**2 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - V**

***(May-June 2005)***

**1. What is meant by Version control in software engineering?**

A *version control system* (also known as a *Revision Control System*) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced, by the change.

**2. Mention the advantages of CASE tools.**

CASE stands for Computer Aided Software Engineering which is software that supports one or more software engineering activities within a software development process, and is gradually becoming popular for the development of software as they are improving in the capabilities and **functionality** and are proving to be beneficial for the development of quality software.

**Advantages and Disadvantages of CASE Tools:**

***(Nov/Dec- 2007 R-2002)***

**3.What is version control?**

A *version control system* (also known as a *Revision Control System*) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced, by the change.

**4. State the objectives of test plan.**

Test plan is the project plan for the [testing work](http://istqbexamcertification.com/what-is-fundamental-test-process-in-software-testing/) to be done. It is not a [test design](http://istqbexamcertification.com/what-is-test-design-technique/) **specification,** a collection of **testcases** or a set of **test procedures;** in fact, most of our test plans do not address that level of detail. Many people have different definitions for test plans.

***(Nov/Dec- 2009 R-2007)***

**5. Define Software safety.**

**software system safety** optimizes system safety in the design, development, use, and maintenance of [software](https://en.wikipedia.org/wiki/Software) systems and their integration with safety-critical hardware systems in an operational environment.

***(Nov/Dec- 2012 R-2009/2010)***

**6. What are the tasks in the SCM Process.**

A process defines the steps by which you perform a specific task or set of tasks. An SCM process is the way SCM is performed on your project—specifically, how an SCM tool is applied to accomplish a set of tasks.

A key mistake most people make is to assume that an SCM tool will, in and of itself, solve their SCM problems or support their SCM requirements. This is wrong! The picture will not hang itself if you buy a hammer and nails. It is not the tool itself that solves a problem, but rather the application of that tool. How you apply the SCM tool to your development environment is called the usage model, or SCM process. It is this model or process that will in part determine how successfully you address your SCM issues.

***(Aug- 2012 R-2007)***

**7. What is the purpose of Version Control?**

A *version control system* (also known as a *Revision Control System*) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced, by the change.

***(Aug- 2012 R-2007)***

**8. What are called as “Software Configuration Items”?**

A **software configuration item (SCI)** is an aggregation of software that is designated for configuration management and is treated as a single entity in the SCM process [IEEE 610]. A variety of items, in addition to the code itself, are typically controlled by SCM. Software items with potentialto become SCIs include plans, specifications and design documentation, testing materials, software tools, source and executable code, code libraries, data and data dictionaries, and documentation for installation, maintenance, operations and software use.

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**16 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - I**

***(May-June 2005)***

**1. Explain linear life cycle model in detail.Give the advantages and disadvantages of the same.**

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**.  It is very simple to understand and use.  In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

**Diagram of Waterfall-model:**



**Advantages of waterfall model:**

* This model is simple and easy to understand and use.
* It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
* In this model phases are processed and completed one at a time. Phases do not overlap.
* Waterfall model works well for smaller projects where requirements are very well understood.

**Disadvantages of waterfall model:**

* Once an application is in the [testing](http://istqbexamcertification.com/what-is-a-software-testing/) stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.

**When to use the waterfall model:**

* This model is used only when the requirements are very well known, clear and fixed.
* Product definition is stable.
* Technology is understood.
* There are no ambiguous requirements
* Ample resources with required expertise are available freely
* The project is short.

