APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY SECOND SEMESTER MCA (LATERAL ENTRY) DEGREE EXAMINATION APRIL/MAY 2017

RLMCA 202: APPLICATION DEVELOPMENT&MAINTENENCE

PART A

Answer all questions. Each question carries 3 marks

1. **Briefly explain the need for Version Control**.

   Version control systems, also known as source control, source code management systems, or revision control systems.

   Need for version control

   - To implement continuous integration, for effective configuration management, managing different versions for the software
   - For keeping multiple versions of your files, so that when you modify a file you can still access the previous revisions.
   - Through which people involved in software delivery collaborate.
   - Developers should use it not only for source code, but also for tests, database scripts, build and deployment scripts, documentation, libraries and configuration files for your application, your compiler and collection of tools, and so on—so that a new member of yourteam can start working from scratch.

   (Points like this or similar to this can be given marks Each point carry's 1 mark) (Ref Text: Continuous Delivery Jez Humblechapter 2)

2. **Explain the ways to fix commits in Git.**

   Git tracks changes to your repository through commits, which we make with the git commit command.

   prompt> git commit -m "Some message"

   - git commit --amend is the way to fix the most recent commit. git revert, git rebase –i(1 mark for each utility's explanation & example)

   (Explanation with these commands Ref Text: Pragmatic guide to git - pg 110,111)

3. **Explain in detail about conflict handling in Git. (Ref Text: Pragmatic guide to git - pg 82,83)**

You can find the conflict inside the file by looking for the special markup that Git adds, <<<<<<< followed by a commit .commit a fixed conflict

- Git conflict handling –explain the cause for merge conflicts and its identification with conflict markers in the conflict file.(3 marks)

4. **What are the best practices for an effective commit stage?**
   - Compile the code (if necessary).• Run a set of commit tests.• Create binaries for use by later stages.
   - Perform analysis of the code to check its health.• Prepare artifacts, such as test databases, for use by later stages.
   - Provide Fast, Useful Feedback
   - What Should Break the Commit Stage?

- Tend the Commit Stage Carefully
- Give Developers Ownership
- Use a Build Master for Very Large Teams

(Ref Text: Continuous Delivery Jez Humble)

(Brief explanation of these points)

5. **What is a test double? Explain its importance in acceptance tests**
    - Replacing part of a system at run time with a simulated version.A test double is an object that can stand in for a real object in a test.The most common types of test doubles are dummy objects,stubs, mocks, and fakes.
    - Acceptance testing relies on the ability to execute automated tests in a production like environment
    - However, a vital property of such a test environment is that it is able to successfully support automated testing
    - Automated acceptance testing is not the same as user acceptance testing. One of the differences is that automated acceptance tests should not run in an environment that includes integration to all external systems
    - Instead, your acceptance testing should be focused on providing a controllable environment in which the system under test can be run

    Test double, definition , uses , interfacing with external systems diagram(1+1+1)

    (**Continuous delivery,chapter 8 page no 210 . Diagram in page no 211** - interfacing with external systems)

6. **Why is it necessary to test for non-functional requirements?**

    Managing NFR s, Addressing capacity problems (1.5+1.5)
    - Non-functional requirements such as availability,capacity, security, and maintainability are importantand valuable as functional ones, and they are essential to the functioning of the system.
    - NFRs are a frequent source of project risk.
    - Discovering late in the delivery process that an application is not fit for purpose because of afundamental security or poor performance is all too frequent—and may cause projects to be late or even to get cancelled.
    - Scalability testing,Longevity testing,Throughput testing, Load testing.

    (Ref Text: Continuous Delivery Jez Humble)

7. **Define orthogonality and its importance in pragmatic projects**

    Two or more things are orthogonal if changes in one do not affect anyof the others.

    (1 mark for definition)

    Importance/ benefits(2 mark for importance) (Ref Text The pragmatic programmer from Journeymen to master)

    - **Changes are localized**, so development time and testing time are reduced.
    - An orthogonal approach also **promotes reuse**. If components have specific, well-defined responsibilities, they can be combined with new components.

