|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **UNITED COLLEGE OF ENGINEERING & RESEARCH, PRAYAGRAJ (010)** | | | | | **Department of CS/IT** | | | |
| Third Sessional Examination (Odd Semester 2021-22) | | | | | SEMESTER: 6th | | | Date: 14/05/2022 |
| TIME: 2 hours. | | | | SUBJECT : Software Engineering | Paper code: KCS-601 | | | MM. 30 |
| **READ ALL INSTRUCTIONS AND QUESTIONS VERY CAREFULLY** | | | | | | | | |
| **SECTION A (Attempt ALL questions) Very short answer** | | | | | | **[9]** | **CO** | **Bloom’s Taxonomy Level** |
| 1 | A | | Identify the need for Software Development Life Cycle Model.  Ans: Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner. Without software life cycle model the entry and exit criteria for a phase cannot be recognized. Without software life cycle model it becomes difficult for software project managers to monitor the progress of the project. So we need software development life cycle to overcome these problems. | | | [1] | 1 | Understand(L2) |
| 1 | B | | State the limitations of Spiral Model?  Ans: Can be a costly model to use. • Risk analysis is important phase, hence requires expert people. • Project’s success is highly dependent on the risk analysis phase. Management is more complex. • Not suitable for small or low risk projects and could be expensive for small projects. • Spiral may go indefinitely (infinitely). | | | [1] | 1 | Remember (L1) |
| 1 | C | | Explain the causes of software crisis.  Ans: Software crisis is a term used in early days of computing science for difficulty of writing useful and efficient computer programs in the required time. Causes of software crisis  1. Project running over budget 2. Project running over time  3. Software was very inefficient 4. Software was of low quality 5. Software often did not meet requirement  6. Project were unmanageable and code difficult to maintain 7. Software was never delivered | | | [1] | 1 | Understand(L2) |
| 1 | D | | What is the use of feasibility study?  Ans: Feasibility studyis not used to solve the problem, but to determine whether the problem is worth solving. | | | [1] | 2 | Remember (L1) |
| 1 | E | | Compare between facilitated application specification technique (FAST) and brainstorming methods.  Ans: Brainstorming is a group technique without a manager of facilitator but FAST is a technique for requirement elicitation in which there is a facilitator assigned to establish rules for group discussion. | | | [1] | 2 | Analyze (L4) |
| 1 | F | | Define SRS and BRS.  Ans: SRS is created by the System architect whereas BRS software is usually created by the business analyst. SRS stands for System Requirement Specification whereas BRS stands for Business Requirement Specification. | | | [1] | 2 | Remember (L1) |
| 1 | G | | Define software maintenance.  Ans: Software maintenance is the last stage of software life cycle . After the product has been released, the maintenance phase keeps the s/w up to date with environment changes & changing user requirements. | | | [1] | 5 | Remember (L1) |
| 1 | H | | What is the use of CASE Tools?  Ans: 1. To increase productivity.  2. To help produce better quality software at low maintenance cost.  3. Reduced Lifetime Maintenance.  4. Improved Accuracy. | | | [1] | 5 | Remember (L1) |
| 1 | I | | State software risk analysis and management.  Ans: Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. A risk is a potential problem – it might happen, it might not. | | | [1] | 5 | Remember (L1) |
| **SECTION B (Attempt All questions) Short answer** | | | | | | **[11]** |  |  |
| 2 |  | 1. Discuss the pros and cons of using top-down approach for software designing.   Ans: Advantages-   * Your organization realizes a focused use of resources from the individual managed application. * The first implementation becomes a showcase for the identity management solution. * When the phases are completed for the managed application, you have implemented a deeper, more mature implementation of the identity management solution. * Operation and maintenance resources are not initially impacted as severely as with the bottom-up approach.   Disvantages-   * The solution provides limited coverage in the first phases. * A minimal percentage of user accounts are managed in the first phases. * You might have to develop custom adapters at an early stage. * The support and overall business will not realize the benefit of the solution as rapidly. * The implementation cost is likely to be higher.   Or   1. Classify the different software designing levels with diagram.   Ans: The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels of phases of design:   1. Interface Design 2. Architectural Design 3. Detailed Design   Lightbox | | | | [3] | 3 | Understand(L2) |
