# 

**Iqra National University Peshawar Pakistan**

**Department of Computer Science**

Spring Semester, Final Term Exam, June 2020

|  |  |  |  |
| --- | --- | --- | --- |
| Paper : | **Topics in Software Engineering** | Date and Starting Time: | **25/June/2020, 9:00 am** |
| Program: | **MS (CS & SE)** | Uploading Date and End Time: | **25/June/2020, 3:00 pm** |
| Teacher Name: | **Dr. Fazal-e-Malik** | Marks | **50** |

**Name: Muhammad Suliman**

**ID: 15587**

**Note: Attempt all Questions. Help can be taken from net where ever is required.**

|  |  |  |  |
| --- | --- | --- | --- |
| Q.1 |  | Do we need prototyping? Explain the risks in prototyping.  Ans:  Prototyping:  Prototyping is the process to get quickly a working model (a prototype) in order to test various aspects of a design, illustrate ideas or features, and gather early user feedback.  A prototype is an essential part of the process of creating web software or a mobile app. During the prototyping process developers, designers, and end users can quickly review the user flow and design, and give important feedback on how to improve it. When you prototype, you get the chance to test your product way before you actually start working on it, and address any key pain points in advance.  Prototyping is one of the five stages of design thinking a methodology for creating solutions to complex problems using creative techniques. In simple words, it means building one or several scaled-down versions of your product or a specific feature. It might be as simple as pen on paper sketches and mockups, or as complex as a simplified coded version of a functioning app.  Prototyping helps you test your ideas early on, and make changes before you and your team have done a lot of expensive work. Building a functional prototype lets you work with your users or your client before you launch the final product. It also allows you to get valuable feedback, with enough time and budget to improve it.  If you’re worried you don’t have the time to prototype, remember that a prototype doesn’t have to involve coding at all. It can be a simple mockup done by your designer or UX expert, basing on which you can draw up the interactions and plan your work.  Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development and is comparable to prototyping as known from other fields, such as mechanical engineering or manufacturing.  A prototype is a test or preliminary model of an idea, design, process, interface, technology, product, service or creative work. A prototype that is close to the end result in functionality.  For example, a user interface that works with test data but isn't properly developed as a well-designed and integrated system.  IEEE defines prototyping as “A type of development in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process.”  **Why Prototyping is needed?**  A working prototype of the system is needed before carrying out the development of the actual software  It is a toy implementation of the system.  A prototype shows a very basic version of the actual system ,  Limited functional capabilities, low reliability, and inefficient performance as compared to the actual software.  It gives a general view of software product to customer regarding the input to the system, the processing needs and the output requirements.  The prototyping model is applied when detailed information related to input and output requirements of the system is not available.  This model allows the users to interact and experiment with a working model of the system known as prototype.  The prototype gives the user an actual feel of the system.  At any stage, if the user is not satisfied with the prototype, it can be discarded and an entirely new system can be developed.  This prototype is developed based on the currently known requirements.  Prototype’s development goes through design, coding, and testing.  The interactions with the prototype customer will better understand the requirements of the desired system.  It is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determine the requirements.  Risks are reduced.  Its development starts when the preliminary version of the requirements specification document has been developed.  It gives reasonable understanding of the system and its needs are unclear or likely to change.  It gives an opportunity to the end users and clients to use the prototype.  Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process. Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.  Prototyping will, most likely, make the development process longer. At least at the beginning. However, the initial time you spend on this task can actually save you more time and money later on. Let’s have a closer look at other benefits of prototyping:  Not only can it save your company a fortune by helping you avoid problems you could anticipate at this stage, but it can also give you the opportunity to research user needs better, and prevent some of your work from going down the drain.  There is a multitude of prototyping tools available, and many of them are free and easy to use – such as pen and paper. This makes prototyping cheap, easy and fast.  **Risks in prototyping:**  Though prototyping decreases the probability of a software development project failure, this activity has its own risks. The biggest risk is that anyone who is interested in the project after facing a working prototype will decide that the final product is almost ready.  Leads to implementing and then repairing way of building systems.  Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.  Incomplete application may cause application not to be used as the full system was designed Incomplete or inadequate problem analysis.  