SOFTWARE ENGINEERING
IAT - II Solutions

1. With a neat diagram explain interaction models.

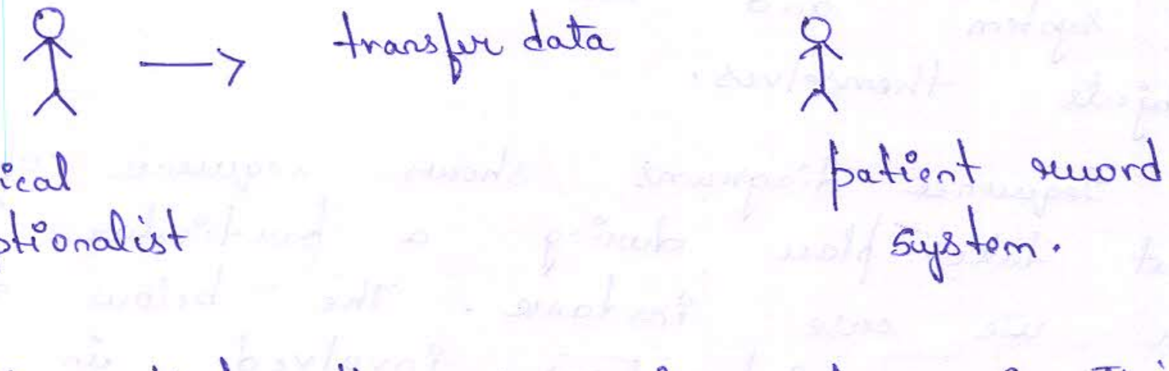
All systems involve interaction of some kind. this can be user interaction, interaction between systems or between components of the system. they are two approaches to interaction modeling.

1. Use case modeling - used to model interactions between a system and external actors.
2. Sequence diagrams - model interactions between components.

1. Use Case Modeling :

. Use case modeling is widely used to support requirements elicitation. each use case represents a discrete task that involves external interaction with the system. The below figure shows use case from MHC-PMS of general patient.

@Transfer data use case

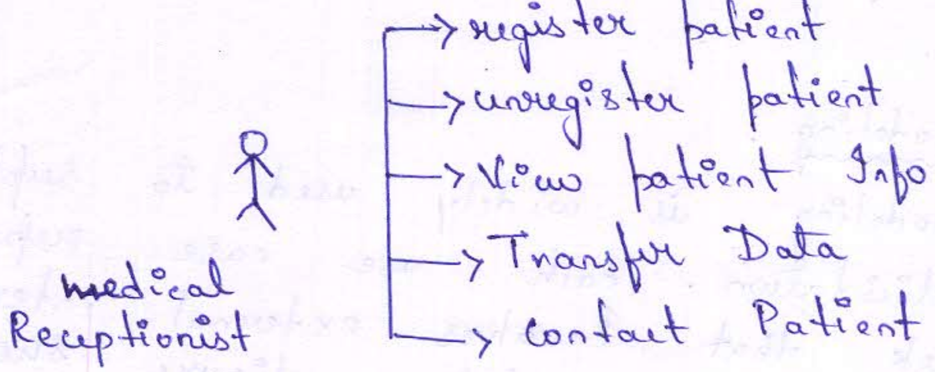


We notice that they are 2 actors in this use case, an operator who is transfering data and the patient record system.

⊙ use case diagrams give a fairly simple overview of an interaction so you have to provide more detail to understand what is involved.

⊙ This detail can either be a simple textual description, a structured description in table or a sequence diagram.

