

ACHIEVER'S EDUCATION CONSULTANT
08141710097
SUMMARY OF CIT333 WITH PAST QUESTIONS

The course consists of 15 units which will enable you to develop the skills necessary for you to develop, operate and maintain software. They are no compulsory pre-requisites to it, although it is good to have a basic knowledge of operating computer.

What You will Learn in this Course

This course guide tells you briefly what the course about, what course materials you will be using and how you can work with these materials. In addition, it advocates some general guidelines for the amount of time you are likely to spend on each unit of the course in order to complete it successfully. It gives you guidance in respect of your Tutor-Marked Assignment which will be made available in the assessment available. There will be regular tutorial classes that are related to the course.

It is advisable for you to attend these tutorial sessions. The course will prepare you for challenges you will meet in the field of software engineering. You should read these objectives before you study the unit. You may wish to refer to them during your study to check to check on your progress.

By doing so, you would have followed the instruction in the unit. In addition to the aims above, this course sets to achieve some objectives.

Explain who a software engineer is

Explain software development life cycle model. Explain programming environment.

Explain Implementation and Testing

Each unit contains self-assessment exercises and at certain point in the course, you will be required to assignments for assessment purposes. The course should take you about a total of 17 weeks to complete. This course entails that you spend a lot of time to read. I would advice that you avail yourself the opportunity of attending the tutorial sessions where you have the opportunity of comparing your knowledge with that of other people.

[...]

What is Software Engineering History of Software Engineering. Software Crisis Software Development Unit 1 Overview of software development Software development life cycle model Modularity.

Pseudocode Programming Enviroment, Case Tools and Hipo Diagram

ACHIEVER'S EDUCATIONAL CONSULTANT 08141710097

Unit 1 Implementation Testing Phase Software Quality Assurance Verification and Validation
Each unit consists of one or two weeks's work and include an introduction, objectives, reading materials, conclusion, summary, Tutor Marked Assignment, references and other resources. In addition to TMAs, these exercises will help you in achieving the stated learning objectives of the individual units and of the course as a whole. Your course materials have important dates for the early and timely completion and submission of your TMAs and attending tutorials. You should remember that you are required to submit all your assignments by the stipulated time and date.

You should guard against falling behind in your work. First is made up of self-assessment exercises, second consists of the Tutor_Marked Assignment and third is the written examination/end of course examination. You are advised to do the exercises. In tackling the assignments, you are expected to apply information, knowledge and techniques you gathered during the course.

The assignments must be submitted to your facilitator for formal assessments in accordance with the deadlines stated in the presentation schedule and the assignment file. The work you submit to your tutor for assessment will count for 30% of your total course work. At the end of the course you will need to sit for a final or end of course examination of about a three hour duration. This examination will count for 70% of your total course mark.

You will be given four TMAs to answer. Three of these must be answered before you are allowed to sit for the end of course examination. The TMAs would be given to you by your facilitator and returned after you have done the assignment. Assignment questions for the units in this course are contained in the assignment file.

You will be able to complete your assignment from the information

and the material contained in your reading, references and the study units. Make sure that each assignment reaches your facilitator on or before the deadline given in the presentation schedule and assignment file. If for any reason you can not complete your work on time, contact your facilitator before the assignment is due to discuss the possibility of an extension. Extension will not be granted after the due date unless there are exceptional circumstances.

The end of your examination for Software Engineering will be for about 3 hours and it has a value of 70% of the total course work. All areas of the course will be assessed. You might find it useful to review your self-test, TMAs and comments on them before the examination.

Facilitator/Tutor and Tutorials

There are 16 hours of tutorials provided in support of the course. You will be notified of the dates, times and location of these tutorials as well as the name and phone number of your facilitator, as soon as you are allocated a tutorial group.

Your facilitator will mark and comment on your assignments, keep a close watch on your progress and any difficulties you might face and provide assistance to you

You are expected to mail your Tutor Marked Assignment to your facilitator before the schedule date. They will be marked by your tutor and returned to you as soon as possible. You can raise any problem encountered in the course of your study. You will learn a lot from participating actively in the discussions.

Upon the completion of the course, you will be equipped with the knowledge of engineering as it relates to software. You will be exposed to details relating to software requirements, design, testing and implementation. Of course a lot more questions you will be able to answer. I wish success in the course and I hope you will find it both interesting and useful.

Basic Concept of Software Engineering Unit 1

In this unit, the categories of software are examined. Differentiate between System, Application and programming Software. A program itself is a sequence of instructions which the computer follows to perform a given task. Programming software offers tools to assist a programmer in writing programs, and software using different programming languages in a more convenient way.

In this unit we have learnt that

Programming software offers tools to assist a programmer in writing programs.

[...]

Software Engineering is the application of engineering to software. Explain the principles of software engineering. Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of

software, and the study of these approaches. In other words, it is the application of engineering to software.

Sub-disciplines of Software engineering Software engineering can be divided into ten sub-disciplines. Computer-Aided Software Engineering tools and use standards for the format, such as the Unified Modeling Language . Software testing Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

Software engineering tools, CASE tools are a class of software that automates many of the activities involved in various life cycle phases. Software quality The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

Software Engineering Goals and Principles Goals

Apart from accomplishing these stated requirements, a good software system must be able to easily support changes to these requirements over the system's life. Therefore, a major goal of software engineering is to be able to deal with the effects of these changes.

Principles

The software should be organized as a ladder of abstraction in which each level of abstraction is built from lower levels. The code is sufficiently conceptual so the user need not have a great deal of technical background in the subject. The reader should be able to easily follow the logical

path of each of the various modules. The decomposition of the code should be clear.

Elements that do not affect other segment of the system are inaccessible to the user, so that only the intended operations can be performed. Logically related computational units are collected together in modules. This gives rise to a more readily alterable system, and enables the reusability of tested components.

Discuss the principles of software engineering

In the 1968, software engineering originated from the NATO Software Engineering Conference. It came at the time of software crisis. Though some people are still confused whether software engineering is actually engineering because software is more of invisible course.

Although it is disputed what impact it has had on actual software development over the last more than 40 years, the field's future looks bright according to Money Magazine and

Salary.com who rated «software engineering» as the best job in America in 2006. This problem necessitated the development. Programming languages started to appear in the 1950s and this was also another major step in abstraction. Major languages such as FORTRAN, ALGOL, and COBOL were released in the late 1950s to deal with scientific, algorithmic, and business problems respectively.

In 1967, the Simula language introduced the object-oriented programming paradigm. The technological advancement in software has always been driven by the ever changing manufacturing of various types of computer hardware. The more the new technologies upgrade, from vacuum tube to transistor, and to microprocessor were emerging, the more the necessity to upgrade and even write new software. In the mid 1980s software experts had a consensus for centralised construction of software with the use of software development Life Cycle from system analysis.