**2. What are the various types of cost estimation models?Explain COCOMO model in detail.**

|  |  |
| --- | --- |
|  | [1. Abstract](http://www.computing.dcu.ie/~renaat/ca421/report.html#abstract)[2. Introduction](http://www.computing.dcu.ie/~renaat/ca421/report.html#introduction) |
|  |  | [2.1 Defining Cost Estimation](http://www.computing.dcu.ie/~renaat/ca421/report.html#2.1) [2.2 Cost Estimation and Project Planning](http://www.computing.dcu.ie/~renaat/ca421/report.html#2.2) [2.3 Cost Estimation During the Software Life Cycle](http://www.computing.dcu.ie/~renaat/ca421/report.html#2.6) [2.4 The Estimator](http://www.computing.dcu.ie/~renaat/ca421/report.html#2.4) [2.5 General Steps and Remarks](http://www.computing.dcu.ie/~renaat/ca421/report.html#2.5) [2.6 Document Overview](http://www.computing.dcu.ie/~renaat/ca421/report.html#2.6) |
|  | [3. Cost Estimation Process](http://www.computing.dcu.ie/~renaat/ca421/report.html#3) |
|  |  | [3.1 Classical View](http://www.computing.dcu.ie/~renaat/ca421/report.html#3.1) [3.2 Actual View](http://www.computing.dcu.ie/~renaat/ca421/report.html#3.2) [3.3 Cost Estimation Accuracy](http://www.computing.dcu.ie/~renaat/ca421/report.html#3.3) |
|  | [4. Methods of Cost Estimation](http://www.computing.dcu.ie/~renaat/ca421/report.html#4) |
|  |  | [4.1 Algorithmic (Parametric) Model](http://www.computing.dcu.ie/~renaat/ca421/report.html#4.1) [4.2 Expert Judgement](http://www.computing.dcu.ie/~renaat/ca421/report.html#4.2) [4.3 Top-Down](http://www.computing.dcu.ie/~renaat/ca421/report.html#4.3) [4.4 Bottom-Up](http://www.computing.dcu.ie/~renaat/ca421/report.html#4.4) [4.5 Estimation by Analogy](http://www.computing.dcu.ie/~renaat/ca421/report.html#4.5) [4.6 Price to Win](http://www.computing.dcu.ie/~renaat/ca421/report.html#4.6) |
|  | [5. COCOMO](http://www.computing.dcu.ie/~renaat/ca421/report.html#5) |
|  |  | [5.1 COCOMO 81](http://www.computing.dcu.ie/~renaat/ca421/report.html#5.1) |
|  |  |  | [5.1.1 Equations Used](http://www.computing.dcu.ie/~renaat/ca421/report.html#5.1.1) [5.1.2 Cost Drivers](http://www.computing.dcu.ie/~renaat/ca421/report.html#5.1.2) |
|  |  | [5.2 COCOMO II](http://www.computing.dcu.ie/~renaat/ca421/report.html#5.2) |
|  |  |  | [5.2.1 Cost Drivers](http://www.computing.dcu.ie/~renaat/ca421/report.html#5.2.1) ,[5.2.2 Calibration](http://www.computing.dcu.ie/~renaat/ca421/report.html#5.2.2) |

***(Nov/Dec- 2007 R-2002)***

**3. Explain in detail the spiral model with a neat sketch and discuss its merits and demerits.**

The spiral model is similar to the [incremental model](http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/), with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

**Planning Phase:** Requirements are gathered during the planning phase. Requirements like ‘BRS’ that is ‘Bussiness Requirement Specifications’ and ‘SRS’ that is ‘System Requirement specifications’.

**Risk Analysis:** In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions.  A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

**Engineering Phase:** In this phase software is **developed**, along with [testing](http://istqbexamcertification.com/what-is-a-software-testing/) at the end of the phase. Hence in this phase the development and testing is done.

E**valuation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

**Diagram of Spiral model:**



**Advantages of Spiral model:**

* High amount of risk analysis hence, avoidance of Risk is enhanced.
* Good for large and mission-critical projects.
* Strong approval and documentation control.
* Additional Functionality can be added at a later date.
* Software is produced early in the [software life cycle](http://istqbexamcertification.com/what-are-the-software-development-life-cycle-phases/).

**Disadvantages of Spiral model:**

* Can be a costly model to use.
* Risk analysis requires highly specific expertise.
* Project’s success is highly dependent on the risk analysis phase.
* Doesn’t work well for smaller projects.

**When to use Spiral model:**

* When costs and risk evaluation is important
* For medium to high-risk projects
* Long-term project commitment unwise because of potential changes to economic priorities
* Users are unsure of their needs
* Requirements are complex
* New product line
* Significant changes are expected (research and exploration)

**4. Diagrammatically illustrate and discuss the waterfall model for software development.**

**Refer Q.No:1**

***(Nov/Dec- 2007 R-2005)***

**5. Categories the iterative models for software process. Explain each of them in detail.**

***(Nov/Dec- 2009 R-2007)***

**6. Explain the concept of waterfall model in Software development.**

**7. Discuss of different COCOMO(Cost Constructive Model) for cost estimation.Refer Q.No:2**

**8. What are the different activities in Spiral Model?ExplainRefer Q.No:2**

**9. How Software risks are assessed? Explain strategies for their containment.**

• Anticipate and Identify risk

• Minimize the impact / damage / loss

 • Reduce the probability

 • Monitor risk areas for early detection

• Ensure management awareness of risks

**10. List out and explain the Requirement Engineering tasks.**

* Problems with requirements practices
* Requirements engineering tasks
* Inception
* Elicitation
* Elaboration
* Negotiation
* Specification
* Validation
* Requirements management

***(Nov/Dec- 2012 R-2009/2010)***

**11. Explain any one method from Evolutionary and Incremental Process models used in Software development? What are the advantages and disadvantages of various Software development life cycle models.**

The software industry includes many different processes, for example, analysis, development, maintenance and publication of software. This industry also includes software services, such as training, documentation, and consulting.

Our focus here about [software development life cycle](http://en.wikipedia.org/wiki/Software_development_process) ([SDLC](http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle)). So, due to that different types of projects have different requirements. Therefore, it may be required to choose the SDLC phases according to the specific needs of the project. These different requirements and needs give us various software development approaches to choose from during software implementation.