- **Reduce Risk** - A Diseased sections of code are isolated. If a module is sick, it is less likely to spread the symptoms around the rest of the system
    - Less fragile
    - Better tested
    - Isolated interface

8. What are the best practices that help to deal user requirements?

- Digging for RequirementsDon't Gather Requirements—Dig for Them
- Documenting Requirements
- ✓ Use caseDiagrams
- Overspecifying- Good requirements documents remain abstract.
- Just One More Wafer-Thin Mint…The key to managing growth of requirements is to pointout each new feature's impact on the schedule to theproject sponsors.
- Maintain a Glossary-Create and maintain a project glossary—one placethat defines all the specific terms and vocabularyused in a project.

(Brief Explanation with the above 3 or 4 points. Each point carry's 1 ,mark)(Ref Text The pragmatic programmer from Journeymen to master,chapter7)

PART B Each question carries 6 marks.

9. a. Explain the principles for effective software delivery.

- Create a Repeatable, Reliable Process for Releasing Software
- Automate Almost Everything
- Keep Everything in Version Control
- If It Hurts, Do It More Frequently, and Bring the Pain Forward
- Build Quality In
- Done Means Released
- Everybody Is Responsible for the Delivery Process
- Continuous Improvement

  Elaborate on the principles (6 marks) (Ref Text: Continuous Delivery Jez Humble)

OR b. Explain the strategy for implementing Continuous integration.

Continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day.
The main aim of CI is to prevent integration problems.Integration tests are usually run automatically on a CI serverwhen it detects a new commit.

- Check In Regularly
- Benefits of check-in regularly
- Create a Comprehensive Automated Test Suite
- Keep the Build and Test Process Short
- The first thing to consider is making your tests run faster.
- Managing Your Development Workspace

(Explanation with these points)**(Ref Text: Continuous Delivery Jez Humble)**

10. a**. Explain the utilities in Git that helps in effective repository management.**

Git is a distributed version control and sourcecode management system.

- **Configuring Gi**t

git config --global user.name "user_name“

git config --global user.email "email_id“

- **Creating a local repository**-git init Mytest
- **Adding repository files to an index**-git add README

git add sample.c

- **Committing changes made to the index**-git commit -m "some_message"
- **Connect to the repository on GitHub**-git remote add originhttps://github.com/user_name/Mytest.git
- git push origin master
- git branch and merge

(Explanation with these points and commands) (Ref Text: Pragmatic guide to git)

OR

b. Explain the importance of Merge & Rebase in Git.

- We have to merge changes from anotherbranch into your current branch in order to beable to use them.
- The simplest way to do this is through gitMerge
- Merging Commits Between Branches
- git merge <branchname>
- git merge new
- Merge changes from new branch to the master - git merge new

(Explanation with these points 3 marks)

Rewriting History by Rebasing(3 marks)

- Rebase takes a series of commits(normally a branch)and replays them on top of another

