Date :

Experiment No - 7

Aim : To design deployment diagram for Expense Tracker Application.

Theory : Deployment diagram is a structure diagram which shows architecture of system as deployment of software artifact deployment targets.

Artifact represent concrete elements in the physical world that are a the result a development process. Examples of artifacts are executable files, libraries, archives database schemas, configuration files etc.

Deployment target is usually represented by a node which is either hardware device or some software execution environment. Nodes could be connected through communication paths to create networked systems of arbitrary complexity.

Deployment diagrams could describe architecture at specification level or at instance level.

Instance level deployment diagram shows deployment of instances of artifacts to specific instances of deployment targets. It could be used for example to show differences in deployments to development, staging or production environments with the name of specific build or deployment servers or devices.

Some common types of deployment diagrams are :

- Implementation (manifestation) of component by artifacts
- Specification level deployment diagram.
- Instance level deployment diagram
- Network architecture of the system

Manifestation of components by Artifacts :

While component diagrams show components & relationships b/w components & classifiers and deploy diagrams deployments of artifacts to deployment targets some missing intermediate diagram i manifestation diagram to be used to show manifestation of components by artifacts and internal structure of artifacts.

Specification level Deployment Diagram :

Specification level deployment diagram shows some overview of deployment of artifacts to deployment targets, without referencing specific instances of artifacts or nodes.

Instance Level Deployment Diagram :

Instance level deployment diagram shows deployment of instances of artifacts to specific instances of

tangets. It could be used for example to show differences in deployments to development staging or production environment with the names/ids of specific deployment servers or devices.

Specification level Network architecture:
Deployment diagrams could be used to show logical or physical network architecture of system. Network architecture diagram could show no artifacts or deployments at all or only the major ones.

Conclusion: We successfully designed deployment diagram for expense tracker application.

Expt. No. 8 Experiment No- 8 Date :

Aim : Write a steps for code engineering, forward and reverse engineering.

Theory :- Following are the steps required for forward, code and reverse engineering.

Code Engineering :-

1. Requirement Gathering : Gather all the requirements for the software that you are going to build.
2. System Analysis and Design : Analyze the requirements and design the system architecture and data model
3. Implementation :- Write code for the software using programming language or a development platform.
4. Testing : Test the software to ensure that it meets the requirements and functions properly.
5. Deployment : Deploy the software in a production environment.

Forward Engineering :

1. Requirement Gathering : Gather all the requirements for the software that you are going to build.
2. System Analysis and Design : Analyze the requirements and design the system architecture and model
3. Code Generation :- Use a modeling tool tool to generate code based on the system design.

4. Implementation: Write additional code and integrate it with the generated code.
5. Testing: Test the software to ensure that it meets the requirements and functions properly.
6. Deployment: Deploy the software in production environment.

Reverse Engineering:

1. Code Inspection: Inspect the code of an existing software system that you want to reverse engineer.
2. Code analysis: Analyze the code to understand its structure, design and functionality.
3. Mode Generation: Use a modelling too to generate a model of s/w system based on code analysis.
4. System Design: Use the model to understand system design.
5. Implementation: Write additional code and integrate it with existing code to modify or enhance the system.
6. Testing: Test modified system to ensure that it meets requirement and functions properly.
7. Deployment: Deploy modified system in production environment.

Conclusion: We successfully studied steps required for code engineering, forward and reverse engineering.

Experiment No - 09

Aim : To study testing of software.

Theory : Software testing is the process of evaluating & verifying that a software product or application what it is supposed to do. The benefits of testing include preventing bugs reducing development costs & improving performance.

Types of software testing :
There are different types of software tests. each with specific objectives and strategies.

i) Acceptance testing :- verifying whether the whole system work as intended.
ii) Integration testing : Ensuring that software components or functions operate together
iii) Unit testing : validating that each software unit performs as expected. A unit is the smallest testable component of an application
iv) functional testing : checking functions by emulating business scenarios based on functional requirements. Block box testing is a common way to verify functions.
v) performance Testing : Testing how the software performs under different workloads. loads testing for ex. is used to evaluate performance under

vi) Usability testing : Validating how well a customer can use a system or web application to complete a task.

In each case, validating base requirements in a critical assesment, just an important explonday testing helps a tester or testing team uncover hard to predict scenarios & situations that can lead to software errors.

Even a simple application can be subject to a large number & variety of tests. A test management plan helps to prioritize which type of testing provide the most value given available time and resources. Testing effectiveness is optimized by running the lowest number of defects

Conclusion : In this way we have studied software testing.