# Types of Software developing life cycles (SDLC)

· [Waterfall Model](http://en.wikipedia.org/wiki/Waterfall_model)

· V-Shaped Model

· [Evolutionary Prototyping](http://en.wikipedia.org/wiki/Software_prototyping) Model

· [Spiral](http://en.wikipedia.org/wiki/Spiral_model) Method ([SDM](http://en.wikipedia.org/wiki/Software_development_methodology))

· [Iterative and Incremental](http://en.wikipedia.org/wiki/Iterative_and_incremental_development) Method

· [Extreme programming](http://en.wikipedia.org/wiki/Extreme_Programming) ([Agile development](http://en.wikipedia.org/wiki/Agile_software_development))

## Waterfall Model

##### Description

The waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach that was used for software development.



Projects did not focus on changing requirements, for example, responses for request for proposals ([RFPs](http://en.wikipedia.org/wiki/Request_for_proposal))

##### Advantages and Disadvantages

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| · Easy to explain to the user· Structures approach.· Stages and activities arewell defined· Helps to plan and schedule the project· Verification at each stage ensures early detection of errors / misunderstanding· Each phase has specific deliverables | · Assumes that the requirements of a system can be frozen· Very difficult to go back to any stage after it finished.· Little flexibility and adjusting scope is difficult and expensive.· Costly and required more time, in addition to detailed plan |

## V-Shaped Model

##### Description

It is an extension for waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The major difference between v-shaped model and waterfall model is the early test planning in v-shaped model.



##### The usage

· Software requirements clearly defined and known

· Software development technologies and tools is well-known

##### Advantages and Disadvantages

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| · Simple and easy to use.· Each phase has specific deliverables.· Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.· Works well for where requirements are easily understood. | · Very inflexible, like the waterfall model.· Little flexibility and adjusting scope is difficult and expensive.· Software is developed during the implementation phase, so no early prototypes of the software are produced.· Model doesn’t provide a clear path for problems found during testing phases.· Costly and required more time, in addition to detailed plan |

## Evolutionary Prototyping Model

##### Description

It refers to the activity of creating prototypes of software applications, for example, incomplete versions of the software program being developed. It is an activity that can occur in software development. It used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team. This also will reduce the iterations may occur in waterfall approach and hard to be implemented due to inflexibility of the waterfall approach. So, when the final prototype is developed, the requirement is considered to be frozen.

It has some types, such as:

· Throwaway prototyping: Prototypes that are eventually discarded rather than becoming a part of the finally delivered software



· Evolutionary prototyping: prototypes that evolve into the final system through iterative incorporation of user feedback.



· Incremental prototyping: The final product is built as separate prototypes. At the end the separate prototypes are merged in an overall design.



· Extreme prototyping: used at web applications mainly. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase the services are implemented

##### The usage

· This process can be used with any software developing life cycle model. While this shall be focused with systems needs more user interactions. So, the system do not have user interactions, such as, system does some calculations shall not have prototypes.

##### Advantages and Disadvantages

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| · Reduced time and costs, but this can be disadvantage if the developer lose time in developing the prototypes· Improved and increased user involvement | · Insufficient analysis· User confusion of prototype and finished system· Developer misunderstanding of user objectives· Excessive development time of the prototype· Expense of implementing prototyping |

## Spiral Method (SDM)

##### Description

It is combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. This model of development combines the features of the prototyping model and the waterfall model. The spiral model is favored for large, expensive, and complicated projects. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.



##### The usage

It is used in shrink-wrap application and large system which built-in small phases or segments.

##### Advantages and Disadvantages

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| · Estimates (i.e. budget, schedule, etc.) become more realistic as work progresses, because important issues are discovered earlier.· Early involvement of developers· Manages risks and develops system into phases | · High cost and time to reach the final product.· Needs special skills to evaluate the risks and assumptions· Highly customized limiting re-usability |

## Iterative and Incremental Method

##### Description

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during development of earlier parts or versions of the system.

It consists of mini waterfalls



##### The usage

It is used in shrink-wrap application and large system which built-in small phases or segments. Also can be used in system has separated components, for example, ERP system. Which we can start with budget module as first iteration and then we can start with inventory module and so forth.

##### Advantages and Disadvantages

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| · Produces business value early in the development life cycle· Better use of scarce resources through proper increment definition· Can accommodate some change requests between increments· More focused on customer value than the linear approaches· Problems can be detected earlier | · Requires heavy documentation· Follows a defined set of processes· Defines increments based on function and feature dependencies· Requires more customer involvement than the linear approaches· Partitioning the functions and features might be problematic· Integration between iteration can be an issue if this is not considered during the development. |

## Extreme programming (Agile development)

##### Description

It is based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams.



##### The usage

It can be used with any type of the project, but it needs more involvement from customer and to be interactive. Also, it can be used when the customer needs to have some functional requirement ready in less than three weeks.