The flexibility of prototypes creates a good ground for “inventing” new requirements, so the BA needs to apply prioritization and verification processes to all new requirements after each prototype discussion session.  Prototype discussions can easily get sidetracked and turn into a discussion of various implementation details. It’s important to keep in mind that prototypes are meant for working out user interface requirements.  Another risk is that users may expect to have the same functionality in the final solution as what’s been indicated by the prototype. Manage the users’ expectations as there are assumptions and constraints which surface between the development of the prototype and the final solution.  Finally, it’s possible to get carried away and spend too much time on prototypes. Remember that information should be presented at the level of detail appropriate to the type of audience. The prototype should carry only a minimum of information which is sufficient to clearly communicate business context, solution scope and specific requirements. | (10 Marks) |
|  |  |  |  |
| Q.2 |  | Explain the distribution of maintenance efforts in terms of :   1. Fault Repair 2. Software Adaptation 3. Functionality Addition or Modification   Ans:   1. **Fault Repair:**   Faults are an important concept in the study of system dependability, and most approaches to dependability can be characterized by the way in which they deal with faults (e.g. fault avoidance, fault removal, fault tolerance, fault forecasting, etc.). In their seminal work on modeling dependable computing, Laprie et al. define a fault as the adjudged or hypothesized cause of an error. In this paper, we propose a more formal definition of a fault in the context of software products, and discuss the diverse implications of our definition.  In developing a software system, we would like to estimate the way in which the fault content changes during its development, as well determine the locations having the highest concentration of faults. In the phases prior to test, however, there may be very little direct information regarding the number and location of faults. This lack of direct information requires developing a fault surrogate from which the number of faults and their location can be estimated. We develop a fault surrogate based on changes in the fault index, a synthetic measure which has been successfully used as a fault surrogate in previous work. We show that changes in the fault index can be used to estimate the rates at which faults are inserted into a system between successive revisions. We can then continuously monitor the total number of faults inserted into a system, the residual fault content, and identify those portions of a system requiring the application of additional fault detection and removal resources.  **Fault Repair** means a service consisting of such repair, maintenance, adjustment or replacement of any part of the Communications Provider’s Electronic Communications Network, or such repair or adjustment of any connected or connectable network, or such repair or replacement for any Apparatus for which the Communication Provider has undertaken the responsibility for repair and maintenance, as is necessary to restore and maintain a sufficient service.   1. **Software Adaptation**   The term “adaptation” in computer science refers to a process  where an interactive system (adaptive system)  adapts its behavior to individual users based on information acquired about its user(s) and its environment.  A software system passes through a potentially long software engineering cycle and before delivery, requirement engineers and software developers realize the components of the system. However, it is impossible to anticipate the requirements of all users, and a single best or optimal system configuration is impossible. The active involvement of users and clear understanding of user and task requirements is a challenge in the development of computer-based interactive systems for two reasons  1: The potential user groups may not be known at the start of the project, and would need to be identified according to future scenarios of how the software system will be used. These groups need to be revised as the system design evolves because there may be various groups of potentially affected users.  2: The design of the project may include substantial changes compared to the users’ current experience of a system; therefore, users may not be confident and precise about their needs concerning this future system.  The use of adaptor-specific computational entities guaranteeing that software components will interact in the right way not only at the signature level, but also at the behavioral, semantic and service levels.   1. **Functionality Addition or Modification**   Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.  A common perception of maintenance is that it merely involves fixing defects. However, one study indicated that over 80% of maintenance effort is used for non-corrective actions. This perception is perpetuated by users submitting problem reports that in reality are functionality enhancements to the system.  Functionality addition means to add another or new some functionality in existing software. It can be done by get feedback from the user, client or any stockholder or it can be done after the testing done by the developer.  The process of functional testing involves a series of tests: Smoke, Sanity, Integration, Regression, Interface, System and finally User Acceptance Testing. Tests are conducted on each feature of the software to determine its behavior, using a combination of inputs simulating normal operating conditions, and deliberate anomalies and errors.  Thus after rigorous functional testing, you receive software with the consistent user interface, proper integration with business processes, well-designed API having robust security and network features.  Modification of a software product after delivery to correct faults to improve performance or other attributes  The performance of the activities required to keep a system operational and responsive after it is accepted and placed into production  Covers the life of a software system from the time it is installed to the time it is phased out.  Modification to code and associated documentation due to a problem or the need for improvement. | (10 Marks) |
