J15 EDUCATIONAL CONSULTS

MOTTO: BRINGING KNOWLEDGE TO YOUR DOORSTEP
TEL: 07013644096, 09057133772

CIT432 SOFTWARE ENGINEERING II

Define software engineering in your own words.

Software engineering can be defined as the act of employing established engineering principles in the development of good, functional, reliable and maintainable computer software. Software engineering is deals with software developed by teams rather than individual programmers.

Why Study Software Engineering?

- software development needs the structured application of scientific and engineering principles in order to analyse, design, construct, document and maintain it
- like any engineering development, large-scale software development also requires the disciplined application of project management principles
- because of software's growing importance, its development must be managed more carefully than other areas of large projects
- individual approach is no longer appropriate; and the departure point for proper software development should be the realisation that software development has grown from an art to a craft, and to a proper engineering discipline
- to acquire skills to be a better programmer: this makes for higher

productivity and better quality programs

 to acquire skills to develop large programs

 it helps you to gather ability to solve complex programming Problems

 to learn techniques of specification, design, interface development, testing and integration, project management etc.

discuss the basic features of each generations

First Generation

This generation came up during early 1950s. In this generation, computers were programmed by changing the wires and tens of dials and switches. Sometimes, these setting could be stored on paper tape that looked like a ticker paper from telegraph - a punch tape or punched card. With tape and or card, the computers were commanded what, how and when to do something. Programming then was done using machine language; so to have a flawless, program, a programmer needed to have very detailed knowledge of the computer.

Second Generation (2GL)

This generation came into existence in the mid of 1950s. This generation made use of symbols and are called an assembler (an assembler is a program that translates symbolic instructions to processor instruction.) The programmer no longer work with one's and zero's when using an assembly language but symbols. These symbols are called mnemonics. Each mnemonics stands for one single machine instruction. However, for each processor, a different assembler was written.

Third Generation (3GL)

This generation came into existence at the end of 1950s. This generation witnessed “natural language” but interpreters and compliers were made. (An Interpreter is a translator that translates high level languages on a statement-by-statement basis so that as each language statement is encountered, it is converted to machine executable codes and executed while a complier is a translator that transforms high-level languages into computer executable language). In 3GL, there was no longer need to work in symbols instead, a programmer could use a programming language that resembled more to “natural language” e.g. FORTRAN, COBOL PASCAL etc.

Fourth Generation (4GL)

In the fourth generation, the primary feature was that you do not indicate HOW a computer must perform a task but WHAT it must do. A few instructions in a 4GL will do the same as hundreds of instructions in a lower generation languages. In most of these cases, one deals with database management system. A trained user in this kind of software can create an application in a much shorter time for development and debugging than would be possible with other generations.

Fifth Generation (5GL)

The basis of this generation was laid in the 1990s by using sound, moving images and agents. Software for the end user may be based on principles of knowbot-agents. An automatic self changing piece of software that creates new agent based on the behaviour of the end user. Human-like quality DNA/RNA (intelligent) algorithms could also play a big role.

list the basic characteristics of software.

Software is Developed or Engineered

Software is not manufactured in a classical sense. Software is different from physical products in a manufacturing plant. Although some similarities exist between software development and hardware manufacturing, the two activities are fundamentally different.

Software is usually Custom-Built

Even though we hear of software components, not many components are there off-the-shelf. Most of the software are custom-built rather than being assembled from existing components. It is however expected that in the coming years, software component and reusability will catch up.

Software does not Wear Out

Unlike any other physical product, software does not wear out. Software is not subjected to the environmental factors like heat, dust, vibration etc.

Cost for Support and Modification of Delivered Software is High

Research has shown that cost for support and modification of delivered software over the life of a system are typically twice the cost of the original acquisition. Even more significant is the fact that the customer's perception of what the software should be able to do changes as experience is gained with its use.

Explain the relationship between the system software and the application software.