| 3 |  | (i) Demonstrate the system testing in detail.  Ans: System testing is defined as testing of a complete and fully integrated software product. This testing falls in black-box testing wherein knowledge of the inner design of the code is not a pre-requisite and is done by the testing team.  System testing is performed in the context of a System Requirement Specification (SRS) and/or a Functional Requirement Specifications (FRS). It is the final test to verify that the product to be delivered meets the specifications mentioned in the requirement document. It should investigate both functional and non-functional requirements.  There are various types of system testing and the team should choose which ones they would need before application deployment.   * 1. Alpha Testing   2. Beta Testing   3. Acceptance Testing   4. Non-Functional Testing     Or   1. Demonstrate stubs and drivers with example.   Ans:  **Stubs :**  Stubs are developed by software developers to use them in place of modules, if the respective modules aren’t developed, missing in developing stage, or are unavailable currently while Top-down testing of modules. A Stub simulates module which has all the capabilities of the unavailable module. Stubs are used when the lower-level modules are needed but are unavailable currently.  **Drivers :**  Drivers serve the same purpose as stubs, but drivers are used in Bottom-up integration testing and are also more complex than stubs. Drivers are also used when some modules are missing and unavailable at time of testing of a specific module because of some unavoidable reasons, to act in absence of required module. Drivers are used when high-level modules are missing and can also be used when lower-level modules are missing.  **Ex :** Suppose, you are told to test a website whose corresponding primary modules are, where each of them is interdependent on each other, as follows:   * **Module-A :** Login page website, * **Module-B :** Home page of the website * **Module-C :** Profile setting * **Module-D :** Sign-out page   It’s always considered good practice to begin development of all modules parallelly because as soon as each gets developed they can be integrated and could be tested further as per their corresponding interdependencies order with a module. But in some cases, if any one of them is in developing stage or not available in the testing process of a specific module, stubs or drivers could be used instead.  Assume Module-A is developed. As soon as it’s developed, it undergoes testing, but it requires Module-B, which isn’t developed yet. So in this case, we can use the Stubs or Drivers that simulate all features and functionality that might be shown by actual Module-B. So, we can conclude that Stubs and drivers are used to fulfill the necessity of unavailable modules. Similarly, we may also use Stubs or Drivers in place of Module-C and Module-D if they are too not available. | | | | [3] | 4 | Apply (L3) |
| 4 |  | (i) State the difference between corrective and preventive maintenance.  Ans: The main characteristic is that corrective maintenance does not require planning. However, this lack of preparation is not always positive or productive.  On the other hand, it is important to bear in mind that corrective maintenance implies a greater stoppage of production and can affect the company’s normal rhythm. Therefore, it should be noted that if a company only carries out preventive maintenance, it will suffer a greater number of stoppages, increased costs and even delays in deliveries and orders.  This type of maintenance is usually carried out by small companies with very little workload. However, no matter how limited the company is, this type of maintenance is not advisable if you are looking for long-term stability and optimization of all resources to the maximum.  The alternative to the classic corrective maintenance is preventive maintenance or, also called, planned maintenance. This type of system is carried out on a regular basis in order to minimise the number of failures in the production line. Basically, it aims to prevent failures due to wear and tear by anticipating possible breakdowns due to deterioration.  