Case study for implementing complete CI builds job automation with TDD and Refactoring process

The following is the specification of the User web Application case study that is to be implemented as a continuous deployment pipeline with developers and other team members working on it in a continuous manner.

The user stories to be implemented are as follows.

1. As a user I should be able to register on the site by submitting my user id, password and email id.
2. Once successfully registered I should be able to login to the application.
3. Once successfully logged in I should be able to get the info page in the browser.
4. In case I forget my login password, I should be able to retrieve it from the site by submitting my mail id.
5. The design document specifies using service and Dao classes in the web application for data access and other purpose. Servlets/Beans will be using service and DAO classes.
6. Start implementing in team (at least 2) environment. One working on Services and DAO classes while other works on domain servlets.
7. Use mocks to test the domain servlets in absence of real service and DAO classes.

8. **Development setup**: JDK1.8,Eclipse-Jee-Photon, Mockito and JUnit

9. **Steps for implementation**
    1. Create a new GIT repository for this project. Add project structure.
    2. Define the test classes in test package with test methods to exercise the tests for domain functional behaviour.
    3. Define domain classes in com.server package.
    4. Create build job in Bamboo server and configure the build trigger.
    5. Start adding tests and code into the repository and it will get automatically built in CI.
    6. Configure the deployment pipeline in bamboo with deployment project on tomcat and triggered by the build activity of the project plan.
    7. Run the tests and add required domain code for the tests to succeed
    8. Repeat the steps

10. **During the TDD cycle**
    1. Define the test
    2. Run the test.
    3. Let it fail first time.The Red bar will be shown
    4. Apply the required minimum implementation code to make the test succeed i.e. Green bar
    5. Fake the implementation
    6. Write obvious code for implementation
    7. Apply generalized approach when the code becomes duplicate and dependent(i.e. Triangulation)
    8. Run the test to see the green bar.
    9. Run all the tests to check the success of all the tests.
    10. Add more test methods to test more behaviour of account
    11. Refactor the code to avoid duplicate code and future updations easier.
    12. Run all the tests again to verify the success of refactoring.

Apply Code Refactoring

1. Remove the common duplicate code across different types of account and abstract it into a top level abstraction of Account.
2. Run all the tests and make all of them succeed.

******