This period gave birth to object-oriented programming languages.

There are a number of areas where the evolution of software engineering is notable

The writing of software was delegated to the women. Today, many fewer women work in software engineering than in other professions, this reason for this is yet to be ascertained. Computers can now be said to be much more available and much more powerful, which has a lot of effects on software. The larger market can sustain large projects to create commercial packages, as the practice of companies such as Microsoft.

The inexpensive machines permit each programmer to have a terminal capable of fairly rapid compilation. The programs under consideration can use techniques such as garbage collection, which make them easier and faster for the programmer to write. Conversely, many fewer organizations are concerned in employing programmers for large custom software projects, instead using commercial packages as much as possible. Programmers had to rewrite all

their programs to run on these new computers.

They did not have computers on their desks and had to go to the «computer room» or “computer laboratory”. Jobs were run by booking for machine time or by operational staff. Custom software was sold by a few companies but no sale of packaged software. Organisation such as like IBM's scientific user group SHARE gave out software free and as a result reuse was order of the day.

Academia did not yet teach the principles of computer science. Modular programming and data abstraction were already being used in programming. It was the crisis that identified many of the problems of software development. Initially the software crisis was defined in terms of productivity, but advanced to emphasize quality.

It was a decade-long project from the 1960s and eventually produced one of the most complex software systems at the time.

1985 to 1989: No silver bullet

Apparently, they proclaim every new technology and practice from the 1970s to the 1990s as a silver bullet to solve the software crisis. They advocated proving all programs correct. Fred Brooks , No Silver Bullet article, argued that no individual technology or practice would ever make a 10-fold improvement in productivity within 10 years. Debate about silver bullets continued over the following decade.

Supporter for Ada, components, and processes continued arguing for years that their favorite technology would be a silver bullet. Skeptics disagreed. Eventually, almost everyone accepted that no silver bullet would ever be found. Yet, claims about silver bullets arise now and again, even today.

Others interpret no silver bullet as evidence that software engineering has finally matured and recognized that projects succeed due to hard work. However, it could also be pointed out that there are, in fact, a series of silver bullets today, including lightweight methodologies, spreadsheet calculators, customized browsers, in-site search engines, database report generators, integrated design-test coding-editors with memory/differences/undo, and specialty shops that generate niche software, such as information websites, at a fraction of the cost of totally customized website development. Nevertheless, the field of software engineering looks as if it is too difficult and different for a single «silver bullet» to improve most issues, and each issue accounts for only a small portion of all software problems.

1990 to 1999: Importance of the Internet

The birth of internet played a major role in software engineering. With its arrival, information could be gotten from the World Wide Web speedily. Programmers could handle illustrations, maps, photographs, and other images, plus simple animation, at a very fast rate. It became easier to display and retrieve information as a result of the usage of browser on the HTML language.

The widespread of network connections brought in computer viruses and worms on MS Windows computers. These new technologies brought in a lot good innovations such as e-mailing, web-based searching, e-education to mention a few. As a result, many software systems had to be re-designed for international searching. It was also required to translate the information flow in multiple foreign languages. Many software systems were designed for multi-language usage, based on design concepts from human translators.

2000 to Present: Lightweight Methodologies

There was a change from rapid-prototyping to entire lightweight methodologies. What is it today Software Engineering as a profession is now being defined as a field of human experts in boundary and content. Software Engineering is rated as one of the best job in developed economies in terms of growth, pay, and flexibility and so on.

Listed below are some renowned software engineers

Charles Bachman is particularly known for his work in the area of databases. Fred Brooks best-known for managing the development of OS/360. Peter Chen, known for the development of entity-relationship modeling. Edsger Dijkstra developed the framework for proper programming.

David Parnas developed the concept of information hiding in modular programming.

What is the situation of software Engineering today?

This unit has looked at the historical development of software engineering.

Discuss the historical development of software engineering

In unit 3 the historical development of software engineering was discussed. Prominence of the Internet, 2000 to Present, Lightweight Methodologies, Software engineering today and the prominent figures in the history of software engineering. The material in this unit will explain who a software engineer is, his tasks, technical and functional knowledge as well as occupational characteristics. It is expected of you that at the end of the unit, you will have achieved the objectives listed below.

Who is a Software Engineer?

A software engineer is an individual who applies the principles of software engineering to the design, development, testing, and evaluation of the software and systems in order to meet with client's requirements.

Software engineer occupational characteristics or features can be categorised as

Many of the occupations require working outside, and do not involve a lot of paperwork or working closely with others. These occupations can involve searching for facts and figuring out problems mentally.

Activity D

This unit has explained to you who software engineer is. You have also been informed of about his various task and occupational characteristics.

Employment and Wages, November 2004

This problem is compounded by poor marketing approaches and the fact that most of the hardware was not manufactured locally. Relatively low price of foreign Lack of quality products.

What is a software crisis?

Discuss how software crisis manifested itself in the early day of software engineering. » Go To Statement Considered Harmful«.

In the last unit, you have learnt about the software crisis- its manifestation, causes, as well as solution to the crisis. In this unit, we are going to look at the overview of software development.

You will learn specifically about the overview of various stages involved in software development. After studying this unit you are expected to have achieved the following objectives listed below. Software development may include research, new development, modification, reuse, reengineering, maintenance, or any other activities that result in software products.

Stages of Software Development

While some take a more structured, engineering-based approach, others may take a more incremental approach, where software evolves as it is developed piece-by-piece.

[...]

These stages are collectively referred to as the software development lifecycle ., These stages may be carried out in different orders, depending on approach to software development. The “waterfall” based approach attempts to assess the majority of risks and develops a detailed plan for the software before implementation begins. It avoids significant design changes and re-coding in later stages of the software development lifecycle. The choice of an approach to solving a problem using software depends on the type of problem.

What is software development?

Edward Hasted.

Unit 2:Software Development Life Cycle Model

ACHIEVER'S EDUCATIONAL CONSULTANT 08141710097

The last unit exposed you to the overview of software development. In this unit you will learn about the various lifecycle models in general. Definition of Life Cycle Model those phases are executed. The General Model General Life Cycle Model Fig 1 the General Model Each phase produces deliverables needed by the next phase in the life cycle.

Requirements are converted into design. Code is generated during implementation that is driven by the design. This is the most common life cycle models, 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 before the next phase can begin. At the end of each phase, there is

ACHIEVER'S EDUCATIONAL CONSULTANT 0814 71097

always a review to ascertain if the project is in the right direction and whether or not to carry on or abandon the project. Unlike the general model, phases do not overlap in a waterfall model.