Or  (ii) Differentiate between reverse, forward and reengineering.  Ans:  https://media.geeksforgeeks.org/wp-content/uploads/20190516125251/Untitled-Diagram-125.png  https://media.geeksforgeeks.org/wp-content/uploads/20190516125343/Untitled-Diagram-214.png  Forward Engineering is a method of creating or making an application with the help of the given requirements. Forward engineering is also known as Renovation and Reclamation. Forward engineering is required high proficiency skills. It takes more time to construct or develop an application. Forward engineering is a technique of creating high-level models or designs to make in complexities and low-level information.  Reverse Engineering is also known as backward engineering, is the process of forward engineering in reverse. In this, the information is collected from the given or existing application. It takes less time than forward engineering to develop an application. In reverse engineering, the application is broken to extract knowledge or its architecture.  Reverse Engineering It performs transformation from a large abstraction level to a higher one. Forward Engineering It performs transformation from a higher abstraction level to a lower one. Reengineering It transforms an existing s/w system into a new but more maintainable system. | | | | [5] | 5 | Analyze (L4) |
| **SECTION C (Attempt ANY ONE question) Long answer** | | | | | | **[10]** |  |  |
| 6 |  | | Demonstrate the COCOMO model. Also explain different modes of software development by using COCOMO model.  Ans: It is a hierarchy of software costs estimation models, which include basic, intermediate & detailed sub models.  **In COCOMO, projects are categorized into three types:**   1. Organic 2. Semidetached 3. Embedded   **1.Organic:** A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**  **2. Semidetached:** A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**  **3. Embedded:** A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.  According to Boehm, software cost estimation should be done through three stages:   1. Basic Model 2. Intermediate Model 3. Detailed Model   **1. Basic COCOMO Model:** The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:  **Effort=a1\*(KLOC) a2 PM**                 **Tdev=b1\*(efforts)b2 Months**  Where  **KLOC** is the estimated size of the software product indicate in Kilo Lines of Code,  a1,a2,b1,b2 are constants for each group of software products,  **Tdev** is the estimated time to develop the software, expressed in months,  **Effort** is the total effort required to develop the software product, expressed in **person months (PMs)**.  **Estimation of development effort**  For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:  **Organic:** Effort = 2.4(KLOC) 1.05 PM  **Semi-detached:** Effort = 3.0(KLOC) 1.12 PM  **Embedded:** Effort = 3.6(KLOC) 1.20 PM  **Estimation of development time**  For the three classes of software products, the formulas for estimating the development time based on the effort are given below:  **Organic:** Tdev = 2.5(Effort) 0.38 Months  **Semi-detached:** Tdev = 2.5(Effort) 0.35 Months  **Embedded:** Tdev = 2.5(Effort) 0.32 Months  Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size. | | | [10] | 5 | Apply (L3) & Understand(L2) |
| 7 |  | | Suppose that a project was estimated to be 350 KLOC. Calculate the effort, development time, average staff size and productivity of the project for each of the three modes define in basic COCOMO model.  Solution:   * 1. Organic Mode:   Effort = 1125.85 PM  Development Time = 36.099 PM  Average Staff Size = 31.187 Persons  Productivity = KLOC/Effort= 350/1125.85= 0.31 KLOC per PM   * 1. Semidetached Mode:   Effort = 2120.69 PM  Development Time = 36.49 PM  Average Staff Size = 58.11 Persons  Productivity = KLOC/Effort= 350/2120.69= 0.165 KLOC per PM   * 1. Embedded Mode:   Effort = 4066.157 PM  Development Time = 35.71 PM  Average Staff Size = 113.86 Persons  Productivity = KLOC/Effort= 350/4066.157= 0.086 KLOC per PM | | | [10] | 5 | Apply (L3) |
| **#### END OF PAPER ####** | | | | | | |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Course Outcome Wise Marks Distribution** | CO1 | CO2 | CO3 | CO4 | CO5 |
| 3 | 3 | 3 | 3 | 28 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Bloom’s Taxonomy Wise Marks Distribution** | L1 | L2 | L3 | L4 | L5 | L6 |
| 3 | 6 | 26 | 5 | 0 | 0 |