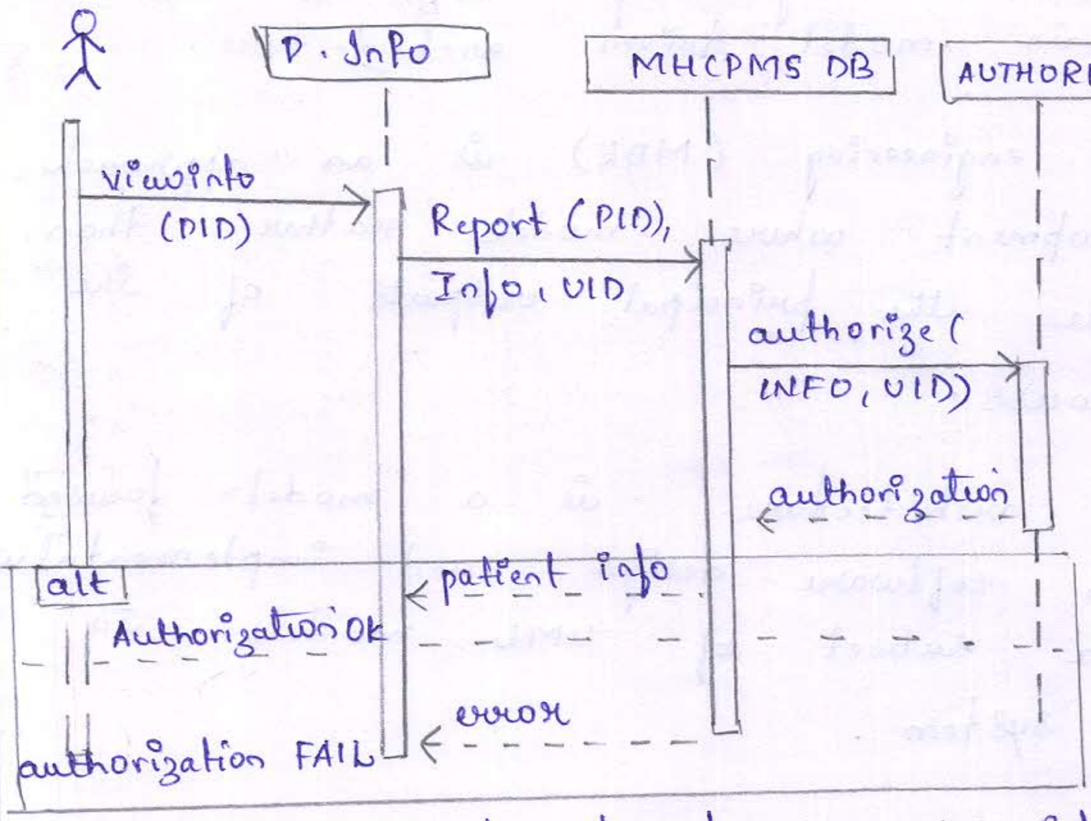
⊙ Sometimes we have to develop several diagrams to show a number of different use cases. For example all of use cases in the MHC-PMS in which 'Medical Receptionalist' are involved.



Sequence Diagrams

- Sequence diagrams are primarily used to model the interactions between actors and objects in a system and the interaction between the objects themselves.

⊙ Sequence diagrams shows sequence of interaction's that take place during a particular use case or use case instance. The below diagram models the interaction's involved in View Patient information use case where a medical receptionist can see some patient information.



1. Medical receptionist triggers viewinfo method in an instance P of the Patient Info object class, supplying patient's identifier , PID.

