Infosys®

Building Tomorrow's Enterprise

Infosys Global Agile Developer Certification

Automation

Study Material

www.infosys.com

Revision History

Version	Date	Remarks
1.0	Dec’2014	Initial Version

Table of Contents

Table of figure

1 Introduction

Automation in IT can be defined as set of programs/steps which can be performed independently to achieve required function and leads in reduction of manual effort and improvement in code quality. Automating the repetitive procedures can provide real value to the projects which are getting executed using Agile methodology.

Automation is a critical component to maintaining agility, and is a priority for the entire team through established practices/disciplines and a focus on continuous improvement. Continuous integration/builds, unit, functional & integration test execution and continuous/automated deployment are common examples of applying automation.

By reducing manual effort, automation in software design and testing eliminates risks associated with human error. It spares developers and testers from repetitive activities and allows them to focus on the more critical aspects of system quality, thus improving build and testing effectiveness as well as operational efficiency. Build and testing automation also ensures reusability and scalability, which in turn translates into significant cost savings. Some of the area where automation can play an important role are:

- Build and deployment of the system under design
- Code deployment
- Unit test execution
- Code coverage report generation
- Functional & Performance test execution
- Code quality metrics report generation
- Coding conventions report generation

The above list is obviously not exhaustive, and every project has its own unique characteristics. As a rule of thumb to help identify any process that should be considered for automation, consider automating processes that are expected to be repeated frequently throughout a system's life cycle. The more often the procedures will be repeated, the higher the value of automating them.

After the process has been identified, the Team is recommended to investigate on how to automate the process and identify the tools that could assist with automation. Also, the Team need to estimate the level of effort required to implement the automation against the total cost and risk of manual procedures. In most of the cases, decision boils down to cost versus benefit analysis. This module covers overall concepts of Automation and the areas where Automation can be implemented along with the associated tools in an Agile project.

2 Concepts

One of the key principles and benefits of following Agile methodology is faster feedback. This feedback includes the feedback from the Clients, Product Owner and Delivery team members. Often the term “Fail Fast” is used to emphasis that it is better to fail earlier so that impediments can be corrected based on the feedback.

Agile projects usually will have shorter release cycles in order to get early feedback from client. Though the functionality delivered is less, it is more important to deliver the right functionality. From the delivery view, it is important that the team (developers and testers) should be able to deliver faster. It is imperative that any repetitive manual tasks should be automated. This will free up the time spent by team members on these repetitive tasks and focuses on delivering more value (might be another story) to the users.

The complexity of a typical enterprise application development and deployment process involving multiple branches and multiple projects adds a huge cost overhead and risks to the organization. Some of the benefits of using automation in these cases are listed below -

Typical Manual process	Automated process
A tight control is needed on individual components for monitoring the overall process	Transparent process as the progress on all components are more visible and risks can be identified more easily
Higher repetitive efforts during deployment	Reduced efforts during deployment
Risk of manual execution error is high	Once standardized, the errors are avoided
Could run in to quality issues	Once standardized, high level of quality is maintained
Inconsistent results as the process is dependent on the persons involved	Standard results always.

Some typical real world scenarios -

Scenario 1 – Developer is supposed to make changes to an existing functionality. After completing development, developer must test the changes made. Now, if there are no existing unit test cases, developer must be performing unit testing for the entire functionality again. This will in-turn increase the time taken for the changes. If the test cases are automated through various unit testing frameworks using xUnit, the test cases can be run quickly and validated immediately. Developer can focus only on testing the changes and creating unit test cases around these changes.

Scenario 2 – After completing unit and integration testing, developer must promote the changes to QA/testing environment for further testing by the QA team. If the promotion to QA environment involves multiple requests to infrastructure team and few hours delay, this will create bottleneck. It will be an impediment to the overall team, as there could be multiple stories that need to be promoted to the QA environment.

3 Usage of Automation in Agile

3.1 Automation in Development

Unit Test suite – Unit Testing is one of the basic building blocks for Continuous Integration. Developers can have series of their testing automated as simple unit tests and make it available along with their versioned codebase. These tests should be living test suites i.e. it should run to success at any point of time. Whenever a code module is changed, the corresponding unit tests should also be updated and ensured that the test cases are passed. Automated Unit Tests are the first line of defense and the key step in "Fail Fast" approach.

Various unit testing frameworks that are available are – *JUnit* (for Java), *NUnit* (for DotNet), *utPlSql* (for Oracle PL/SQL).

Code Quality Analysis – Static code analysis needs to be performed to identify and quantify the code quality metrics like standards, security risks, and adherence to best practices. As part of the development practice, the code quality checks should be run. Only after attaining satisfactory quality metrics, the code should be checked into the version control tool.

