CMP 403 SOFTWARE ENGINEERING

Group Assignment

GROUP MEMBERS:

1.	EJIGA ALEXANDER ACHILE.	BHU/20/04/05/0046
2.	OBENTA CHRISTOPHER OWOICHO.	BHU/20/04/05/0090
3.	IKEMKA ROMASON ROMANUS.	BHU/20/04/05/0016
4.	OGUH CHUBIYOJO FAVOUR.	BHU/20/04/05/0048
5.	KPONGO BEMSEN CHUKWUMA.	BHU/20/04/05/0115
6.	ADAMANI MIVANYI ENAN.	BHU/20/04/05/0044
7.	YOROM JOSHUA RIMAN	BHU/20/04/05/0014
8.	IDAMINABO CLEMENT-DAVIES	BHU/20/04/05/0054
9.	ADEJORI OLUWAFERANMI	BHU/20/04/05/0080

Summary and learning outcomes of chapter 8 & 9

Answers:

Chapter 8 - Software Testing:

The topics discussed in this chapter include development testing, test-driven development, release testing, user testing.

Testing is an action intended to show that a program does what it is intended to do and to discover program defects before it is used. You check the results of the test run for errors, anomalies or information about the program's non-functional attributes. Testing is part of a more general verification and validation process.

The goal of program testing is to demonstrate to the developer and the customer that the software meets its requirements and to discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.

The first goal leads to validation testing i.e checking if the program performs correctly using a set of test cases that reflect the system's expected use. The second goal leads to defect testing. The test cases are designed to expose defects. The test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used.

The contrast between verification and validation is such that verification asks the question "Are we building the product right".The software should conform to its specification. And validation: asks "Are we building the right product". The software should do what the user really requires.

The difference between software inspection and software testing is that inspections are concerned with analysis of the static system representation to discover problems (static verification) and software testing is concerned with exercising and observing product behaviour (dynamic verification).

The advantages of inspection include not having to be concerned with interactions between errors., incomplete versions of a system can be inspected without additional costs and an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability. Inspections and testing are complementary and not opposing verification techniques.

The stages of testing are development testing, release testing, user testing. **Development testin**g includes all testing activities that are carried out by the team developing the system. These activities include: unit testing, component testing, and system testing.

Unit testing is the process of testing individual components in isolation. Units may be: Individual functions or methods within an object, object classes with several attributes and methods and composite components. Complete test coverage of a class involves testing all operations associated with an object , setting and interrogating all object attributes, and exercising the object in all possible states. There are 2 unit test case that should be done; the first should reflect normal operation of a program and should show that the component works as expected. The other kind of test should use abnormal inputs to check that these are properly processed and do not crash the component. Unit Testing strategies include:

Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way then choose tests from within each of these groups.

Guideline-based testing where you use testing guidelines to choose test cases.

These guidelines reflect previous experience of the kinds of errors that programmers often make when developing components.

Components testing focuses on showing that the component interface that are made up of several interacting objects behaves according to its specification. Its objectives are to detect faults due to interface errors or invalid assumptions about interfaces.

Interface types include:

Parameter interfaces Data passed from one method or procedure to another.

Shared memory interfaces Block of memory is shared between procedures or functions.

Procedural interfaces Sub-system encapsulates a set of procedures to be called by other sub-systems.

Message passing interfaces Sub-systems request services from other sub-systems.

Common component interface errors include Interface misuse, Interface misunderstanding, timing errors.

System testing during development involves integrating components to create a version of the system and then testing the integrated system.The focus in system testing is testing the interactions between components. System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces. During system testing, reusable components that have been separately developed may be integrated with other components. The complete system is then tested.

Testing policies define the required system test coverage may be developed.

Examples of testing policies:

i. All system functions that are accessed through menus should be tested.
ii. Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.

iii. Where user input is provided, all functions must be tested with both correct and incorrect input.

Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development. Code is developed incrementally, along with a test for that increment. TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

The activities involved in TDD are:

i. Start by identifying the increment of functionality that is required.
ii. Write a test for this functionality and implement this as an automated test.
iii. Run the test, along with all other tests that have been implemented.
iv. Implement the functionality and re-run the test.

Once all tests run successfully, implementing the next set of functionality.

The benefits of TDD are:

i. Code coverage : Every code segment that you write has at least one associated test so all code written has at least one test.
ii. Regression testing :A regression test suite is developed incrementally as a program is developed.
iii. Simplified debugging :When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
iv. System documentation: The tests themselves are a form of documentation that describe what the code should be doing.

Release testing is the process of testing a particular release of a system that is intended for use outside of the development team. The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.

Release testing is a form of system testing but System testing focuses on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Requirements-based testing involves examining each requirement and developing a test or tests for it.

Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.

Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.types of user testing are; Alpha testing, Beta testing, Acceptance testing.