2. Instance P calls the database to return the information required.

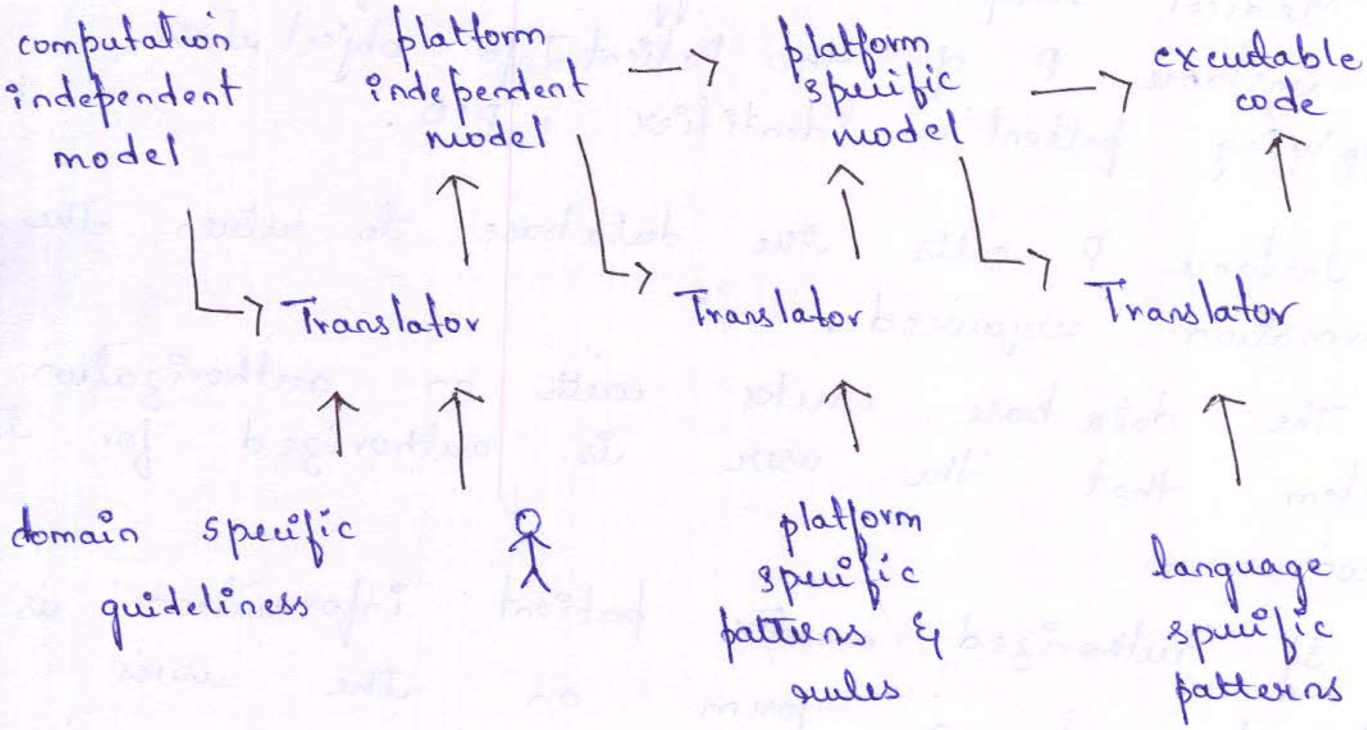
3. The data base checks with an authorization system that the user is authorized for this action.

4. If authorized , the patient information is returned and a form on the user's screen is filled in . If authorization fails then an error message is returned.

2) Define Model-driven engineering ? With a neat diagram explain model-driven architecture.

Model Driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.

Model-Driven architecture is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.



computation independent model
platform independent model
platform specific model
executable code
Translator
Translator
Translator
domain specific guidelines
platform specific patterns & rules
language specific patterns