Some of the key tools for Static Code Analysis are *CAST, SONAR*. Note that developers can make use of the IDE(Integrated Development Environment) integration capabilities of these tools to integrate static code analysis in their development process.

3.2 Automation in Build

Continuous Integration (CI) – The aim of practicing Continuous Integration is to maintain a build environment which is able to generate "Production Ready" builds. As part of this practice, following actions will be performed -

- analyze the code for quality
- build the code
- execute unit test cases
- upload build artifacts
- publish the build and test results
- alert the team with results

This can be achieved effectively through automation. For further details on CI practice, please refer the study material on 'Continuous Integration and Associated Tools' of this course.

To aid in this practice following support systems are required -

Source Version Control tools – The version control tool should be able to determine and communicate to other tools on any new code changes through "hooking" (proactively let the other tools know when a new code changes are made) or "polling" (other tools have to poll the Version Control tool to determine whether a new code changes are made). Example – *Git, ClearCase, SVN*

Build tool – A robust tool/framework that helps in maintaining and code build more easily. Example – *Maven, ANT*.

Static code analysis tool – As described in above section, these tools are used to check the code quality. Example – *CAST, SONAR*.

Build repositories – A robust repository to maintain the builds that can be uploaded/downloaded/searched with ease. Example – *Nexus* (for maven builds)

CI server – Interacts with all the above components and perform the actual build of the application along with execution of tests either automatically or just by click of a button. Example – *Jenkins, Hudson*.

3.3 Automation in Deployment

Continuous Deployment (CD) – A mature Agile team extends the Continuous Integration practice so that the application can also be automatically and regularly deployed into multiple environments (realistically lower environments like DEV and QA). This is needed for maintaining a "Production Deployable" builds. Any deployment issues will also be validated, as part of this process. This practice is called Continuous Deployment.

A typical CD process includes –

- All Continuous Integration steps
- Deployment into needed environments
- Execution of smoke tests
- Publish the deployment results
- Alerting the team with results

Usually an enterprise application will follow multi-tier architecture which includes multiple technology (Example - J2EE application with Oracle backend). So, to achieve deployment automation in multi-tier architecture, there should be an automated orchestration.

For automating the deployments, the delivery team can have a custom scripts or use automation framework like *uDeploy*. The CI build servers like *Jenkins/Hudson* can be configured to perform the CD activities as well.

3.4 Automation in Testing

With the nature of the Agile methodology, the software that is developed iteratively will be subject to testing multiple times either as part of the system integration testing or regression testing. So it is imperative to automate test execution as much as possible. There are various testing automation frameworks that are available like *Cucumber, FitNesse, QTP* etc. that helps in writing and executing the test cases. It is recommended to create the automated test cases right from the beginning of Sprint execution in parallel with the User story development.

Once the test cases are automated, it can be scheduled as part of the build servers to be triggered automatically, say every day. This will ensure that the tests are run often and the failures are reported as early as possible. Following are the various stages that can be automated

Smoke Testing – To validate whether the deployment of the application is successful and basic features are working, few sanity test cases can be handpicked and a suite of tests can be created for smoke testing. These test suites will be executed from the build server after every deployment tasks. This will ensure that the basic features are validated automatically.

System Testing - As part of the regular system integration testing, the test cases are created in the needed automation framework. Though it might take some effort initially, it will ensure that there is no need for automating the tests in future. The Rule of Thumb should be whatever test cases that can be automated, should be automated. Only for areas where manual intervention is unavoidable, manual testing should be performed. This will ensure that the test case library is always updated. Also any future maintenance/iterative work on the same module will not need excessive testing effort.

Regression Testing – For any application, before moving the code for production, regression testing will be done to ensure that the new changes are not affecting other features. Typically testers will validate sanity checks on other modules for every release. Since in an Agile model, the releases are more frequent, the regression testing should also be repeated frequently. Now, for each release, more application features are also getting released, which in turn increases the scope of regression testing. So for keeping the regression testing effort to a minimum, automation of test cases is mandatory. Within the automated test cases, required regression test cases can be identified and run whenever required. Also for new features, since the test cases are getting automated simultaneously during the System Integration testing, the regression testing cases are also getting refreshed as part of each release. With the minimal regression testing effort, more time can be spent on building new features.

Acceptance Testing – Acceptance testing, as the name suggests, is about formally defining the behavior of a system and testing the system against the defined acceptance criteria. The requirement of the system itself can be written as multiple acceptance criteria/tests. Mature Agile teams prefer their requirements as these acceptance criteria and validate their system against these criteria as they develop the components. Note that the aim of defining these criteria is to automate them easily. Multiple frameworks/tools exists to support these acceptance testing including *Cucumber, FitNesse* etc. This method of defining the acceptance test cases initially and then developing the code based on it is known as Acceptance Test Driven Development (ATDD).