System software is computer software designed to operate the computer hardware and to provide and maintain a platform for running application software. System software helps use the operating system and computer system. The purpose of system software is to insulate the applications programmer as much as possible from the details of the particular computer complex being used, especially memory and other hardware features, and such accessory devices as communications, printers, readers, displays, keyboards etc. System software includes diagnostic tools, compilers, servers, window systems, utilities, language translator, data communication programs, database systems and more.

The most basic types of system software are: Operating system, the computer BIOS and device firmware, and the utility software.

Application Software

Application software, also known as an application, is computer software designed to help the user to perform singular or multiple related specific tasks. Examples include word processor, presentation software, spreadsheet, desktop publishing, enterprise software, accounting software, office suites, graphics software, and media players etc.

Explain the following:

i. The low level languages

The low level languages are divided into two; the machine language and the assembly language.

- Machine Language: It deals directly with the computer hardware. It uses 0's and 1's to form commands that cause the

computer to perform series of operations as specified by the programmer. Machine language is difficult to use and more time consuming.

- Assembly Language: This is also a low level language. The assembly language uses symbols instead of 0's and 1's. This language reduced the complexity of program authoring. However, each computer or family of computers has its own assembly language which prevented the software of one computer model from being used on a different computer model.

ii. The high level languages

High level languages are more like natural languages of the computer users. These types of languages do not bother about the knowledge of the computer hardware. They were developed for two reasons; for the programmer to work on different computers without having to learn a new assembly language each time, and secondly, for software written on one computer could be used on another. Translators (a complier, or interpreter) were used to help solve these problems by translating program into machine language and checking the program syntax errors.

List any seven (7) software engineering models you know.

- Build-and-Fix Model
- Spiral Model
- Rapid Prototyping Model
- Waterfall Model
- Incremental Model
- Clean Room

 Extreme Programming

When to Use the Build-and-Fix Model

The build-and-fix model should be used when the product is small and there is no possibility of the product ever having to be maintained in the future. For example, if a student is to write a 25-50 line home work problem to solve a particular computational need (e.g. program to keep track of student's record in the class), then it would be a waste of time to specify, plan and design the development effort.

List the Phases Involved in Waterfall Model

 Requirements stage/ phase

 Specification stage/phase

 Planning stage/phase

 Design stage/phase

 Implementation stage/phase

 Integration stage/phase

 Operations stage/phase

 Retirement.

Describe Waterfall Model

The waterfall model is an approach to software development that emphasizes completing a phase of the development before proceeding to the next phase. The waterfall model was derived from engineering models to put some order in the development of large software product. It consists of different stages which are processed in a linear fashion. In

conjunction with certain phase completions, a baseline is established that "freezes" the products of the development at that point. If a need is identified to change these products, a formal change process is followed to make the change. The graphic representation of these phases in software development resembles the downward flow of a waterfall. In the waterfall model, no phase is started until the result of the previous phase has been carefully verified.

Where to Use the Waterfall Model

Because of the strengths and the weaknesses shown above, the application of the waterfall model should be in situations where the requirements and the implementation of those requirements are very well understood and also when the software to be produced is large. For example, if a company has experience in building accounting systems, I/O controllers, or compilers, then building another such product based on the existing designs is best managed with the waterfall model.

When should developers use the spiral model

Spiral model is particularly useful in ADE (Aerospace, Defense and Engineering) projects, because they are risky in nature. They tend to use mature technology and to work well-known problems. Spiral model is also applicable to many business applications, especially those for which success is not guaranteed or the applications require much computation, such as in decision support systems.

Advantages and Disadvantages of the Rapid Prototyping Model

Advantages include

a. developers can benefit from the experience gained from building the prototype and apply this experience towards building a better product

b. gathers client's feedback early in the process to avoid costly redesign later

c. early functionality

d. provides a process to perfect the requirements definition.

e. provides risk control

f. documentation focuses on the end product not the evolution of the product.

Disadvantages include

a. adding the rapid prototype will lengthen the requirements phase

b. depending on the type of software system, it may not be possible to build a meaningful prototype without considerable effort

c. less applicable to existing systems than to new, original development.