Waterfall Life Cycle Fig 2 Waterfall Life Cycle

Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process. Phases are processed and completed one at a time. Works well for smaller projects where requirements are very well understood.

V-Shaped Model

Each phase must be completed before the next phase begins. The test plan focuses on meeting the functionality specified in the requirements gathering. The low-level design phase is where the actual software components are designed, and unit tests are created in this phase as well. The implementation phase is, again, where all coding takes place.

Fig 3 V-Shaped Life Cycle Model 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 small projects where requirements are easily understood. 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 discovered during testing phases. Cycles are broken into smaller, more easily managed iterations. Each of the iterations goes through the requirements, design, implementation and testing phases. The first iteration produces a working version of software and this makes possible to have working software early on during the software life cycle.

Subsequent iterations build on the initial software produced during the first iteration.

Incremental Life Cycle Model Fig 4 Incremental Life Cycle Model

Each of the iterations is an easily managed landmark Each phase of an iteration is rigid and do not overlap each other. Problems as regard to system architecture may arise as a result of inability to gathered requirements up front for the entire software life cycle. Spiral Model The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases namely Planning, Risk Analysis, Engineering and Evaluation.

A software project continually goes through these phases in iterations which are called spirals. In the baseline spiral requirements are gathered and risk is assessed. Requirements are gathered during the planning phase. In the risk analysis phase, a process is carried out to discover risk and alternate solutions.

A prototype is produced at the end of the risk analysis phase. Software is produced in the engineering phase, alongside with testing at the end of the phase. The evaluation phase provides the customer with opportunity to evaluate the output of the project to date before the project

continues to the next spiral.

[...]

Software is produced early in the software life cycle.

Requirements Phase

ACHIEVER'S EDUCATIONAL CONSULTANT 08141710097

Business requirements are gathered in this phase. Meetings with managers, stake holders and users are held in order to determine the requirements. The requirements development phase may have been preceded by a feasibility study, or a conceptual analysis phase of the project. In systems engineering, a requirement can be a description of what a system must do, referred to as a Functional Requirement.

This type of requirement specifies something that the delivered system must be able to do. Such requirements are often called Non-functional requirements, or 'performance requirements' or 'quality of service requirements. Non-functional requirements which are the ones that act to constrain the solution.

Nonfunctional requirements are sometimes known as quality requirements or

Constraint requirements No matter how the problem is solved the constraint requirements must be adhered to. It is important to note that functional requirements can be directly implemented in software. The non-functional requirements are controlled by other aspects of the system. For example, in a computer system reliability is related to hardware failure rates, performance controlled by CPU and memory.

Non-functional requirements can in some cases be broken into functional requirements for software. For example, a system level non-functional safety requirement can be decomposed into one or more functional requirements. In addition, a non-functional requirement may be converted into a process requirement when the requirement is not easily measurable. For example, a system level

maintainability requirement may be decomposed into restrictions on software constructs or limits on lines or code.

Requirements must be actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design. Studies reveal that insufficient attention to Software Requirements Analysis at the beginning of a project is the major reason for critically weak projects that often do not fulfil basic tasks for which they were designed.

The Need for Requirements Analysis

Requirements are a description of how a system should behave, in other words, a description of system properties or attributes. This job is dedicated to a specialized Requirements Analyst. The Requirements Analysis function may also come under the scope of Project Manager, Program Manager or Business Analyst, depending on the organizational hierarchy.

Steps in the Requirements Analysis Process

Here information is gathered from the multiple stakeholders identified. The Requirements Analyst brings out from each of these groups what their requirements from the application are and what they expect the application to achieve. Taking into account the multiple stakeholders involved, the list of requirements gathered in this manner could go into pages.

Problems faced in Requirements Elicitation

Other methods like flowcharting of business processes and the use of existing documentation like user manuals, organizational charts, process models and systems or process specifications, on-site analysis, interviews with end-users, market research and competitor analysis are also used widely in Requirements Elicitation. There are of course, modern tools that are better equipped to handle the complex and multilayered process of Requirements Elicitation. 4 Requirements Analysis The moment all stakeholder requirements have been gathered, a structured analysis of these can be done after modeling the requirements. After requirements have been elicited, modeled and analyzed, they should be documented in clear, definite terms.

The software requirements specification is a document that lists out stakeholders' needs and communicates these to the technical community that will design and build the system. It is really a challenge to communicate a well-written requirements specification, to both these groups and all the sub-groups within.

Specifications may be documented separately as

It describes the function of the system, performance of the system and the operational and user-interface constraints that will govern system development. Design Phase The software system design is formed from the results of the requirements phase. This is where the details on how the system will work are produced. Definition of software design A software design is a meaningful

engineering representation of some software product that is to be built.

A design can be traced to the customer's requirements and can be assessed for quality against predefined criteria. The design process is very important. In the same way, the approach to building software products is no unlike. As such, software design serves as the foundation for all software engineering steps that follow regardless of which process model is being employed.

During the design process the software specifications are changed into design models that express the details of the data structures, system architecture, interface, and components. Each design product is re-examined for quality before moving to the next phase of software development. At the end of the design process a design specification document is produced. This document is composed of the design models that describe the data, architecture, interfaces and components.

Design Specification Models

Data design – created by changing the analysis information model into data structures needed to implement the software. Part of the data design may occur in combination with the design of software architecture. More detailed data design occurs as each software component is designed. Architectural design - defines the relationships among the major structural elements of the software, the “design patterns” that can be used to attain the requirements that have been defined for the system, and the constraint that affect the way in which the architectural patterns can be applied.

It is derived from the system specification, the analysis model, and the subsystem interactions defined in the analysis model . Much of the necessary information required is provided by the e data flow and control flow diagrams. Component-level design – It converts the structural elements defined by the software architecture into procedural descriptions of software components using information acquired from the process specification , control specification , and state transition diagram . In order to assess the quality of a design the yardstick for a good design should be established.

Design Guidelines

These criteria are not acquired by chance. The software design process promotes good design through the application of fundamental design principles, systematic methodology and through review. “The design process is a series of steps that allow the designer to describe all aspects of the software to be built.

The set of principles which has been established to help the software engineer in directing the design process are

The design process should not suffer from tunnel vision – Alternative approaches should be considered by a good designer. Designer should judge each approach based on the requirements of the problem, the resources available to do the job and any other constraints. These patterns should always be chosen as an alternative to reinvention. The design should reduce intellectual distance between the software and the problem as it exists in the real world – This means that, the structure of the software design should imitate the structure of the problem domain.

The design should show 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. It should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.

The design should be reviewed to minimize conceptual errors –

There is sometimes the tendency to focus on minute details when the design is reviewed, missing the forest for the trees. The only design decisions made of the coding level address the small implementation details that enable the procedural design to be coded.