##### Advantages and Disadvantages

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| · Decrease the time required to avail some system features.· Face to face communication and continuous inputs from customer representative leaves no space for guesswork.· The end result is the high quality software in least possible time duration and satisfied customer | · Scalability· Skill of the software developers· Ability of customer to express user needs· Documentation is done at later stages· Reduce the usability of components.· Needs special skills for the team. |

**12. Identify and explain the various the suitable software life cycle model for weather monitoring system.**

***(Aug- 2012 R-2007)***

**13. Describe the different requirement engineering tasks.Refer Q.No:10**

**14. Discuss the elements of analysis model.**

* Requirements analysis
* Flow-oriented modeling
* Scenario-based modeling
* Class-based modeling
* Behavioral modeling

**15. Explain the data modelling concepts in detail.**

A data model can be thought of as a diagram or [flowchart](http://whatis.techtarget.com/definition/flowchart) that illustrates the relationships between data. Although capturing all the possible relationships in a data model can be very time-intensive, it's an important step and shouldn't be rushed. Well-documented models allow stake-holders to identify errors and make changes *before* any programming [code](http://whatis.techtarget.com/definition/code) has been written.

Data modelers often use multiple models to view the same data and ensure that all processes, entities, relationships and data flows have been identified. There are several different approaches to data modeling, including:

**Conceptual Data Modeling** - identifies the highest-level relationships between different entities.

**Enterprise Data Modeling** - similar to conceptual data modeling, but addresses the uniquerequirements of a specific business.

**Logical Data Modeling** - illustrates the specific entities, attributes and relationships involved in a business function. Serves as the basis for the creation of the physical data model.

**Physical Data Modeling** - represents an application and database-specific implementation of a logical data model.

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**16 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - II**

***(May-June 2005)***

**1. Explain the set of principles for software engineering design.**

The set of principles which has been established to aid the software engineer in navigating the design process are:

1.      The design process should not suffer from tunnel vision – A good designer should consider alternative approaches. Judging each based on the requirements of the problem, the resources available to do the job and any other constraints.

2.      The design should be traceable to the analysis model – because a single element of the design model often traces to multiple requirements, it is necessary to have a means of tracking how the requirements have been satisfied by the model

3.      The design should not reinvent the wheel – Systems are constructed using a set of design patterns, many of which may have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited! Design time should be invested in representing truly new ideas and integrating those patterns that already exist.

4.      The design should minimise intellectual distance between the software and the problem as it exists in the real world – That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.

5.      The design should exhibit uniformity and integration – a design is uniform if it appears that one person developed the whole thing. Rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.

6.      The design should be structured to degrade gently, even with bad data, events, or operating conditions are encountered – Well-designed software should never “bomb”. It should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.

7.      The design should be reviewed to minimize conceptual (semantic) errors – there is sometimes the tendency to focus on minute details when the design is reviewed, missing the forest for the trees. The designer team should ensure that major conceptual elements of the design have been addressed before worrying about the syntax if the design model.

8.      Design is not coding, coding is not design – Even when detailed designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made of the coding level address the small implementation details that enable the procedural design to be coded.

9.      The design should be structured to accommodate change

10.  The design should be assessed for quality as it is being created

When these design principles are properly applied, the design exhibits both external and internal quality factors. External quality factors are those factors that can readily be observed by the user, (e.g. speed, reliability, correctness, usability). Internal quality factors relate to the technical quality (which is important to the software engineer) more so the quality of the design itself. To achieve internal quality factors the designer must understand basic design concepts.

**2. Describe the concept of information hiding.**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), information hiding is the principle of segregation of the design decisions in a [computer program](https://en.wikipedia.org/wiki/Computer_program) that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed. The protection involves providing a stable [interface](https://en.wikipedia.org/wiki/Interface_%28computer_science%29) which protects the remainder of the program from the implementation (the details that are most likely to change).

Written another way, information hiding is the ability to prevent certain aspects of a [class](https://en.wikipedia.org/wiki/Class_%28computer_science%29) or [software component](https://en.wikipedia.org/wiki/Software_component) from being accessible to its [clients](https://en.wikipedia.org/wiki/Client_%28computing%29), using either programming language features (like private variables) or an explicit exporting policy.

The term *encapsulation* is often used interchangeably with information hiding. Not all agree on the distinctions between the two though; one may think of information hiding as being the principle and encapsulation being the technique. A software module hides information by encapsulating the information into a module or other construct which presents an interface.[[1]](https://en.wikipedia.org/?title=Information_hiding#cite_note-not_encaps-1)

A common use of information hiding is to hide the physical storage layout for data so that if it is changed, the change is restricted to a small subset of the total program. For example, if a three-dimensional point (x,y,z) is represented in a program with three [floating point](https://en.wikipedia.org/wiki/Floating_point)[scalar](https://en.wikipedia.org/wiki/Scalar_%28computing%29) variables and later, the representation is changed to a single [array](https://en.wikipedia.org/wiki/Array_data_structure) variable of size three, a module designed with information hiding in mind would protect the remainder of the program from such a change.