Advantages and Disadvantages of Spiral Model

Advantages include:

a. spiral model combines the best features of the waterfall and prototyping models

b. it address risks associated with software development

c. it enables the developer to apply prototyping at any stage in the evolution of the software product

d. it control costs and risk through prototyping

e. allows for work force specialisation

f. facilitates allocation of resources

g. does not require a complete set of requirements at the onset.

Disadvantages include:

a. the overall cost is comparatively high

b. it is complicated

c. it is unstable for small projects were risks are modest

d. requires considerable risk assessment expertise

Explain software conversion.

Conversion is simply a change over from the manual or less automated existing system to the newly developed system. It is however expedient to employ a systematic change over from the old process to the newly automated process.

Explain the different types of software conversion

Abrupt Conversion (Cut-Over)

This is a conversion strategy where on a specific date chosen, the old system is terminated and the new system is placed into operation. Jeffery L. et al (2001). This approach of software conversion may be a high risk approach since they may still be major problems that will not be discovered until the new system has been in operation for some time.

Parallel Conversion

Here, both the old and the new system are operated simultaneously for some time period. Jeffery L. et al (2001). This approach ensures that all major problems in the new system are discovered and solved before the old system is discarded. The final cut-over may be either abrupt or gradual as portions of the new system are deemed adequate.

Staged Conversion

This is the type of conversion based on the version concept. Jeffery L. et al (2001). Here, each successive version of the new system is converted as it is developed. The conversion of each version may be abrupt, parallel or location.

Location Conversion

This type of conversion applies when the same system will be used in many different geographical areas. Here, location takes place at one location first using either abrupt or parallel conversion. When the new system is certified by the site, it will be adopted by other sites. The cutover now may be abrupt since other sites have certified the newly built product ok.

Explain the three basic aspect of the feasibility study.

- Budget: The budget aspect seeks to find out if there are financial constraints that will make the realisation of the software project impossible. It extends to examining an estimate of the overall cost of the project
- Expertise: The expertise aspect of the feasibility study deals with the technical know-how needed to make the project feasible. It seeks to answer the questions such as; are there some experts who

specialise in that area?

 Technology: The technology aspect of the feasibility study examines the enabling technology that will make the realisation of the software project feasible (possible). You need to check for example, if there are programming languages available that can be used to develop such software project etc.

Discuss the advantages and disadvantages of on-site observation method of data collection.

Advantages

a. data gathered by observation can be highly reliable

b. the development team will be able to see exactly what is being done

c. observation is relatively inexpensive compared with other factfinding methodology etc.

Disadvantages

a. because people usually feel uncomfortable when being observed, they may unknowingly perform differently when being observed.

b. the work being observed may not involve the level of difficulty or volume experienced during that time period.

c. the tasks being observed are subject to various types of interruptions etc.

Qualities of "Good Code"

 Consistency: consistent code is written in such a way that it is easy for people to understand how the program works. When

reading consistent code, one subconsciously forms a number of assumptions and expectations about how the code works.

 Cleanliness: Clean code is written in a form that is easy to read. It is in a form that people can read it with minimum effort so that they can understand it easily.

 Correctness: A code is designed to be correct to a point that people spend less time worrying about bugs and more time enhancing the features of a program. Correct code does not crash.

 Extensibility: Extensible code is written in a form that programmers can easily add new features to the program. These are codes that are easier to reuse and modify without much assumptions.

Limitations of Formal Methods

 Formal method is used as an adjunct to, not a replacement for, standard quality assurance methods

 Formal methods are not a panacea, but can increase confidence in a product's reliability if applied with care and skill

 Formal methods, though very useful for consistency checks, cannot assure completeness of a specification.

Dwell concisely on the two basic classifications of formal methods as discussed

Basic Classifications

Formal methods can be used at a number of levels as shown below:

 Level 0: Formal specification may be undertaken and then a

program developed from this informally. This may be the most cost-effective option in many cases.