Fundamental Software Design Concepts

Over the past four decades, a set of fundamental software design concepts has evolved, each providing the software designer with a foundation from which more sophisticated design methods can be applied.

[...]

Those phases are executed. Software life cycle models describe phases of the software cycle and the order in which those phases are executed. In general model, each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design.

Code is produced during implementation that is driven by the design. In a waterfall model, each phase must be completed in its entirety before the next phase can begin. 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. Unlike what I mentioned in the general model, phases do not overlap in a waterfall model.

Multiple development cycles take place here, making the life cycle a “multiwaterfall” cycle. Cycles are divided up into smaller, more easily managed iterations. Each iteration passes through the requirements, design, implementation and testing phases.

The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The baseline spirals, starting in the planning phase, requirements are gathered and risk

is assessed. In requirement phase business requirements are gathered and that the phase is the main focus of the project managers and stake holders.

Toronto, McGraw-Hill Ryerson Modularity

In unit 2 we discussed about software lifecycle models in general and also in detailed the requirement and the design phases of software development. In this unit we will look at Modularity in programming. Differentiate between Top-down and Bottom-up programming approach Modularity is a general systems concept which is the degree to which a system's components may be separated and recombined. In essence, the software is divided into separately names and addressable components called modules that are integrated to satisfy problem requirements.

It is important to note that a reader cannot easily understand large programs with a single module. The number of variables, control paths and sheer complexity make understanding almost impossible. As a result a modular approach will allow for the software to be intellectually manageable. However, it is important to note that software cannot be subdivided indefinitely so as to make the effort required to understand or develop it negligible.

This is because the more the number of modules, the less the effort to develop them.

Logical Modularity

Generally in software, modularity can be categorized as logical or physical. Logical Modularity is concerned with the internal organization of code into logically-related units. In modern high level languages, logical modularity usually starts with the class, the smallest code group that can be defined. In languages such as Java and C#, classes can be further combined into packages which allow developers to organize code into group of related classes.

Depending on the environment, a module can be implemented as a single class, several classes in a package, or an entire API . You should be able to describe the functionality of tour module in a single sentence . Your module should expose its functionality as simple interfaces that shield callers from all implementation details.

Physical Modularity

Physical Modularity is probably the earliest form of modularity introduced in software creation. These are usually a variation of the JAR file format with special meta data to target the intended runtime environment. The current trend of adoption seems to be pointing to OSGi as a generic physical module format. OSGi provides the Java environment with additional functionalities that should allow developers to model their modules to scale from small emddeable to complex enterprise components .

Benefits of Modular Design

With the proper testing framework, developers can exercise each module without having to bring

up the entire project. Modularity lets you build complex system composed of smaller parts that can be independently managed and maintained. Developers have the ability to fix and make non-infrastructural changes to module without affecting other modules. The updated module can independently go through the build and release cycle without the need to re-build and redeploy the entire system.

This allows your software system to scale in functionality without becoming brittle and a burden on developers.

Process-oriented design

This approach places the emphasis on the process with the objective being to design modules that have high cohesion and low coupling.

[...]

Evaluate the first iteration of the program structure to reduce coupling and improve cohesion. Once program structure has been developed modules may be exploded or imploded with aim of improving module independence. An exploded module becomes two or more modules in the final program structure. An imploded module is the result of combining the processing implied by two or more modules.

An exploded module normally results when common processing exists in two or more modules and can be redefined as a separate cohesive module. When high coupling is expected, modules can sometimes be imploded to reduce passage of control, reference to global data and interface complexity. The structure shown inside the cloud in Fig. 3 does not make effective use of factoring.

Fig 6 Example of a program structure

Evaluate module interfaces to reduce complexity, reduce redundancy, and improve consistency. The scope of effect of a module is defined as all other modules that are affected by a decision made by that module. For example, the scope of control of module e is all modules that are subordinate i.e. modules f, g, h, n, p and q. Interfaces should be designed to pass information simply and should be consistent with the function of a module.

The module in question should be reevaluated. Modules that have internal “memory” can be unpredictable unless care is taken in their use.

Strive for controlled entry modules, avoid pathological connection

This warns against content coupling. Software is easier to understand and maintain if the module interfaces are constrained and controlled. Modular programming can be performed even where the programming language lacks explicit syntactic features to support named modules. Software tools can create modular code units from groups of components.

Libraries of components built from separately compiled modules can be combined into a whole by using a linker. Module Interconnection Languages Module interconnection languages provide formal grammar constructs for deciding the various module interconnection specifications required to assemble a complete software system. 3 Top-Down Design Top-down is a programming style, the core of traditional procedural languages, in which design begins by specifying complex pieces and then dividing them into successively smaller pieces. Finally, the components are precise enough to be coded and the program is written.

It is the exact opposite of the bottom-up programming approach which is common in object-oriented languages such as C++ or Java. The method of writing a program using top-down approach is to write a main procedure that names all the major functions it will need. These compartmentalized sub-routines finally will perform actions so straightforward they can be easily and concisely coded. The program is done when all the various sub-routines have been coded.

Merits of top-down programming

Separating the low level work from the higher level abstractions leads to a modular design. Modular design means development can be self contained. Having »skeleton« code illustrates clearly how low level modules integrate.

Fewer operations errors

Much less time consuming . Very optimized way of processing each programmer has to apply their own knowledge and experience to their parts . Easy to maintain .

4 Bottom-up approach

We need to use a lot of perception to decide the functionality that is to be provided by the module. This approach is more suitable if a system is to be developed from existing system, because it starts from some existing modules.

What are the steps to create effective modules?

The benefits of modular programming cannot be overemphasised. It among other things, allows for scalar development, it facilitates code testing, helps in building robust system, allows for easier modification and maintenance. Modularity is a general systems concept, the degree to which a system's components may be separated and recombined.

Unit 4 Pseudo code

In the last unit, you have learnt about Modularity in programming. Its benefits, design approaches and criteria, attributes of a good Module and the steps to creating effective module. You equally learnt about Top-Down and Bottom-up approaches in programming. This unit ushers you into Pseudo code a way to create a logical structure

that will describing the actions, which will be executed by the application.

Explain General guidelines for writing Pseudo code.

Definition of Pseudo code

The big advantage of the pseudo-code is that the application logic can be easily comprehended by any developer in the development team. In addition, when the application algorithm is expressed in pseudo-code, it is very easy to convert the pseudo-code into real code .

Here are a few general guidelines for writing your pseudo code

Using parameters in specifying procedures is concise, clear, and accurate, and hence should not be omitted from pseudocode. Don't lose sight of the underlying model. Note that the operations are numbered and each operation is unambiguous and effectively computable. We also extract and list all variables used in our pseudo-code.