|  |  |  |  |
| Q.3 |  | Re-usability can be done at various levels? Explain at least Three (3) levels.  Ans:  A definition of software reuse is the process of creating software systems from predefined software components.  Reusability is the use of existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation  One of the promises of object-orientation is reuse. Developing new software systems is expensive, and maintaining them is even more expensive. Reuse is therefore sensible in both business and technology perspectives.  Software reuse can cut software development time and costs. The major advantages for software reuse are to:  Increase software productivity.  Shorten software development time.  Improve software system interoperability.  Develop software with fewer people.  Move personnel more easily from project to project.  Reduce software development and maintenance costs.  Produce more standardized software.  Produce better quality software and provide a powerful competitive advantage.  **Reusability** **can be done at various levels:**  1: Application level:  Where an entire application is used as sub-system of new software  2: Componentlevel:  Where sub-system of an application is used  3: Moduleslevel”  Where functional modules are re-used.  Reusability have few more levels include:  Code reuse,  design reuse,  specification reuse  and application system reuse  1**: Code reuse:** Code reuse consider as the most common form of software reuse. This type of reuse happens in the development implementation stage of the system development process. The reusable code can be object code, data objects, source code and standard subroutines.  2: **Design reuse:**  Design reuse considers as a higher level of reuse where a design model of a software system is reused. This type of reuse is required when a system needs to be reported in an entirely different software or hardware environment.  3: **Specification reuse:**  The reuse of specifications consider as a higher level of reuse, the problems at the specification level which are arising from the inefficiencies of reusable code, operating system dependencies and programming language dependencies disappear.  **4: Application system reuse:**  Application system reuse is considered as a special case of software reuse, where the whole system is reused by implementing it through a range of different operating systems and computers.  The whole of an application system may be reused either by incorporating it without change into other systems (COTS reuse) or by developing application families. | (10 Marks) |
|  |  |  |  |
| Q.4 |  | Provision of failure-free software is the main objective of the software engineering but this failure of software is due to different reasons. Explain these different reasons of software failure.  Ans: Software failure occurs when the user perceives that the software has ceased to deliver the expected result. Hence, with respect to the specification input values. The user may need to identify the severity of the levels of failures. Such as catastrophic, critical, major or minor, depending on their impact on the systems.  Most software projects fail completely or partial because they don’t meet all their requirements. These requirements can be the cost, schedule, quality, or requirements objectives. According to many studies, failure rate of software projects ranges between 50% – 80%. There are a variety of causes for software failures but the most common are:  Because of the environmental conditions as well like a radiation burst, a strong magnetic field, electronic field or pollution could cause faults in hardware or firmware. Those faults might prevent or change the execution of software.  Failures may also arise because of human error in interacting with the software. Perhaps a wrong input value being entered or an output being misinterpreted.  Finally failures may also be caused by someone deliberately trying to cause a failure in the system.  Lack of user participation.  Changing requirements.  Unrealistic or unarticulated project goals.  Inaccurate estimates of needed resources.  Badly defined system requirements.  Poor reporting of the project’s status.  Lack of resources.  Unmanaged risks.  Poor communication among customers, developers, and users.  Use of immature technology.  Inability to handle the project’s complexity.  Sloppy development practices.  Poor Project Management.  Stakeholder politics.  Lack of Stakeholder involvement.  Commercial pressures.  **Software Acquisition Worst Practices:**  Use schedule compression to justify new technology on a time critical project.  Have the government mandate technological solutions.  Specify implementation technology in the Request for Proposal (RFP).  Expect to recover more than 10% schedule slip without a reduction in delivered functionality.  Put items out of project control on the critical path.  