In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), information hiding (by way of [nesting](https://en.wikipedia.org/wiki/Nesting_%28computing%29) of types) reduces software development risk by shifting the code's [dependency](https://en.wikipedia.org/wiki/Dependency_%28computer_science%29) on an uncertain implementation (design decision) onto a well-defined [interface](https://en.wikipedia.org/wiki/Interface_%28computer_science%29). Clients of the interface perform operations purely through it so if the implementation changes, the clients do not have to change.

**3. Explain various design concepts in detail.**

***Design Principles and Concepts***

[**An introduction to Software Design**](http://www.vocw.edu.vn/content/m10075/latest/)A brief tutorial on software design and relevant terminology.

[**Design - Concepts and Principles**](http://scitec.uwichill.edu.bb/cmp/online/cs22l/design_-_concepts_and_principles.htm)An abbreviated discussion that can serve as a basic overview.

[**What are Software Design Principles?**](http://www.oodesign.com/design-principles.html)Brief but worthwhile discussion of selected design principles.

[**Design principles**](https://www-01.ibm.com/software/ucd/designconcepts/designbasics.html)Pragmatic design guidelines proposed at the IBM developer site.
[**Software design principles**](http://rearchitect.wordpress.com/2006/02/12/software-design-principles-mined-from-books-and-papers/)

Excellent lists of design principles. Will require further expansion, but interesting to see how authors differ in their emphasis.

***(Nov/Dec- 2007 R-2002)***

**4. Discuss in detail the various design concepts with respect to software. Refer Q.No:3**

***(Nov/Dec- 2007 R-2005)***

**5. Explain in detail about the pattern based software design.**

#### What is pattern-based software development?

What was the original impetus behind the development of software development patterns, and why do we need them? Why did programmers invent patterns for software development?

Well, developing software is very difficult, and developing software that can be easily reused is even harder. the designs for sections of software code should be general enough solutions to be able to address future problems and requirements flexibly while still being specific enough in order to address the current problem at hand. Programmers that are experienced at designing software systems know better than to design their system using one-off problem solutions, and instead reuse patterns that they have grown familiar with through prior use in similar situations and scenarios and reuse these solutions as a basis for their new designs. The basic fundamental principle of software engineering known as the “Principle of generality” predicts and encourages this behavior.

#### What’s so great about programmers using pattern-based development on software projects?

For one thing, it is absolutely fascinating to sit in a meeting room with a group of programmers who have been working all together on a software development project using patterns for a few months. The rate of information exchange is extremely high, with a idea mentioned by one programmer, and a few others simultaneously finishing the first programmer’s sentence with an exclaimed, unison word like “Bridge!”, and then one of them scribbling lines of code frantically on the whiteboard as the rest nod in compliment.

The language of the programming team using patterns is mysterious and magical, almost like incantations spoken in some artful black language. Many computer science instructors contend with conviction that the teaching of patterns and the learning of them speeds the learner’s adoption of the principles of object oriented software technology. It is undeniable that the learning of patterns improves the programmers’ development vocabulary.

Software design patterns also help in finding appropriate objects, in determining the apropos object granularity and in designing a software system that is architected from the outset to better adapt to change. At the design level, patterns enable large-scale reuse of software architectures by capturing the expert knowledge of pattern based development and distributing it throughout the development team.

It is generally acknowledged that these are the two most important benefits: the way in which they form a vocabulary for articulating design decisions during the normal course of development conversations among promgrammers. This can also come into play during the close programming work of so-called “[pair programming](http://en.wikipedia.org/wiki/Pair_programming)“, among those who have found it to be useful for them.

When you are working with a group of programmers who are either working in pairs or as part of a group using pattern-based development, you frequently hear talk like “I think we need a strategy here”, or, from one programmer to the rest of the group, “Let’s implement this functionality as an Observer”.

Programmers’ familiarity with pattern-based development has also become a kind of hiring shorthand. Whenever a talented programmer leaves a software development team I am leading, and we need to replace him or her with anther programmer, I use the “Do we need a programmer familiar with design patterns” question as a line of demarcation for recruiting and hiring decisions. The answer is \*not\* always to hire an expensive programmer intimately familiar with design patterns, either.

It is fashionable in development manager circles to use design patterns as a hiring demarcation line as well, as in the following exchange:

“So…regarding design patterns: what would you say is your favorite design pattern?”

“Well, the factory, I guess.”

“Yeah…OK…thanks for coming down.”

#### What does a software development pattern look like?

A pattern is a problem-solution pair that can be applied in a similar fashion in new contexts; the pattern is complete with advice on how to apply it in the new context. It is important to note that the formal definition of a pattern is not consistent in the literature.

There are three types of patterns:

1. An architectural pattern occurs across software subsystems.

2. A design pattern occurs within a subsystem but is independent of the language.

3. An idiom is a low-level pattern that is programming language-specific.

Each individual pattern is compromised of four elements:

1. A **name**. Some of the names of the [software design patterns](http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29) can be rather whimsical: “flyweight”, and “singleton”. The whimsy is to serve the purpose of making the patterns memorable to programmers.