3.5 Emerging Trends

Today, automation is an exciting focus area for technology organizations and lot of technologies and tools are emerging in it.

- ***LiquiBase*** – This is a tool that tracks the changes that are made to the database. The changes to the database are maintained in an XML. This helps in subsequent Rollback of changes or applying a change in one environment to another. This can be used to create/rollback test data for unit test cases or to move the changes from DEV environment to higher environments.

- ***Chef*** – This is a tool that automates deployment and management of infrastructure. With this tool, Infrastructure is treated as versionable, testable, and repeatable similar to the application code.

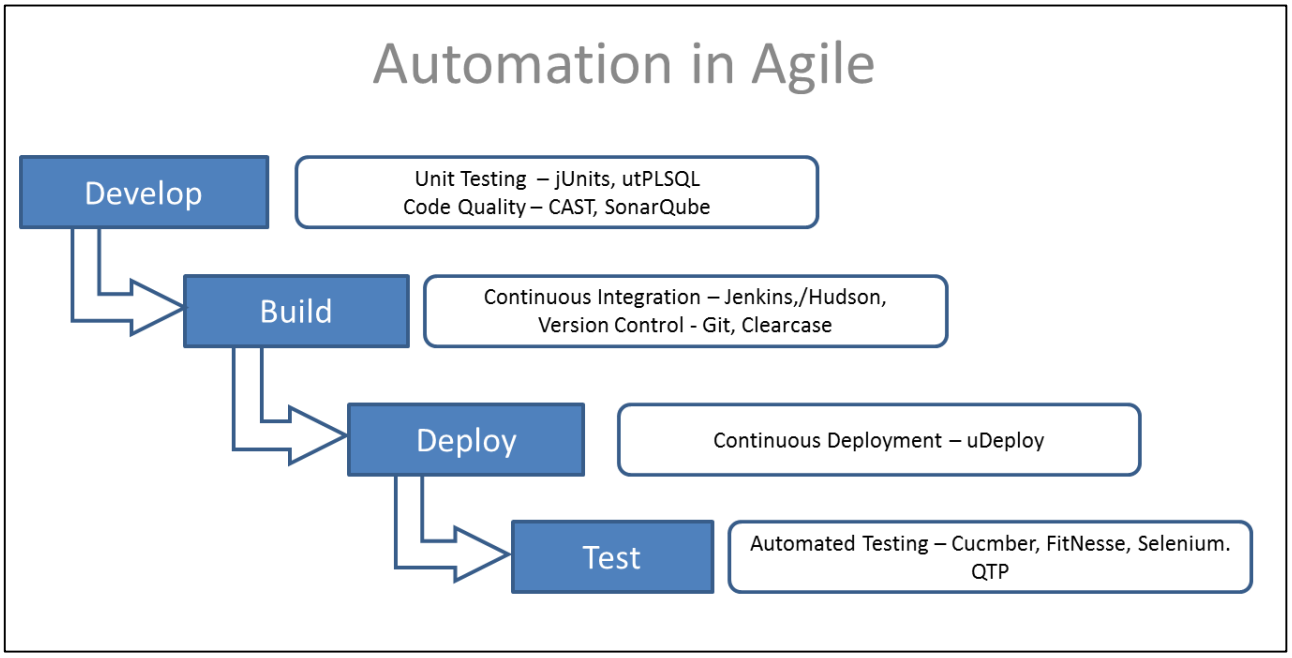

Figure 1: Representation of Usage of Automation in Agile using sample set of Tools

4 Challenges in Automation

There are many obstacles for implementing automation in Agile projects. These are:

- **Pressure from Product Owners:** Product Owners would like to get their working product out and may not be willing to let the team spend the time on automation frameworks that might benefit long term. All the stakeholders must be taken into confidence on the benefits of automation. Return on Investment (ROI) can be calculated on the adoption of automation and used for articulation.

- **Maintenance of automated scripts:** It is important that the automation scripts are well maintained along with the production code. Enough support to be provided to the team to ensure that the changes to the maintenance of the automation scripts are taken up along with business priority. As the total number of automated tests increase, the time taken for building and executing the tests tends to increase. Team should decide on the priority of the tests executed along with the build and ensure that needed refactoring of automation scripts is performed to improve, wherever possible.

- **Lack of follow through:** Though the delivery team adopts the best practices early in the development process, team might lose sight of the vision and may compromise on the automation activities. Delivery lead or any member of the team should constantly remind the team in case of any deviations.

- **Agile Maturity:** Development team might have the mental makeup of giving preference to the code than the automation scripts or frameworks. Team should be trained appropriately and instill the practice of creating automated tests along with development, for sustainable maintenance of the application.