Pseudo-code the task of computing gross pay given pay rate and hours worked. This example presents the conditional control structure. In both cases pseudocode is resumed at line 5. This is an example of a top-test or while do iterative control structure.

There is also a bottom-test or repeat until iterative control structure which executes a block of statements until the condition tested at the end of the block is False.

Some Keywords That Should be Used

Do not include data declarations in your pseudo code.

Write a pseudo code to find the average of even number between 1 and 20

The role of pseudo-code in program design cannot be underestimated.

What is Pseudo code

Explain the General guidelines for writing Pseudo code.

Unit 5 Programming Environment, CASE Tools & HIPO Diagrams

In the last unit, you have learnt about pseudo code. In this unit you will be exposed to Programming Environment, CASE Tools & HIPO Diagrams.

Definition of Programming Environment

Programming environments help the creation, modification, execution and debugging of programs. The goal of integrating a programming environment is more than simply building tools that share a common data base and provide a consistent user interface. In the early 1970s the tools became famous with Unix with tools like grep, awk and make that were meant to be combined flexibly with pipes. As some tools have been maintained, they have been integrated into more powerful integrated development environments .

These environments combine functionality into one place, sometimes increasing simplicity and productivity, other times part with flexibility and extensibility. The workflow of IDEs is routinely contrasted with alternative approaches, such as the use of Unix shell tools with text editors like Vim and Emacs. The ability to use a variety of tools productively is one quality of a skilled software engineer.

Integrated development environments

They are usually simpler and make it easier to do simple tasks, such as searching for content only in files in a particular project. IDEs are often used for development of enterprise-level applications.

[...]

CASE tools are a class of software that automates many of the activities involved in various life cycle phases. Subsequently, system designers can use automated design tools to transform the prototyped functional requirements into detailed design documents. Programmers can then use automated code generators to convert the design documents into code. Automated tools can be used collectively, as mentioned, or individually.

The next competitor into the market was Excelerator from Index Technology in

Burroughs machines, the allure of IBM was strong, and Excelsior came to prominence.

AD/Cycle which was an alliance of software vendors centered around IBM's Software repository using IBM DB2 in mainframe and OS/2

With the decline of the mainframe, AD/Cycle and the Big CASE tools died off, opening the market for the mainstream CASE tools of today. Interestingly, nearly all of the leaders of the CASE market of the early 1990s ended up being purchased by Computer Associates, including IEW, IEF, ADW, Cayenne, and Learmonth & Burchett Management Systems .

Categories of Case Tools CASE Tools can be classified into 3 categories

Toolkits Toolkits are loosely integrated collections of products easily extended by aggregating different tools and workbenches. Typically, the support provided by a toolkit is limited to programming, configuration management and project management. And the toolkit itself is environments extended from basic sets of operating system tools, for example, the Unix Programmer's Work Bench and the VMS VAX Set. In addition, toolkits' loose integration requires user to activate tools by explicit invocation or simple control mechanisms.

The resulting files are unstructured and could be in different format, therefore the access of file from different tools may require explicit file format conversion. However, since the only constraint for adding a new component is the formats of the files, toolkits can be easily and incrementally extended. The environment itself is written in the programming language for which it was developed, thus enable users to reuse, customize and extend the environment. A process-centered environment operates by interpreting a process model created by specialized tools.

Process-model production

Examples are East, Enterprise II, Process Wise, Process Weaver, and Arcadia.

Application areas of CASE Tools

From project management software through tools for business and functional analysis, system design, code storage, compilers, translation tools, test software, and so on.

Common CASE risks and associated controls include

Generally, management must be willing to accept a long-term payback period. Controls include requiring senior managers to define their purpose and strategies for implementing CASE technologies. Controls include protecting the repositories with appropriate access, version, and backup controls. HIPO Diagrams The HIPO technique is a tool for planning and/or documenting a computer program.

A HIPO model consists of a hierarchy chart that graphically represents the program's control structure and a set of IPO charts that describe the inputs to, the outputs from, and the functions performed by each module on the hierarchy chart. Using the HIPO technique, designers can evaluate and refine a program's design, and correct flaws prior to implementation. By its very nature, the HIPO technique is best used to plan and/or document a hierarchically structured program. The HIPO technique is often used to plan or document a structured program. A variety of tools, including pseudocode and structured English can be used to describe processes on an IPO chart.

System flowcharting symbols are sometimes used to identify physical input, output, and storage devices on an IPO chart.

Components of HIPO

Each box represents one module that can call its subordinates and return control to its higher-level parent.

[...]

The module names consist of an active verb followed by a subject that suggests the module's function. The objective of the module identifiers is to uniquely identify each module and to indicate its place in the hierarchy. Some designers use Roman numerals or letters to designate levels. The box at the lower-left of Figure 7 is a legend that explains how the arrows on the hierarchy chart and the IPO charts are to be interpreted.

The IPO charts An IPO chart is prepared to document each of the modules on the hierarchy chart. Where appropriate, system flowcharting symbols are used to identify the physical devices that generate the inputs and accept the outputs. The processes are typically described in brief paragraph or sentence form. They often do not appear in the completed documentation package.

Detail diagrams A detail diagram is a low-level IPO chart that shows how specific input and

output data elements or data structures are linked to specific processes. In effect, the designer integrates a system flowchart into the overview diagram to show the flow of data and control through the module. Figure 7.2 shows a detail diagram for module 2.0, Update stock. The process steps are written in pseudocode.

Note that the first step writes a menu to the user screen and input data flows from that screen to step 2. The solid black arrows at the top and bottom of the process box show that control flows from module 1.0 and, upon completion, returns to module 1.0. Within the case structure are other solid black arrows. Note that each subroutine is documented in a separate IPO chart.

Following the default case, the arrow points to an on-page connector symbol numbered 1. Note the matching on-page connector symbol pointing to the select structure. On-page connectors are also used to avoid crossing arrows on data flows. Figure 7.2 A detail diagram for process 2.1.

The notes might specify access methods, data types, and so on. Figure 64.4 shows a detail diagram for process 2.1. The module writes a template to the user screen, reads a stock number and a quantity from the screen, uses the stock number as a key to access an inventory file, and updates the stock on hand. Note that the logic repeats the data entry process if the stock number does not match an inventory record.

A real IPO chart is likely to show the error response process in greater detail. Simplified IPO charts Some designers simplify the IPO charts by eliminating the arrows and system flowchart symbols and showing only the text. Often, the input and out put blocks are moved above the process block , yielding a form that fits better on a standard 8.5 11 sheet of paper. Some programmers insert modified IPO charts similar to Figure 64.5 directly into their source code as comments.

Because the documentation is closely linked to the code, it is often more reliable than stand-alone HIPO documentation, and more likely to be maintained.