2. A **problem description**. The problem part of the pattern describes the problem and its context, as well as specific design issues such as how to represent algorithms as objects. The problem statement may also speak about when it is best to apply this particular pattern and may also describe class structures that are symptoms of an inflexible software design.

3. A **solution** to the problem. The solution part of the design pattern does not desibe any one particular concrete design or implementation, but only describes the elements that make up the design, The solution only provides a general arrangement of objects and classes which can be used to solve this type of problem.

4. The **consequences** of the solution. This part of the design pattern describes the results and inherent risks and trade-offs associated with applying this particular design pattern. It may include the impact of this design pattern on space and time, programming language and implementation issues, or include notes on software flexibility, system extensibility, and portability. These consequences are critical for evaluating alternative software design patterns.

#### What is the history of software design patterns?

The concept of design patterns was first introduced by [Christopher Alexander](http://en.wikipedia.org/wiki/Christopher_Alexander) for use in architecture and town planning. He realized that architects encounterd the same sorts of problems when engaged in the design of buildings and once an elegant architectural solution to these common problems was discovered, it could be repeated over and over again. In 1977, he wrote a book, published by the Oxford University Press, called “[A Pattern Language”](http://www.amazon.com/Pattern-Language-Buildings-Construction-Environmental/dp/0195019199/ref%3Dsr_1_1?ie=UTF8&qid=1310881288&sr=8-1), in which he stated:

**6. Define cohesion and coupling. Explain the various types in each of them.**

**Coupling**: Two modules are considered independent if one can function completely without the presence of other. Obviously, if two modules are independent, they are solvable and modifiable separately. However, all the modules in a system cannot be independent of each other, as they must interact so that together they produce the desired external behavior of the system.

The more connections between modules, the more dependent they are in the sense that more knowledge about one module is required to understand or solve the other module. Hence, the fewer and simpler the connections between modules, the easier it is to understand one without understanding the other. Coupling between modules is the strength of interconnection between modules or a measure of independence among modules.

To solve and modify a module separately, we would like the module to be loosely coupled with other modules. The choice of modules decides the coupling between modules. Coupling is an abstract concept and is not easily quantifiable. So, no formulas can be given to determine the coupling between two modules. However, some major factors can be identified as influencing coupling between modules.

Among them the most important are the type of connection between modules, the complexity of the interface, and the type of [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information%22%20%5Co%20%22information%22%20%5Ct%20%22_self) flow between modules. Coupling increase with the complexity and obscurity of the interface between modules. To keep coupling low we would like to minimize the number of interfaces per module and the complexity of each interface. An interface of a module is used to pass information to and from other modules. Complexity of the interface is another factor affecting coupling.

The more complex each interface is, higher will be the degree of coupling. The type of information flow along the interfaces is the third major factor-affecting coupling. There are two kinds of information that can flow along an interface: data or control, Passing or receiving control information means that the action of the module will depend on this control information, which makes it more difficult to understand the module and provide its abstraction. Transfer of data information means that a module passes as input some data to another module and gets in return some data as output.

Cohesion: Cohesion is the concept that tries to capture this intra-module. With cohesion we are interested in determining how closely the elements of a module are related to each other. Cohesion of a module represents how tightly bound the internal elements of the module are to one another. Cohesion of a module gives the designer an idea about whether the different elements of a module belong together in the same module.  Cohesion and coupling are clearly related. Usually the greater the cohesion of each module in the system, the lower the coupling between modules is. There are several levels of Cohesion:

-              Coincidental

-              Logical

-              Temporal

-              Procedural

-              Communicational

-              Sequential

-              Functional

Coincidental is the lowest level, and functional is the highest. Coincidental Cohesion occurs when there is no meaningful relationship among the elements of a module. Coincidental Cohesion can occur if an existing program is modularized by chopping it into pieces and making different pieces modules.

A module has logical cohesion if there is some logical relationship between the elements of a module, and the elements perform functions that fill in the same logical    class. A typical example of this kind of cohesion is a module that performs all the inputs or all the outputs. Temporal cohesion is the same as logical cohesion, except that the elements are also related in time and are executed together. Modules that perform activities like “initialization”, “clean-up” and “termination” are usually temporally bound.

A procedurally cohesive module contains elements that belong to a common procedural unit. For example, a loop or a sequence of decision statements in a module may be combined to form a separate module. A module with communicational cohesion has elements that are related by a reference to the same input or output data. That is, in a communicationally bound module, the elements are together because they operate on the same input or output data.

When the elements are together in a module because the output of one forms the input to another, we get sequential cohesion. Functional cohesion is the strongest cohesion. In a functionally bound module, all the elements of the module are related to performing a single function. By function, we do not mean simply mathematical functions; modules accomplishing a single goal are also included.