- **Timing of Automation:** Automating the build process when the project is nearly over, provides very little benefit. It can, however, save dozens, if not hundreds of hours when automated as soon as development/ testing begins

5 Case Study

5.1 Case Study 1 – Deployment Automation

Scenario – As part of an application development in Agile, developers are responsible for performing deployment in multiple environments. The application consists of Java code components and Oracle Database code components. The application (app) is hosted in Tomcat web servers, for which developers will have access. Usually the app developers will manually copy the .ear file (the compiled version of Java code component) to the UNIX server. Later in the tomcat server, the existing .ear file will be deleted and the new .ear file will be copied. But before the server is re-started to reflect the change, app developer will coordinate with DB (database) developer to perform the database install in the corresponding environment. Once DB developer manually executes the script, the app developer will restart the tomcat server. The application code is hosted in Git repository. Jenkins is used as the build server to build the code base.

Problem - The above mentioned process requires coordination between app developers and DB developers. Since the Agile methodology is followed, multiple deployments in DIT environment are needed every day. How to efficiently deploy the application to avoid the escalating efforts and cost for deployment?

Solution approach – Automate as much as possible. Since the Unix servers are used, make use of the unix scripts to perform needed operation. Try to configure the order of installing the components.

Solution –

To perform the tomcat app deployments in UNIX, a customized tomcat admin toolkit was created. With the toolkit, one can perform following operations -

- Application install/uninstall
- Server Start/Stop/Restart

Similarly to perform DB install, a customized and comprehensive UNIX toolkit is created. This toolkit will perform the following –

- Copy the SQL files that need to executed from the code repository
- Connect to the database from UNIX
- Execute the SQL files in the needed order.

With the automation of these operations, the overall process of independent component deployment is now simplified.

Since Jenkins support execution of UNIX scripts, a new Deploy job is created in Jenkins. As part of this deploy job, a series of UNIX commands are executed to get the application and DB component from the last successful build and deploy simultaneously it in needed servers using the above mentioned toolkits. With this approach, the developers can deploy the components to the needed tomcat server and DB in the right order through a simple click of a button in Jenkins.

5.2 Case Study 2 - Code Generator

Scenario – As part of legacy modernization, the requirement was to load around 200+ feed files to the oracle database. Informatica was the preferred tool to load the file to the database. Project was to be executed using agile methodology.

Problem – Average development effort for each loader was around 4 days. There were constant requirement for change from customer and the developer has to spend lot of time in changing the Informatica mappings which was impacting the productivity of the project. Changing the Infomatica mapping was little tedious as the developer has to first understand the mapping and then make the respective changes. Also there were cases when developer induced new defects.

Solution approach – After brainstorming the issues, team came up with a solution of automating the process of generating Informatica mapping such that developer don't have to look into Informatica tool for any changes. Also, as the mapping were similar in nature, this kind of automation would have helped in increasing the productivity of the team by developing new load faster.

Solution –

The team developed an excel based tool where user can give the source and target mapping. UNIX scripts were developed to read the mapping from excel, generate the Infromatica workflow (XML files) and export it to Informatica server. The scripts were also developed to generate the supporting configuration file and install them in respective development server.

After the tool was developed, any changes from the business were easily accepted and changed in quick time. Team also used the tool to develop new loaders which helped in increasing the productivity and hence the velocity of the team

6 Practice Questions

Question 1

Where can automation be applied in an agile project?

a. Only in development
b. Only in testing
c. Only in deployment
d. All of the above

Question 2

Which of the following is NOT an advantage of automation in a project?

a. Reduces manual effort
b. Helps in frequent releases
c. Frees team time so that they can focus on other features
d. None of the above

Question 3

Ram is working as a QA analyst in a research project. Client has already communicated that there would be frequent enhancements to the ratings screen. To avoid testing the old features again and again, what should Ram do?

a. Automate the testing scenarios at the end of the project
b. Test all the feature manually whenever the enhancement request is sent
c. Automate the testing scenarios and keep enhancing whenever the new request is sent
d. None of the above

Question 4

JUnit is a tool used for:

a. Unit testing for Java component
b. To check the code quality
c. Unit testing for database components
d. JUnit is a UI development framework

Question 5

Which of the following can be used for code quality?

a. SONAR
b. Clear case
c. XUnit
d. QTP

7 References

- An excellent guide to Agile Practices - http://guide.agilealliance.org/
- Software Practices Automation - http://www.developerdotstar.com/mag/articles/automate_software_process.html
- Jenkins - http://jenkins-ci.org/
- SONAR – http://www.sonarqube.org/
- CAST - http://www.castsoftware.com/
- LiquiBase - http://www.liquibase.org/
- Chef - https://www.getchef.com/chef/
- Informatica - http://www.informatica.com/in/

www.infosys.com