HIPO —A tool for planning and/or documenting a computer program that utilizes a hierarchy chart to graphically represent the program's control structure and a set

of IPO charts to describe the inputs to, the outputs from, and the functions performed by each module on the hierarchy chart.

Software

The examples in this # were prepared using Visio.

What is HIPO technique?

With the aid of well labeled diagrams, discuss the components of Hipo. After studying the unit you are expected to have achieved the following objectives listed below.

What is Implementation?

Since code is produce here, the developer regards this phase as the main focus of the life cycle. Implementation my overlap with both the design and testing phases. The implementation phase concerns with issues of quality, performance, baselines, libraries, and debugging.

Deliverable Implementation Code Table 1 The Implementation Phase

There may indeed be many known errors in the system. They are usually listed in the release notes and have well established work arounds. Actually, the system is likely to have many, yet-to-be-discovered errors. When a company is attempting new software integration, it can be hectic Six Sigma is a management approach meant to discover and control defects.

A summary of Six Sigma can be found in Natasha Baker's "Key Concepts of Six Sigma.

Defining the Implementation

By defining the goals, projects, and deliverables your company will have greater direction during the changeover. The goals and projects must be measurable. Measurement of the Implementation Goals and projects must be usable with metrics. By using Six Sigma measurement methods, it is possible to follow user understanding, familiarity, and progress accurately.

It should be noted that, continuous data is more useful than discrete data. Implementation Analysis Analysis is important to tackle defects occurrence. The Six Sigma method examines essential relationships and ensures all factors are considered. For example, in a software implementation trial, employees are frustrated and confused by new processes.

Careful analysis will look at the reasons behind the confusion.

Implementation Improvement

In the example utilizing the team members, perhaps utilizing proficient resources to mentor struggling resources will help. Six Sigma improvements depends upon experimental design and carefully constructed analysis of data in order to keep further defects in the implementation

process at bay.

Control of the Implementation

If implementation is going to be successful, control is important. This ensures that the implementation does not fail. Any deviations from the goals set demand correcting before they become defects. For example, if you notice your team does not adapt quickly enough, you need to identify the causes before the deadline.

Major Tasks in Implementation

This part provides a brief description of each major task needed for the implementation of the system. The tasks described here are not particular to site but overall project tasks that are needed to install hardware and software, prepare data, and verify the system. Add as many subsections as necessary to this section to describe all the major tasks adequately. The tasks described in this section are not site-specific, but generic or overall project tasks that are required to install hardware and software, prepare data, and verify the system.

Examples of major tasks are the following

It should include the primary security features associated with the system hardware and software. This part addresses security issues particularly related to the implementation effort. It will be necessary to consider for example, if LAN servers or workstations will be installed at a site with sensitive data preloaded on nonremovable hard disk drives.

Implementation Support

This section offers a list of support equipment and includes all hardware used for testing time implementation. For example, if a client/server database is implemented on a LAN, a network monitor or “sniffer” might be used, along with test programs. This section provides a list of software and databases required to support the implementation. Identify the software by name, code, or acronym.

Identify any software used to facilitate the implementation process. This section identifies the physical facilities and accommodations required during implementation. Examples include physical workspace for assembling and testing hardware components, desk space for software installers, and classroom space for training the implementation staff. This section provides a list of required support materials, such as magnetic tapes and disk packs.

Personnel

This section describes personnel requirements and any known or proposed staffing requirements. It also describes the training, to be provided for the implementation staff.

Personnel Requirements and Staffing

This section, describes the number of personnel, length of time needed, types of skills, and skill levels for the staff required during the implementation period. If particular staff members have been selected or proposed for the implementation, identify them and their roles in the implementation.

Training of Implementation Staff

If appropriate, identify which courses particular types of staff should attend by job position description. If training will be provided by one or more commercial vendors, identify them, the course name, and a brief description of the course content. If the training will be provided by State staff, provide the course name and an outline of the content of each course. Identify the resources, support materials, and proposed instructors required to teach the course.

Performance Monitoring

This section describes the performance monitoring tool and techniques and how it will be used to help decide if the implementation is successful.

Implementation Requirements by Site

The “X” in the subsection number should be replaced with a sequenced number beginning with I. Each subsection with the same value of “X” is associated with the same implementation site. If a complete set of subsections will be associated with each implementation site, then “X” is assigned a new value for each site.

1 Site Name or identification for Site X

This section provides the name of the specific site or sites to be discussed in the subsequent sections.

2 Site Requirements

This section defines the requirements that must be met for the orderly implementation of the system and describes the hardware, software, and sitespecific facilities requirements for this area. Hardware Requirements - Describe the site-specific hardware requirements necessary to support the implementation . Software Requirements - Describe any software required to implement the system . Data Requirements - Describe specific data preparation requirements and data that must be available for the system implementation.

An example would be the assignment of individual IDs associated with data preparation. Facilities Requirements - Describe the site-specific physical facilities and accommodations required during the system implementation period. Some examples of this type of information are provided in Section 3.

3 Site implementation Details

Include a description of the implementation team, schedule, procedures, and database and data updates. Team--If an implementation team is required, describe its composition and the tasks to be performed at this site by each team member. Describe the required tasks in chronological order with the beginning and end dates of each task. If appropriate, charts and graphics may be used to present the schedule.

Procedures--Provide a sequence of detailed procedures required to accomplish the specific hardware and software implementation at this site. If necessary, other documents may be referenced. If appropriate, include a step-by-step sequence of the detailed procedures. A checklist of the installation events may be provided to record the results of the process.

If the system will replace an already operating system, then address the startup and cutover processes in detail. If there is a period of parallel operations with an existing system, address the startup procedures that include technical and operations support during the parallel cycle and the consistency of data within the databases of the two systems. Database--Describe the database environment where the software system and the database, if any, will be installed. Include the host computer database operating procedures, database file and library naming conventions, database system generation parameters, and any other information needed to effectively establish the system database environment.

Include database administration procedures for testing changes, if any, to the database management system before the system implementation. Data Update--If data update procedures are described in another document, such as the operations manual or conversion plan, that document may be referenced here. This section specifies when to make the go/no go decision and the factors to be included in making the decision.

Restart and recovery procedures Back-Off Plan

This section describes the process for reviewing the implementation and deciding if it was successful. It describes how an action item list will be created to rectify any noted discrepancies. It also references the Back-Off Plan for instructions on how to back-out the installation, if, as a result of the post-implementation verification, a no-go decision is made.

Unit 2 Testing Phase

In the last unit, we looked at implementation phase of software development. In this unit, we shall consider the testing phase. It is important for stakeholders to have information

about the quality of product , hence the importance of testing cannot be overemphasised. Explain testing methods.