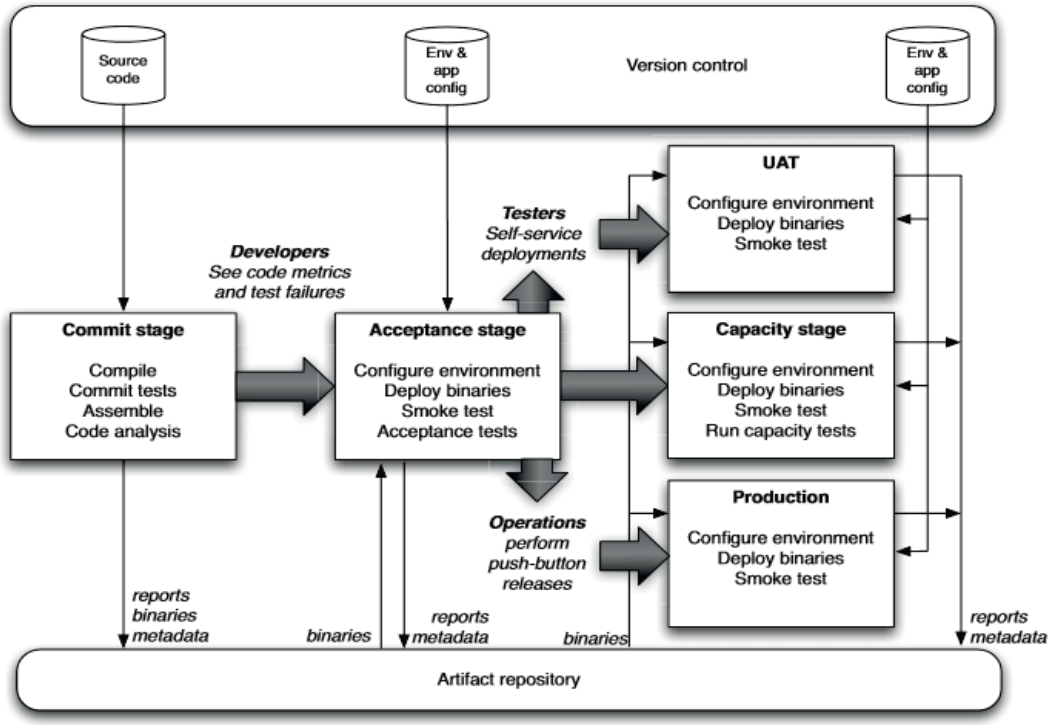
commit(normally the last commit in another branch).

- prompt> git rebase master

(Ref Text: Pragmatic guide to git)

11. a. Explain the stages in a Deployment Pipeline.

(Figure 1.marks)(Ref Text: Continuous Delivery Jez Humble)



- **The commit stage** asserts that the system works at the technical level. It compiles, passes a suite of (primarily unit-level) automated tests, and runs code analysis.
- **Automated acceptance test** stages assert that the system works at the functional and nonfunctional level, that behaviorally it meets the needs of its users and the specifications of the customer
- **Manual test stages** assert that the system is usable and fulfills its requirements, detect any defects not caught by automated tests, and verify that it provides value to its users. These stages might typically include exploratory testing environments, integration environments, and UAT (user acceptance testing).
- **Release stage** delivers the system to users, either as packaged software or by deploying it into a production or staging environment (a staging environment is a testing environment identical to the production environment).

  (The above 4 stages-5 marks)

  OR

b. Explain the importance of scripting in deployment phase

- **Commit script** - containing all the targets required to compile your application, package it, run the commit test suite, and perform static analysis of the code.
- **Functional acceptance test script** that calls your deployment tool to deploy the application to the appropriate environment, then prepares any data, and finally runs the acceptance tests.
- **Nonfunctional tests scripts** that run non functional test such as stress tests or security tests.

Principles and Practices of Build and Deployment Scripting

- ***Create a Script for Each Stage in Your Deployment Pipeline***

- Use an Appropriate Technology to Deploy Your Application
- *Use the Same Scripts to Deploy to Every Environment*
- ***Use Your Operating System's Packaging Tools***
- ***Ensure the Deployment Process Is Idempotent***
- ***Evolve Your Deployment System Incrementally***

12. a. Explain the factors the guide effective software release.

(Points in release strategyrelease plan release products(2+2+2)

Ref Text: Continuous Delivery Jez Humble)

Release strategy

- Stakeholders meet up during the project planning process.
- The point of their discussions should be working out a common understanding concerning the deployment and maintenance of the application throughout its lifecycle
- When creating the first version of your release strategy at the beginning of the project, you should consider including the following:
- Parties in charge of deployments to each environment, as well as in charge of the release.
- Configuration management strategy.
- A description of the technology used for deployment. This should be agreed upon by both the operations and development teams.
- A plan for implementing the deployment pipeline.

Release plan

- The first release is usually the one that carries the highest risk; it needs careful planning.
- The steps required to deploy the application for the first time
- How to smoke-test the application and any services it uses as part of the deployment process
- The steps required to back out the deployment should it go wrong

Releasing Products

- Pricing model
- Licensing strategy
- Copyright issues around third-party technologies used
- Packaging
- Marketing materials—print, web-based, podcasts, blogs, press releases, conferences, etc.