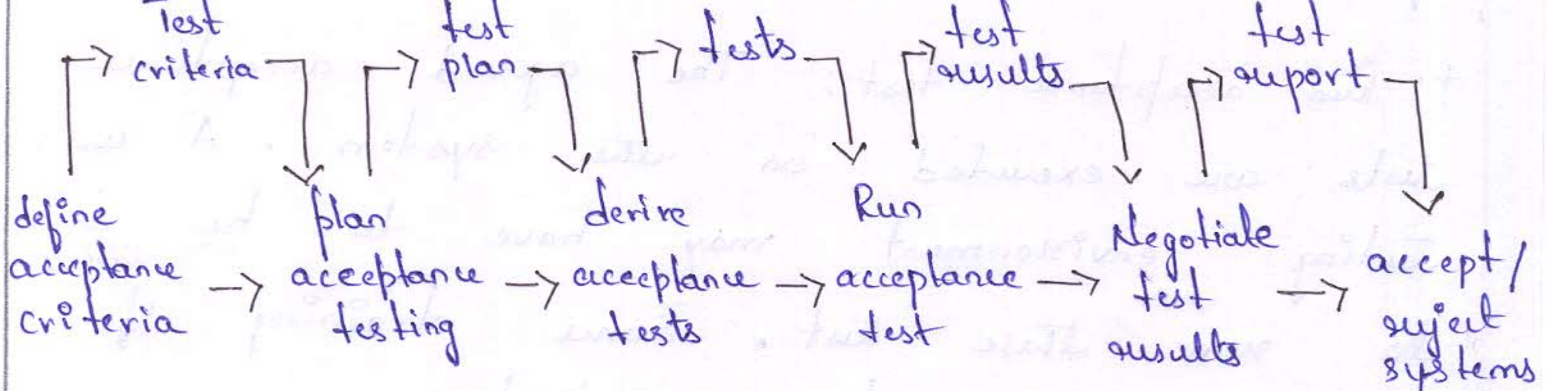
⊙ A computation independent model (CIM) that models the important domain abstractions used in the systems. CIM's are sometimes called domain models.

① A platform independent model (PIM) that model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show static system structure.

② Platform specific models (PSM) which are transformations of platform-independent model with a separate PSM for each application platform.

③ Define Acceptance testing. With a neat diagram explain acceptance testing process.

Acceptance testing is an inherent part of custom system development. It takes place after release testing. Acceptance implies that payment should be made for the system.



There are 6 stages in acceptance testing process shown in above figure. They are

1. Define acceptance criteria: This stage should ideally take place early in the process before the contract for the system is signed. The acceptance criteria should be part of the system contract and be agreed between the customer and developer.

2. Plan acceptance test: this involves deciding on the resources, time, and budget for acceptance testing and establishing a testing schedule. It should define risks to testing process, and discuss how these risks should be mitigated.

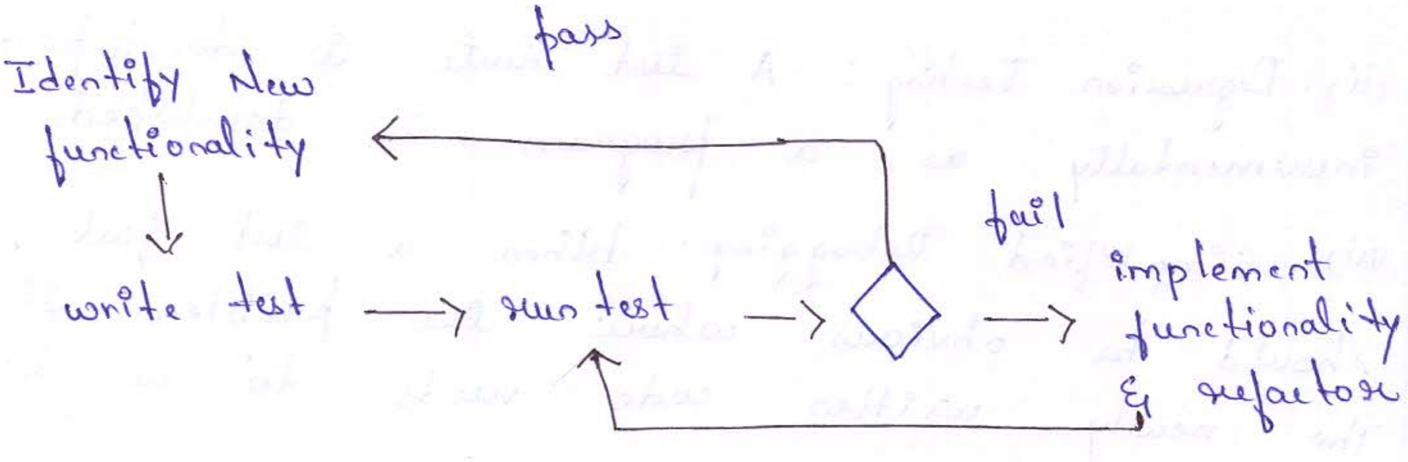
3. Derive acceptance test: Once acceptance criteria have been established, tests have to be designed to check whether or not a system is acceptable. Acceptance test should aim to test both functional and non-functional characteristics of the system.