Plan on achieving more than 10% improvement from observed past performance.  Bury as much of the project complexity as possible in the software as opposed to the hardware.  Conduct critical system engineering tasks without software expertise.  Believe that formal reviews alone will provide an accurate picture of the project.  Expect that the productivity of a formal review is directly proportional to the number of attendees above five.  The most common software failures are caused by viruses and spyware. The computer is running slow, Internet does not work. There is an icon next to the clock that tries to scare you into buying some kind of “anti-spyware”. Internet Explorer does not work. When you try to start a program it takes more than a minute for it to appear, etc.  The other common cause of software problems is when updating or installing a new driver or a Windows update. In these cases the fix is easy. Restart the computer in safe mode. Then use Windows’ build in System Restore to undo the last installation/update. If the computer does not want to start in safe mode, Windows has to be reinstalled.  **Important Things Testers Must Take Note:**  When a tester is executing a test he/she may observe some difference in the behavior of the feature or functionality. But this not because of the failure. This may happen because of the wrong test data entered. The tester may not be aware of the feature or functionality. Or because of the bad environment. Because of these reasons incidents are reported. They are known as incident report. The condition or situation which requires further analysis or clarification is known as incident. To deal with the incidents, the programmer needs to do the analysis. Whether this incident has occurred because of the failure or not.  It is not necessary that defects or bugs introduced in the product are only by the software. To understand it further let us take an example. A bug or defect can also be introduced by a business analyst. Defects present in the specifications like requirements specification and design specifications can be detected during the reviews. When the defect or bug is caught during the review cannot result into failure. Because the software has not yet been executed.  These defects or bugs are reported, not blame the developers or any people. But to judge the quality of the product. The quality of product is of utmost importance. To gain the confidence of the customers it is very important to deliver the quality product on time. | (10 Marks) |
|  |  |  |  |
| Q.5 |  | The study of software reliability can be categorized into three (3) parts: modeling, measurement and improvement. Explain them in terms of:   1. Software Reliability Models 2. Software Reliability Metrics 3. Software Reliability Improvement   Ans:   1. **Software Reliability Models:**   Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment.  It can be an important factor affecting system reliability.  Different from hardware reliability since it doesn’t age, wear out, rust, deform or crack.  Software usually stays in the same condition as when it was created and unless there are changes caused by hardware - like changes in the storage content or data path.  It could be assumed that software doesn’t really “break”.  A software reliability model indicates the form of a random process that defines the behavior of software failures to time.  Software reliability models have appeared as people try to understand the features of how and why software fails, and attempt to quantify software reliability.  Over 200 models have been established since the early 1970s, but how to quantify software reliability remains mostly unsolved.  There is no individual model that can be used in all situations. No model is complete or even representative.  Most software models contain the following parts:   1. Assumptions 2. Factors   A mathematical function that includes the reliability with the elements. The mathematical function is generally higher-order exponential or logarithmic.  Software modeling technique can be divided into two sub categories  1:Prediction Modeling  2:Estimation Modeling  Both kinds of modeling methods are based on observing and accumulating failure data and analyzing with statistical inference.  A reliability growth model is a numerical model of software reliability, which predicts how software reliability should improve over time as errors are discovered and repaired. These models help the manager in deciding how much efforts should be devoted to testing. The objective of the project manager is to test and debug the system until the required level of reliability is reached.   1. **Software Reliability Metrics:**   Reliability metrics are used to quantitatively expressed the reliability of the software product. The option of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain.  Some reliability metrics which can be used to quantify the reliability of the software product are as follows:  1:  Mean Time to Failure (MTTF)  2: Mean Time to Repair (MTTR)  3. Mean Time Between Failure (MTBR)  4. Rate of occurrence of failure (ROCOF)  5. Probability of Failure on Demand (POFOD)  6. Availability (AVAIL)  Now let explain one by one  1:  Mean Time to Failure (MTTF):  MTTF is described as the time interval between the two successive failures. An MTTF of 200 mean that one failure can be expected each 200-time units. The time units are entirely dependent on the system & it can even be stated in the number of transactions. MTTF is consistent for systems with large transactions.  For example, It is suitable for computer-aided design systems where a designer will work on a design for several hours as well as for Word-processor systems.  