OR

b. Explain the steps to improve the efficiency of acceptance test stage

The acceptance test gate is an extremely important threshold andmust be treated as such if your development process is tocontinue smoothly.

Refactor common tasks, use compute grid , parallel testing & share expensive resources(explanation + diagram**)**

Refactor Common Tasks

- The obvious first step is to look for quick wins by keeping a list of the slowest tests and regularly spending a little time on them to find ways to make them more efficient. This is precisely the same strategy that we advised for managing unit tests.
- One step up from this is to look for common patterns, particularly in the test setup. In general, by their nature, acceptance tests are much more stateful than unit tests. Since we recommend that you take an end-to-end approach to accep- tance tests and minimize shared state, this implies that each acceptance test should set up its own start conditions. Frequently, specific steps in such test setup are the same across many tests, so it is worth spending some extra time on ensuring that these steps are efficient.

Share Expensive Resources

- We have already described some techniques to achieve a suitable starting state for tests in commit stage testing in earlier chapters. These techniques can be adapted to acceptance testing, but the black box nature of acceptance tests rules some options out.
- The straightforward approach to this problem is to create a standard blank instance of the application at the start of the test and discard it at the end. The test is then wholly responsible for populating this instance with any start data it needs. This is simple and very reliable, having the valuable property that each test is starting from a known, completely reproducible starting point.

Parallel Testing

- When the isolation of your acceptance tests is good, another possibility to speed things up presents itself: running the tests in parallel. For multiuser, server-based systems this is an obvious step. If you can divide your tests so that there is no risk of interaction between them, then running the tests in parallel against a single instance of the system will provide a significant decrease in the duration of your acceptance test stage overall.

Using Compute Grids

- For systems that are not multiuser, for tests that are expensive in their own right, or for tests where it is important to simulate many concurrent users, the use of compute grids is of enormous benefit. When combined with the use of virtual servers, this approach becomes exceedingly flexible and scalable. At the limit, you could allocate each test its own host, so the acceptance test suite would only take as long as its slowest test.

(Ref Text: Continuous Delivery Jez Humblechapter 8 ,pages 218,219,220)

13. a. Explain the pragmatic steps in maintaining a good knowledge portfolio.

Building Your Portfolio

- ✓ Invest regularly
- ✓ Diversify
- ✓ Manage risk.
- ✓ Buy low, sell high
- ✓ Review and rebalance

Goals

- ✓ Learn at least one new language every year.
- ✓ Read a technical book each quarter
- ✓ Read nontechnical books, too.
- ✓ Take classes.

- ✓ Participate in local user groups.

Opportunities for Learning
Critical Thinking
(Points like this can be given marks (Ref Text The pragmatic programmer from Journeymen to master)

OR b.

Explain the basic tools used by pragmatic programmers with their usage strategies.

- The Power of Plain Text
- Shell Games
- Power Editing
- Source Code Control
- Debugging
- Text Manipulation
- Code Generators

(**Ref Text The pragmatic programmer from Journeymen to master)**

14. a. Explain the importance of concurrency in design with examples.

- Always Design for Concurrency
- Design by Contract,
- Cleaner Interfaces**(Explanation of these points Ref Text The pragmatic programmer from Journeymen to master – Bend or break/Design for concurrency)**

OR

b. Explain the importance of refactoring in pragmatic projects.

- **Refactoring**

Rewriting, reworking, and re-architecting code is collectively known asrefactoring**.**
When Should You Refactor?

- **Duplication**. You've discovered a violation of the DRY principle (TheEvils of Duplication).
- **Nonorthogonal design**. You've discovered some code or design thatcould be made more orthogonal (Orthogonality).
- **Outdated knowledge**. Things change, requirements drift, and yourknowledge of the problem increases. Code needs to keep up.
- **Performance**. You need to move functionality from one area of thesystem to another to improve performance.

Benefits , when and how to refactor(2+2+2)

(Ref Text The pragmatic programmer from Journeymen to master)