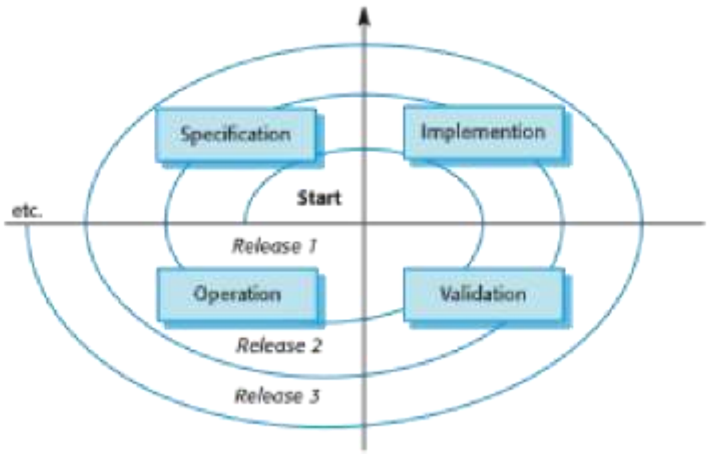
Stages in acceptance testing include Define acceptance criteria, Plan acceptance testing, Derive acceptance tests, Run acceptance tests, Negotiate test results, Reject/accept system

In agile methods, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.

Chapter 9 - Software Evolution:

The topics covered in this chapter include : Evolution processes, Program evolution dynamics, Software maintenance ,Legacy system management.
Software change is inevitable and can occur when; New requirements emerge when the software is used, the business environment changes, errors must be repaired, new computers and equipment is added to the system, the performance or reliability of the system may have to be improved.

Software evolution is important to organisations have huge investments in their software systems to maintain the value of these assets to the business, they must be changed and updated.. The majority of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.

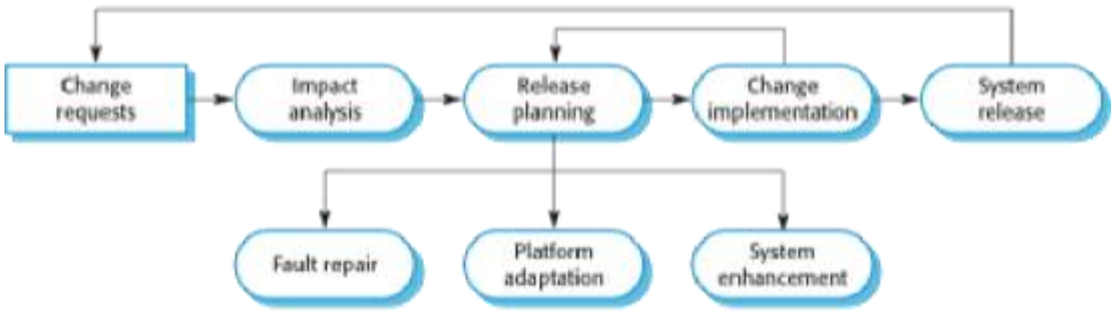


A spiral model of development and evolution

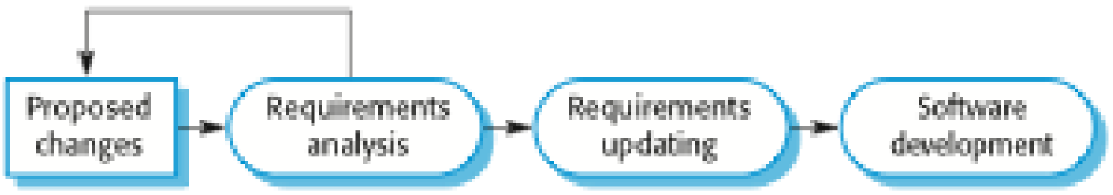
Evolution : The stage in a software system's life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.
Servicing stage: the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software's environment. No new functionality is added.
Phase-out stage : The software may still be used but no further changes are made to it.

Software evolution processes depend on:
i. The type of software being maintained;
ii. The development processes used;
iii. The skills and experience of the people involved.
iv. Proposals for change are the driver for system evolution.

v. Should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated.
vi. Change identification and evolution continues throughout the system lifetime.



Software Evolution Process



Change Implementation process

Change implementation process involves the Iteration of the development process where the revisions to the system are designed, implemented and tested. A critical difference is that the first stage of change implementation may involve program understanding, especially if the original system developers are not responsible for the change implementation. During the program understanding phase, you have to understand how the program is structured, how it delivers functionality and how the proposed change might affect the program.
Evolution is simply a continuation of the development process based on frequent system releases.
Program evolution dynamics is the study of the processes of system change. Lehman and Belady proposed that there were a number of 'laws' which applied to all systems as they evolved**. These laws include:**