**7. Elaborate the rules for good use interface design.**

***(Nov/Dec- 2009 R-2007)***

**8. Define Cohesion and Coupling. Explain the various types in each of them.**

**9. Elaborate the rules for good user interface.**

**10. Write short note on User interface Analysis and Design.**

***(Nov/Dec- 2012 R-2009/2010)***

**11. Describe the various concepts and notations used in Software Design.**

**12. Describe the fundamental software design concepts.**

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**16 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - III**

***(May-June 2005)***

**1. Who should perform the validation test the software developer or the software user? Justify your answer.**

Validation Testing

* Validation Test Criteria
* Configuration Review
* Alpha and Beta Testing

**2. Explain software testing principles in detail.**

* All tests should be traceable to customer requirements
* Tests should be planned long before testing begins
* The Pareto principle applies to software testing
* Testing should begin “in the small” and progress toward testing “inthe large
* Exhaustive testing is not possible.
* To be most effective, testing should be conducted by an independentthird party

**3. With suitable example explain the Basis Path Testing in detail.**

The basis path method enables the test case designer to derive a logical complexitymeasure of a procedural design and use this measure as a guide for defining abasis set of execution paths. Test cases derived to exercise the basis set are guaranteedto execute every statement in the program at least one time during testing.

* Flow Graph Notation
* Cyclomatic Complexity
* Deriving Test Cases
* Graph Matrices

***(Nov/Dec- 2007 R-2002)***

**4. What is cyclomatic complexity? Why it is important in basis path testing? Explain with an example. Refer Q.No:3**

**5. Explain the various types of software maintenance in detail. Discuss how can you enhance maintainability during development.**

The maintenance of existing software can account for over 60 percent of all effortexpended by a development organization, and the percentage continues to rise asmore software is produced

***(Nov/Dec- 2007 R-2005)***

**6. Explain the various system testing methods**

System Testing

* Recovery Testing
* Security Testing
* Stress Testing
* Performance Testing

**7. Write notes on testing of real time environments.**

**8. Explain the variations in testing for OO environment**

**9. Assume a problem and device the test cases for it using the method equivalence partitioning**

***(Nov/Dec- 2009 R-2007)***

**10. Explain the various System testing methods. Refer Q.No:6**

**12. Explain Unit testing and debugging guidelines.**

Unit Testing

* Unit Test Considerations
* Unit Test Procedures

**13. Define White-box testing.Explain two testing approaches involved in it.**

White-box testing, sometimes called glass-box testing, is a test case design methodthat uses the control structure of the procedural design to derive test cases. Usingwhite-box testing methods, the software engineer can derive test cases that (1) guaranteethat all independent paths within a module have been exercised at least once,(2) exercise all logical decisions on their true and false sides, (3) execute all loops attheir boundaries and within their operational bounds, and (4) exercise internal datastructures to ensure their validity.

***(Nov/Dec- 2012 R-2009/2010)***

**14. Explain the various Black Box testing techniques.**

Black-Box Testing

* Graph-Based Testing Methods
* Equivalence Partitioning
* Boundary Value Analysis
* Comparison Testing
* Orthogonal Array Testing

**15. Describe the various types of activities done in maintenance of Software.**

Much of the software we depend on today is on average 10 to 15 years old. Even whenthese programs were created using the best design and coding techniques known at thetime [and most were not], they were created when program size and storage space wereprinciple concerns. They were then migrated to new platforms, adjusted for changes inmachine and operating system technology and enhanced to meet new user needs—all withoutenough regard to overall architecture.

***(Aug- 2012 R-2007)***

**16. Explain the issues in a successful software testing strategy.**

**17.Describe in detail the basis path testing. Refer Q.No:3**

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**16 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - IV**

***(May-June 2005)***

**1. Describe the process and product metrics in detail.**

Measures, Metrics, and Indicators

Metrics in the Process and Project Domains

* Process Metrics and Software Process Improvement
* Project Metrics

Software Measurement

* Size-Oriented Metrics
* Function-Oriented Metrics
* Extended Function Point Metrics

Reconciling Different Metrics Approaches

Metrics for Software Quality

* An Overview of Factors That Affect Quality
* Measuring Quality
* Defect Removal Efficiency

Integrating Metrics Within the Software Engineering Process

* Arguments for Software Metrics
* Establishing a Baseline
* Metrics Collection, Computation, and Evaluation

 Managing Variation: Statistical Quality Control

 Metrics for Small Organizations

 Establishing a Software Metrics Program

**2. What are all the quality criteria used in software development? What are the direct and indirect measures that can be used on th quality criteria to assess the quality?**

***(Nov/Dec- 2007 R-2002)***

**3. Why quality assurance is an important phase in software development? Discuss in detail various quality assurance standards.**

Quality assurance consists of the auditing and reporting functions of management.The goal of quality assurance is to provide management with the data necessary tobe informed about product quality, thereby gaining insight and confidence that productquality is meeting its goals. Of course, if the data provided through quality assuranceidentify problems, it is management’s responsibility to address the problemsand apply the necessary resources to resolve quality issues.