4. Run acceptance test: The agreed acceptance tests are executed on the system. A user testing environment may have to be set to run these test. Some training of end-users may be required.

5. Negotiate test results: In this case the acceptance testing is complete and the systems can be handed over.

6. Reject / Accept System : This stage involves a meeting between the developers and the customer to decide on whether or not the system should be accepted.

04) Explain Test - Driven Development process in detail


pass
Identify New functionality
write test
run test
fail
implement functionality & refactor

The fundamental Test Driven Development (TTD) process are shown in following steps :

1. You start by identifying increment of functionality that is required . This should normally be small & implementable in a few lines of code.

2. You write a test for this functionality & implement this as an automated test.

3. You then run the test along with other test that have been implemented.

4. You then implement the functionality & re-run the test.

5. Once all test run successfully, you move on to implementing next chunk of functionality

Benefits of TTD

(i) Code Coverage: every code segment that you write should have at least one associated test. ∴ you can be confident that all of the code in the system has actually been executed.

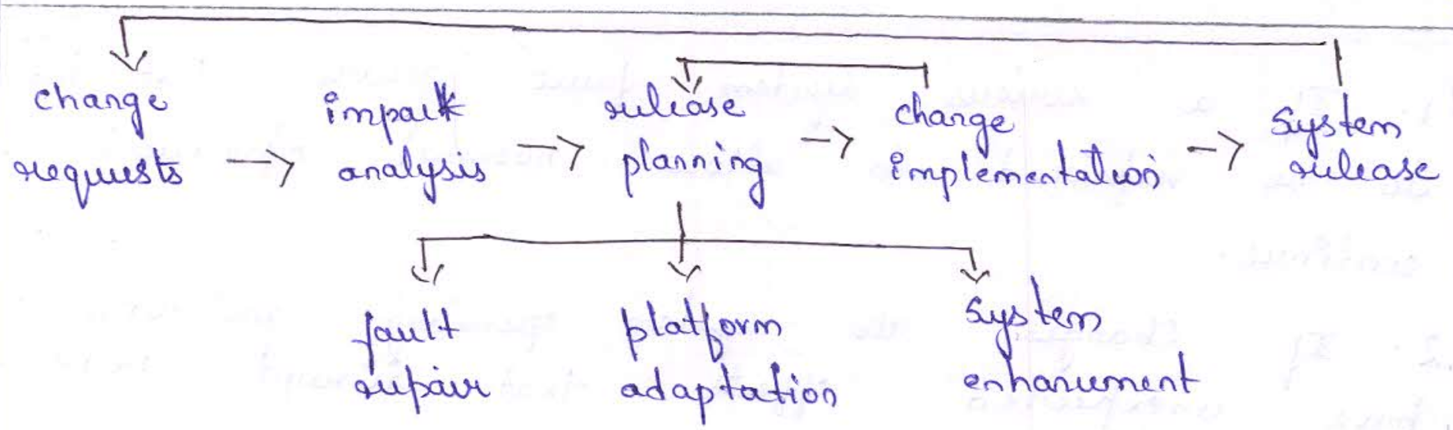
(ii) Regression Testing: A test suite is developed incrementally as a program is developed.

(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 act as a form of documentation that describe what the code should be doing.