 Level 1: Formal development and formal verification may be used to produce a program in a more formal manner. For example, proofs of properties or refinement from the specification to a program may be undertaken. This is most appropriate in high-integrity systems involving safety or security.

 Level 2: Theorem provers may be used to undertake fully formal machine-checked proofs. This can be very expensive and is only practically worthwhile if the cost of mistakes is extremely high (e.g., in critical parts of microprocessor design).

Classification as with Programming Language Semantics

As with programming language semantics, styles of formal methods may be roughly classified as follows:

 Denotational semantics: Here, the meaning of a system is expressed in the mathematical theory of domains. Proponents of such methods rely on the well-understood nature of domains to give meaning to the system; critics point out that not every system may be intuitively or naturally viewed as a function.

 Operational semantics: Here, the meaning of a system is expressed as a sequence of actions of a (presumably) simpler computational model. Proponents of such methods point to the simplicity of their models as a means to expressive clarity; critics oppose that the problem of semantics has just been delayed (who defines the semantics of the simpler model?).

 Axiomatic semantics: Here, the meaning of the system is

expressed in terms of preconditions and postconditions which are true before and after the system performs a task, respectively. Proponents note the connection to classical logic; critics note that such semantics never really describe what a system does (merely what is true before and afterwards).

Features of VDM

- **Syntax checking:** The syntax-checker verifies whether the syntax of the selected files matches the VDM++ language specifications. If the check passes, it gives access to the other features of VDMTools.
- **Type checking:** The type-checker tests mis-uses of values and operators and can also show places, where runtime errors may occur.
- **Code generation:** VDMTools is able to generate a fully executable code for about 95% of all VDM++ constructs. Code generation is available for Java and C++.
- **Specification manager:** A manager-window displays all classes and files in the specification. It also shows the status for each file.
- **Interpreter and Debugger:** VDMTools allows the execution of all executable VDM++ constructs. Debugging is also supported.
- **Integrity examiner:** It extends the static checking capabilities of VDM++. The tool scans through all sources to find possible inconsistencies or integrity violations. The examiner creates expressions which should evaluate to be true. If they evaluate to be false, there may be a problem.

 Rose-VDM++ Link: The Rational-Rose-Link provides a bidirectional link between Rational Rose (UML) and the Toolbox (VDM++).

 Java to VDM++ Translator: It is possible to generate a VDM++ specification from a Java application. The generated model can be examined at VDM++ level.

 Several input types: Models for VDMTools can be written either with Microsoft Word (RTF) or in Latex. Plain text is also possible but is not recommended.

Define Software Project Management

Project management can be defined as the process of planning, organising, staffing, directing and controlling the production of software. Project management can also be said to mean the art of planning, organising, directing and controlling of resources for a finite period of time to complete specific goals and objectives.

List and explain seven properties of the crystal clear Methodology

1. Frequent Delivery: When delivering is working, tested code to the actual software users once every few months (or more often, if possible), users will be able to deliver feedback on implemented requirements, sponsors will see progress and developers will get a morale boost.

2. Reflective Improvement: Taking time to let the team reflect on what works and what does not work for the project, and improving the things that do not work.

3. Osmotic Communication: Having the entire team so close together (if possible in the same room, otherwise in adjacent rooms) that people do not have to go to a lot of trouble to raise answer or answer questions, but can do so instantly, will make people work together naturally, inspect each others' work and pick up relevant information as if by osmosis.

4. Personal Safety: If people feel safe to speak up without fear of reprisal, they can give constructive criticism on other people's work and admit their own mistakes, leading to honesty and ultimately, trust.

5. Focus: If everybody has time to focus on their primary objectives for two hours a day, for two consecutive days every week, without any distractions that can make them lose their train of thought (like meetings or other work), people will be more focused and work will be finished quicker.

6. Easy Access to Expert Users: If expert users are available to the team, they can answer questions and deliver feedback on quality and design decisions.

7. Technical Environment with Automated Tests, Configuration Management and Frequent Integration: A proper technical environment where testing and configuration management/version control tasks (like making backups and merging changes) do not have to be done by hand will make life easier for developers.