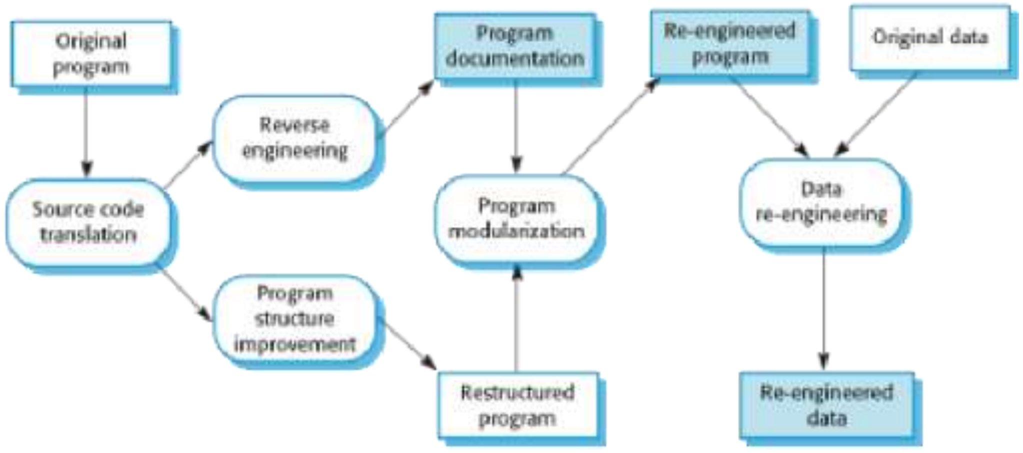
i. Continuing change:A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
ii. Increasing complexity: As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.

iii. Large program evolution: Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release.
iv. Organizational stability: Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
v. Conservation of familiarity: Over the lifetime of a system, the incremental change in each release is approximately constant.
vi. Continuing growth: The functionality offered by systems has to continually increase to maintain user satisfaction.
vii. Declining quality: The quality of systems will decline unless they are modified to reflect changes in their operational environment.
viii. Feedback system: Evolution processes incorporate multiagent, multiloop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Software Maintenance is modifying a program after it has been put into use it does not normally involve major changes to the system's architecture. Changes are implemented by modifying existing components and adding new components to the system.
Types of maintenance include:
i. Maintenance to repair software faults
ii. Maintenance to adapt software to a different operating environment
iii. Maintenance to add to or modify the system's functionality

Maintenance costs are usually greater than development costs (2* to 100* depending on the application). and is affected by both technical and non-technical factors. It increases as software is maintained. Maintenance costs factors include Team stability, Contractual responsibility, Staff skills, Program age and structure
Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs. Predicting the number of changes requires and understanding of the relationships between a system and its environment. Factors influencing this relationship are Number and complexity of system interfaces, Number of inherently volatile system requirements, The business processes where the system is used.
Predictions of maintainability can be made by assessing the complexity of system components.Complexity depends on , Complexity of control structures, Complexity of data structures, Object, method (procedure) and module size.
Process metrics may be used to assess maintainability based on number of requests for corrective maintenance;, average time required for impact analysis, average time taken to implement a change request, number of outstanding change requests. If any or all of these is increasing, this may indicate a decline in maintainability.

System re-engineering is Re-structuring or re-writing part or all of a legacy system without changing its functionality.Re-engineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented. The advantages include reduced risk and reduced cost.



Re-engineering processes include:
i. Source code translation
ii. Reverse engineering
iii. Program structure improvement
iv. Program modularisation
v. Data re-engineering

Refactoring is the process of making improvements to a program to slow down degradation through change. Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand. When you refactor a program, you should not add functionality but rather concentrate on program improvement.
Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing. Refactoring is a continuous process of improvement throughout the development and evolution process.
'Bad smells' in program code include: long methods, data clumping, speculative generality

Legacy system management for organizations that rely on legacy systems must choose a strategy for evolving these systems. They could either
i. Scrap the system completely and modify business processes so that it is no longer required;
ii. Continue maintaining the system;
iii. Transform the system by re-engineering to improve its maintainability;
iv. Replace the system with a new system.

v. The strategy chosen should depend on the system quality and its business value.
vi.

Issues in business value assessment

i. The use of the system
ii. The business processes that are supported
iii. System dependability

System quality assessment can be assessed through the following metrics

i. Business process assessment: How well does the business process support the current goals of the business?
ii. Environment assessment: How effective is the system's environment and how expensive is it to maintain?
iii. Application assessment: What is the quality of the application software system?

Factors used in environment assessment :

i. Supplier stability: Is the supplier still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, does someone else maintain the systems?
ii. Failure rate: Does the hardware have a high rate of reported failures?
iii. Age: How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to a more modern system.
iv. Performance: Is the performance of the system adequate?
v. Support requirements: What local support is required by the hardware and software? If there are high costs associated with this support, it may be worth considering system replacement.
vi. Maintenance costs: What are the costs of hardware maintenance and support software licences?
vii. Interoperability: Are there problems interfacing the system to other systems? Can compilers, for example, be used with current versions of the operating system?
viii. e.t.c

To collect quantitative data to make an assessment of the quality of the application system:

i. The number of system change requests
ii. The number of different user interfaces used by the system
iii. The volume of data used by the system.