Definition of software testing Software testing is an empirical examination carried out to provide stakeholders with information about the quality of the product or service under test. Instead, it provides a comparison that put side by side the state and behavior of the product against the instrument someone applies to recognize a problem. These instruments may include specifications, contracts, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

Testing methods

Consequently, the tester inputs data into, and only sees the output from, the test object. Specification-based testing though necessary, but it is insufficient to guard against certain risks. «.

[...]

Static testing - White box testing includes all static testing.

Regression Testing

The extent of testing depends on the phase in the release process and the risk of the added features.

Non Functional Software Testing The following methods are used to test non-functional aspects of software

This is generally referred to as software scalability. This activity of Non Functional Software Testing is often referred to as Endurance Testing. Stability testing checks to see if the software can continuously function well in or above an acceptable period. This activity of Non Functional Software Testing is oftentimes referred to as load testing.

Usability testing is used to check if the user interface is easy to use and understand. Security testing is essential for software that processes confidential data to prevent system intrusion by hackers. Internationalization and localization is needed to test these aspects of software, for which a pseudo localization method can be used. An example of non-functional testing is software fault injection, in the form of fuzzing.

Destructive testing

Destructive testing attempts to cause the software or a sub-system to fail, in order to test its robustness. Another practice is to start software testing at the same time the project starts and it continues until the project finishes.

Testing can be done on the following levels

Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system. Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

Finally, acceptance testing can be conducted by the end-user, customer, or client to validate whether or not to accept the product. Acceptance testing may be performed as part of the hand-off process between any two phases of development. Testing Tools Program testing and fault detection can be aided significantly by testing tools and debuggers.

[...]

Discuss the various testing methods. Software testing is an empirical examination carried out to provide stakeholders with information about the quality of the product or service under test. A test engineer used these approaches to describe his opinion when designing test cases. **Software Quality Assurance**

In the last unit, we looked at testing phase of software development. Discuss Software Quality Assurance During the Software Acquisition Life Cycle. Concepts and Definitions Software Quality Assurance is defined as a planned and systematic approach to the evaluation of the quality of and adherence to software product standards, processes, and procedures. SQA includes the process of assuring that standards and procedures are established and are followed throughout the software acquisition life cycle.

Compliance with agreed-upon standards and procedures is evaluated through process monitoring, product evaluation, and audits. Software development and control processes should include quality assurance approval points, where an SQA evaluation of the product may be done in relation to the applicable standards.

Standards and Procedures

Procedures are the established criteria to which the development and control processes are compared. The SQA role is to ensure their existence and adequacy. Proper documentation of standards and procedures is necessary since the SQA activities of process monitoring, product evaluation and auditing rely upon clear definitions to measure project compliance. Documentation Standards specify form and content for planning, control, and product documentation and provide consistency throughout a project.

Procedures are explicit steps to be followed in carrying out a process. All processes should have documented procedures. Examples of processes for which procedures are needed are configuration management, non-conformance reporting and corrective action, testing, and formal inspections. If developed according to the NASA DID, the Management Plan describes the software development control processes, such as configuration management, for which there have to be procedures, and contains a list of the product standards.

Standards are to be documented according to the Standards and Guidelines DID in the

Audits are a key technique used to perform product evaluation and process monitoring. Review of the Management Plan should ensure that appropriate SQA approval points are built into these processes. Product evaluation is an SQA activity that assures standards are being followed. Ideally, the first products monitored by SQA should be the project's standards and procedures.

SQA assures that clear and achievable standards exist and then evaluates compliance of the software product to the established standards. Product evaluation assures that the software product reflects the requirements of the applicable standard as identified in the Management Plan. Process monitoring is an SQA activity that ensures that appropriate steps to carry out the process are being followed. SQA monitors processes by comparing the actual steps carried out with those in the documented procedures.

The Assurance

section of the Management Plan specifies the methods to be used by the SQA process monitoring activity. A fundamental SQA technique is the audit, which looks at a process and/or a product in depth, comparing them to established procedures and standards. Audits are used to review management, technical, and assurance processes to provide an indication of the quality and status of the software product.

SQA assures that

Software configuration identification is consistent and accurate with respect to the numbering or naming of computer programs, software modules, software units, and associated software documents. Configuration status accounting is performed accurately including the recording and reporting of data reflecting the software's configuration identification, proposed changes to the configuration identification, and the implementation status of approved changes. Software configuration authentication is established by a series of configuration reviews and audits that exhibit the performance required by the software requirements specification and the configuration of the software is accurately reflected in the software design documents. Software development libraries provide for proper handling of software code, documentation, media, and related data in their various forms and versions from the time of their initial approval or acceptance until they have been incorporated into the final media.

The SQA role in formal testing is described in the next section. SQA's main objective in reviews is to assure that the Management and Development Plans have been followed, and that the product is ready to proceed with the next phase of development. Although the decision to proceed is a management decision, SQA is responsible for advising management and participating in the decision. An inspection or walkthrough is a detailed examination of a product on a step-by-step or line-of-code by line-of-code basis to find errors.

For inspections and walkthroughs, SQA assures, at a minimum that the process is properly completed and that needed follow-up is done. The inspection process may be used to measure compliance to standards. Formal Test Monitoring SQA assures that formal software testing, such as Acceptance testing, is done in accordance with plans and procedures. The test procedures are testing the software requirements in accordance with test plans.

The test procedures are followed. Nonconformances occurring during testing are noted and recorded.

Software Quality Assurance during the Software Acquisition Life Cycle

In addition to the general activities described in subsections C and D, there are phasespecific SQA activities that should be conducted during the Software Acquisition Life Cycle. At the conclusion of each phase, SQA concurrence is a key element in the management decision to initiate the following life cycle phase.

Software Concept and Initiation Phase

Assuring adherence to approved design standards as designated in the Management

Plan. Assuring all software requirements are allocated to software components. Assuring that a testing verification matrix exists and is kept up to date. Assuring the Interface Control Documents are in agreement with the standard in form and content.

Reviewing PDR documentation and assuring that all action items are

Assuring the approved design is placed under configuration management.

Software Detailed Design Phase SQA activities during the detailed design phase include

Assuring that approved design standards are followed.

Assuring that allocated modules are included in the

Assuring that results of design inspections are included in the design. Reviewing CDR documentation and assuring that all action items are Software Implementation Phase detailed design.

Software Integration and Test Phase SQA activities during the integration and test phase include

Certifying that testing is complete and software and documentation are ready for delivery. Participating in the Test Readiness Review and assuring all action items are completed.