The ISO 9000 Quality Standards

* The ISO Approach to Quality Assurance Systems 217
* The ISO 9001 Standard

**4. Discuss in detail the various quality attributes with suitable illustrations.**

The standard identifies six key quality attributes:

**Functionality.** The degree to which the software satisfies stated needs as indicatedby the following subattributes: suitability, accuracy, interoperability,compliance, and security.

**Reliability.** The amount of time that the software is available for use as indicatedby the following subattributes: maturity, fault tolerance, recoverability.

**Usability.** The degree to which the software is easy to use as indicated by thefollowing subattributes: understandability, learnability, operability.

**Efficiency.** The degree to which the software makes optimal use of systemresources as indicated by the following subattributes: time behavior, resourcebehavior.

**Maintainability**. The ease with which repair may be made to the software asindicated by the following subattributes: analyzability, changeability, stability,testability.

**Portability.** The ease with which the software can be transposed from oneenvironment to another as indicated by the following subattributes: adaptability,installability, conformance, replaceability.

***(Nov/Dec- 2007 R-2005)***

**5. Write notes on statistical software quality assurance .**

Software Quality Assurance

* Background Issues
* SQA Activities

**6. Write notes on software safety.**

Software safety is a software quality assurance activity that focuses on the identificationand assessment of potential hazards that may affect software negatively andcause an entire system to fail. If hazards can be identified early in the software engineeringprocess, software design features can be specified that will either eliminateor control potential hazards.

For example, some of thehazards associated with a computer-based cruise control for an automobile might be

• causes uncontrolled acceleration that cannot be stopped

• does not respond to depression of brake pedal (by turning off)

* does not engage when switch is activated

• slowly loses or gains speed

***(Nov/Dec- 2012 R-2009/2010)***

**7. Explain quality and product metrics in detail.**

* Metrics for Software Quality
* An Overview of Factors That Affect Quality
* Measuring Quality
* Defect Removal Efficiency

**E.G.S.PILLAY ENGINEERING COLLEGE, NAGAPATTINAM**

**DEPARTMENT OF MCA**

**MC7303-SOFTWARE ENGINEERING**

**16 MARKS IMPORTANT QUESTIONS & ANSWERS**

**UNIT - V**

***(May-June 2005)***

**1. What is the difference between an SCM audit and a formal technical review? Can their function be folded into one review? What are the pros and cons?**

* Software Configuration Management
* Baselines
* Software Configuration Items
* The SCM Process
* Identification of Objects in the Software Configuration
* Version Control
* Change Control
* Configuration Audit
* Status Reporting
* SCM Standards

**2. How do you identify the objects in software configuration by using various methods?**

* Identification of Objects in the Software Configuration

**3. With suitable example describe the various software configuration management tasks in detail.Refer Q.No:1**

***(Nov/Dec- 2007 R-2002)***

**4. Describe the various software configuration management takes in details.Refer Q.No:1**

**5. Write short notes on:**

 i. System documentation

* Requirements documents
* External/internal designs
* User manuals

 ii. Reusability

**Reusability:** Extent to which a program [or parts of a program] can be reused in otherapplications—related to the packaging and scope of the functions that the programperforms.*.*

 iii. CASE tools

Computer-aided softwareengineering (CASE) toolsassist software engineering managersand practitioners in every activity associatedwith the software process. They automateproject management activities, manage all workproducts produced throughout the process, andassist engineers in their analysis, design, codingand test work. CASE tools can be integrated withina sophisticated environment.

 iv. Test cases.

The design of tests for software and other engineered products can be as challengingas the initial design of the product itself. Yet, for reasons that we have alreadydiscussed, software engineers often treat testing as an afterthought, developing testcases that may "feel right" but have little assurance of being complete. Recalling theobjectives of testing, we must design tests that have the highest likelihood of findingthe most errors with a minimum amount of time and effort.

***(Nov/Dec- 2007 R-2005)***

**6. Explain in detail about the software configuration management layered activities.**

**Refer Q.No:1**

***(Nov/Dec- 2012 R-2009/2010)***

**7. Elaborate on the series of tasks of a software configuration management process.**

Software configuration management is an important element of software qualityassurance. Its primary responsibility is the control of change. However, SCM is alsoresponsible for the identification of individual SCIs and various versions of the software,the auditing of the software configuration to ensure that it has been properlydeveloped, and the reporting of all changes applied to the configuration.

**8. Discuss the concept , role, functions, features of SCM repository.**

***(Aug- 2012 R-2007)***

**9. Explain the layers of the software configuration management process. Refer Q.No:7**

**10. Discuss the activities of software quality assurance.**

* Prepares an SQA plan for a project
* Participates in the development of the project’s software process description.
* Reviews software engineering activities to verify compliance with the definedsoftware process
* Audits designated software work products to verify compliance with thosedefined as part of the software process
* Ensures that deviations in software work and work products are documentedand handled according to a documented procedure
* Records any noncompliance and reports to senior management

**11. Write about ISO 9001:2000 quality standard.**

The ISO 9000 Quality Standards

* The ISO Approach to Quality Assurance Systems
* The ISO 9001 Standard