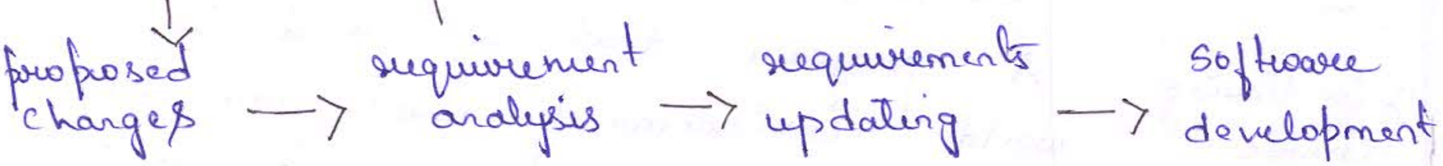
5) Define Software evolution. With neat diagram explain software evolution process.

Software evolution refers to the process of developing software initially, then repeatedly updating it for various reasons.

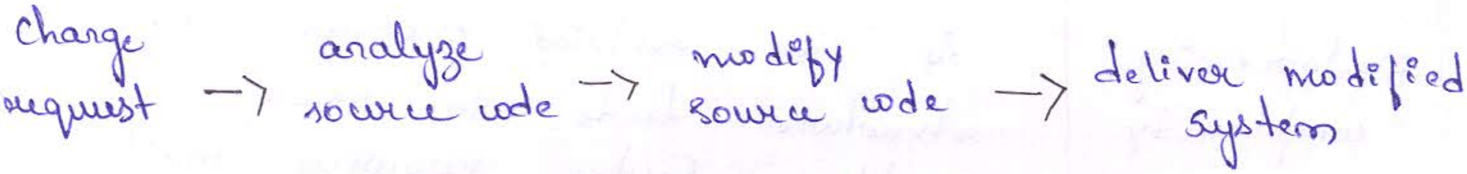


The process includes fundamental activities of change analysis, release planning, system implementation, and releasing a system to customers. The cost & impact of these changes are assessed to see how much of the system is affected by the change and how much it might cost to implement the change. If proposed Change are accepted a new release of the system is planned. All proposed changes are considered.

@ Change Implementation



@ emergency repair process



1. If a serious system fault occurs that has to be repaired to allow normal operation to continue.

2. If changes to system operating environment have unexpected effects that disrupt normal operation.

3. If there are unanticipated changes to the business running system affects system.

⊙ System repairs have to be completed as quickly as possible

⊙ This accelarates process of software ageing so that future changes become progressively more difficult and maintenance cost increase.

Evolution involves continuing agile development process.

6. State and explain Lehman's law

law	Description
1> Continuing change:	A program that is used in real world environment must necessarily change or else become progressively less useful in that environment.
2> Increasing complexity	As an evolving program change its structure tends to become more complex. Extra resources must be devoted to simplify structure

3. large program evolution - program evolution is self regulating process. System attribute such as size, time between devices and no. of reported errors is approximately invarient to each system.

4. Organizational Stability Over programs lifetime its rate of development is approximately constant & independent of resources devoted to system development.

5. conservation of familiarity Over life time of system the incremental Change in each release is approximately constant.

6. continuing growth functionality offered by system has to continuing increase to maintain users software.

7. Declining Quality The quality of system will decline until they are modified to reflect Change in their operational environment.

8. feebback system : evolution processes incorporate multi agent, multigroup feed back system & you have to treat them as feedback system to acheive significant product increment

7. Write note on

i) Configuration Management: It is a name given to general process of managing & changing software system.

1) Version Management: Where support is provided to keep track of different versions of software components

2) System Integration: Where support is provided to help develop us define what version of components are we use to create each version of system.

3) Problem Tracking: Where support is provided to allow users to report bugs & other problems & to allow all developers to see who is working on problems & when they are fixed.

(ii) Host-Target Development.

A software development platform should provide range of tools to support software engg. process. This include

(i) integrated compiler & syntax directed editing system that allows you to create, edit, create & compile code.

(ii) a language debugging system.

(iii) Graphical editing tools such as tools to edit, UML models.

(IV) Testing tools such as JUNIT that can automatically run a set of test on a new version of program.

(V) Project support tools that helps in organizing code for different development projects.