Software Acceptance and Delivery Phase

During these development cycles, SQA conducts the appropriate phasespecific activities described above. Techniques and Tools SQA should evaluate its needs for assurance tools versus those available off-the-shelf for applicability to the specific project, and must develop the others it requires. Useful tools might include audit and inspection checklists and automatic code standards analyzers.

There is the need to ensure Software quality and adherence to software product standards, processes, and procedures and this is what Software Quality Assurance is out to achieve.

Discuss Software Quality Assurance Activities

In this unit, we shall look at Compatibility testing. Explain Usefulness of Compatibility Testing.

What is Compatibility Testing?

Many things are decided n compatibility of a system/application being developed with, for example, other systems/applications, OS, Network. This explains the reason why the efforts of developers may appear to be in vain.

Browser compatibility testing which can also be referred to as user experience testing requires that the web applications are tested on different web browsers, to ensure the

In terms of functionality, the application must behave and respond the same way across different browsers. This is sometimes referred to as backward and forward compatibility. For example, Windows 98 was developed with backward compatibility for Windows 95. Database compatibility testing is used to evaluate an application/system's performance in connection to the database it will interact with.

Certification testing falls within the range of Compatibility testing. What is Browser compatibility testing Compatibility testing is highly beneficial to software development. This helps in figuring out necessary changes/modifications/additions required to make the system/application compatible with the computing environment.

What are the elements in computing environment?

2-11, October 1990, New York, New York, United States

S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang.

Myra B. Cohen , Matthew B. Dwyer , Jiangfan Shi, Coverage and adequacy in software product line testing, Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis, p.

Uni 5 Software verification and validation

In the last unit, we considered Compatibility testing . You will recall that, Compatibility testing is highly beneficial to software development. In this unit we are going to look at software verification and validation. What is Verification and Validation Verification and Validation is the process of checking that a software system meets specifications and that it fulfils its expected purpose. Validation is the process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. Validation ensures that ‘you built the right thing’. Verification ensures that ‘you built it right’.

Validation confirms that the product, as provided, will fulfill its intended use. Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use. Accreditation is the formal certification that a model or simulation is acceptable to be used for a specific purpose. Verification is the process of determining that a computer model, simulation, or federation of models and simulations implementations and their associated data accurately represents the developer's conceptual description and specifications.

Classification of methods

In this case, syntactic methods are often used. The Quality Assurance team prepares test cases for verification and these help to determine if the process that was followed to develop the final product is right. Other methods, such as reviews, provide for validation in Software Development Life Cycle provided it is used early for validation.

Software Verification & Validation Model

‘Verification & Validation Model’ is used in improvement of software project development life cycle.

Fig 8 Verification and Validation Model

Software Verification Model helps to achieve this and also help to improve the quality of the software product.

Few terms involved in Verification

It usually led by a leader, which properly reviews the documents and work product during various phases of the product development life cycle. The product, as well as related documents is presented to the team, the members of which carry different interpretations of the presentation. The bugs that are discovered during the inspection are conveyed to the next level in order to take care of them.

Walkthroughs

ACHIEVER'S EDUCATIONAL CONSULTANT 08/11/20097

In walkthrough inspection is carried out without formal preparation . Though walkthroughs can help in finding bugs, they are used for knowledge sharing or communication purpose.

Buddy Checks

Buddy Checks does not involve a team rather, one person goes through the documents prepared by another person in order.

Code Validation/Testing

Code testing other than Unit Testing can be done by testers or developers.

Integration Validation/Testing

This test helps in finding out if there is any defect in the interface between different modules.

Functional Validation/Testing

This type of testing is meant to find out if the system meets the functional requirements. In this type of testing, the system is validated for its functional behavior. Functional testing does not deal with internal coding of the project, in stead, it checks if the system behaves as per the expectations.

User Acceptance Testing or System Validation

The product is validated to find out if it works according to the system specifications and satisfies all the user requirements.

Activity L

Discuss Software Verification & Validation Model The importance of Verification and Validation cannot be overemphasised. Verification and Validation checks that a software system meets specifications and that it fulfils its intended purpose. Verification and Validation is the process of checking that a software system meets specifications and that it fulfils its intended purpose.



**NATIONAL OPEN UNIVERSITY OF NIGERIA,
PLOT 91 CADASTRAL ZONE, NNAMDI AZIKIWE EXPRESSWAY, JABI – ABUJA
FACULTY OF SCIENCES**

NOVEMBER 2018 EXAMINATION

COURSE CODE: CIT 333

COURSE CREDIT: 2

COURSE TITLE: SOFTWARE ENGINEERING

TIME ALLOWED: 2 Hours

INSTRUCTION: ANSWER QUESTION 1 AND ANY OTHER THREE (3) QUESTIONS

QUESTIONS

1a. Imagine that you had applied to a software development firm for industrial training, and that you were accepted and deployed to work in the software engineering unit. State five (5) key roles you would play in that unit in order to ensure that you are relevant.
(10 marks)

1b. Assuming you were assigned the role of a project manager in a software development firm that requires you to cross-check their products before distribution, which software life cycle model would you employ in this firm. (1 mark)

1c. With the aid of a well-labelled diagram, outline the phases to be undertaken in the course of applying the software life cycle model chosen in 1b. (10 marks)

1d. State two reasons for the model you employed in 1b. (4 marks)

[Total = 25 marks]

2a. Within the context of formal software testing, write down any six (6) goals of software quality assurance. (6 marks)

2b. Give a short explanation of three (3) common standards' in Software Engineering (6 marks)

2c. State any three (3) challenges encountered in the course of undertaking the process of requirements elicitation. (3 marks)

[Total = 15 marks]

3a. Enumerate four (4) major software quality assurance activities undertaken during the detailed design phase. (4 marks)

3b. List six (6) elements required for compatibility testing in a computing setting. (6 marks)

3c. State the main reason for establishing buddy checks. (2 marks)

3d. Outline any three (3) merits of the incremental life cycle model. (3 marks)

[Total = 15 marks]

4a. Write short notes on the following:

- i. Product evaluation (4 marks)
- ii. Product monitoring (4 marks)

4b. State the key processes involved in the following phases:

- i. Software Acceptance and Delivery Phase (2 marks)
- ii. Software Sustaining Engineering and Operations Phase (2 marks)

4c. List any three (3) demerits of the spiral life cycle model. (3 marks)

[Total = 15 marks]

5a. Give a brief description of the following:

- i. Symbolic debugging tool (2 marks)
- ii. Profiling tool (2 marks)
- iii. Alpha testing (2 marks)
- iv. Acceptance testing (2 marks)

5b. List the four (4) levels of testing in software engineering: (4 marks)

5c. Write down any three (3) elements in the computing environment that will require compatibility testing. (3 marks)

[Total = 15 marks]

ACHIEVER'S EDUCATIONAL CONSULTANT 08141710097