To measure MTTF, we can evidence the failure data for n failures. Let the failures appear at the time instants t1, t2.....tn.  2: Mean Time to Repair (MTTR:  Once failure occurs, some-time is required to fix the error. MTTR measures the average time it takes to track the errors causing the failure and to fix them.  3. Mean Time Between Failure (MTBR):  We can merge MTTF & MTTR metrics to get the MTBF metric  Thus, an MTBF of 300 denoted that once the failure appears, the next failure is expected to appear only after 300 hours. In this method, the time measurements are real-time & not the execution time as in MTTF.  4. Rate of occurrence of failure (ROCOF):  It is the number of failures appearing in a unit time interval. The number of unexpected events over a specific time of operation. ROCOF is the frequency of occurrence with which unexpected role is likely to appear. A ROCOF of 0.02 mean that two failures are likely to occur in each 100 operational time unit steps. It is also called the failure intensity metric.  5: Probability of Failure on Demand (POFOD):  POFOD is described as the probability that the system will fail when a service is requested. It is the number of system deficiency given several systems inputs.  POFOD is the possibility that the system will fail when a service request is made.  6. Availability (AVAIL):  Availability is the probability that the system is applicable for use at a given time. It takes into account the repair time & the restart time for the system. An availability of 0.995 means that in every 1000 time units, the system is feasible to be available for 995 of these. The percentage of time that a system is applicable for use, taking into account planned and unplanned downtime. If a system is down an average of four hours out of 100 hours of operation, its AVAIL is 96%.   1. **Software Reliability Improvement:**   It can be improved by sufficient understanding of software reliability, characteristics of software & sound software design. Complete testing of the software is not possible; however sufficient testing & proper maintenance will improve software reliability to great extent.  Digital systems offer various advantages over analog systems. Their use in large-scale control systems has greatly expanded in recent years. This raises challenging issues to be resolved. Extremely high-confidence in software reliability is one issue for safety-critical systems, such as NPPs. Some issues related to software reliability are tightly coupled with software faults to evaluate software reliability (Chapter 4). There is not “one right answer” as to how to estimate software reliability. Merely measuring software reliability does not directly make software more reliable, even if there is a “proper answer” for estimation of software reliability. Software faults should be carefully handled to make software more reliable with as many reliability improvement techniques as possible. However, software reliability evaluation may not be useful. Software reliability improvement techniques dealing with the existence and manifestation of faults in software are divided into three categories:  - Fault avoidance/prevention that includes design methodologies to make software provably fault-free  - Fault removal that aims to remove faults after the development stage is completed. This is done by exhaustive and rigorous testing of the final product  - Fault tolerance that assumes a system has unavoidable and undetectable faults and aims to make provisions for the system to operate correctly, even in the presence of faults  Software Reliability is an important facet of software quality. Software reliability is the probability of the failure free operation of a computer program for a specified period of time in a specified environment. Software Reliability is dynamic and stochastic. It differs from the hardware reliability in that it reflects design perfection, rather than manufacturing perfection. This article provides an overview of Software Reliability which can be categorized into: modeling, measurement and improvement, and then examines different modeling technique and metrics for software reliability, however, there is no single model that is universal to all the situations. The article will also provide an overview of improving software reliability and then provides various ways to improve software reliability in the life cycle of software development.  In diverse industrial and academic environments, the quality of the software has been evaluated using different analytic studies. The contribution of the present work is focused on the development of a methodology in order to improve the evaluation and analysis of the reliability of web-based software applications. The Personal Software Process (PSP) was introduced in our methodology for improving the quality of the process and the product. The Evaluation + Improvement (Ei) process is performed in our methodology to evaluate and improve the quality of the software system. We tested our methodology in a web-based software system and used statistical modeling theory for the analysis and evaluation of the reliability. The behavior of the system under ideal conditions was evaluated and compared against the operation of the system executing under real conditions. The results obtained demonstrated the effectiveness and applicability of our methodology.  --------------------------------------End------------------------------------------ | (10 Marks) |

☺ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*☺

***Note: Please write your Name and ID on top of your answer Paper otherwise you will get zero marks.***