NAVIGATE YOUR NEXT

Infosys Global Agile Developer Certification Refresher Part – 2

Infosys®
Navigate your next

Sections Covered (Agile Generic)

Following study contents cover the basic concepts of Agile, Planning and Estimation techniques in Agile and Soft Skills which are key for Agile project execution.

Study Content for Generic Modules in Agile		
Overview	Planning & Estimation	Soft Skills

Technical aspects play an important role for the disciplined delivery in Agile and maintain the quality of the output. Following study contents provide the key engineering practices which the Team must be conversant with while getting started in the Agile project.

Study Content for Engineering Modules in Agile		
Requirements, Architecture & Design	Test Driven Development	Refactoring
Automation	Continuous Integration and Tools	Testing Techniques

Content from study material covered in the ppt

Need to be read from the study material in Learning Hub. No content covered.

Requirements, Architecture and Design

Requirement Categorization: Theme, Epic, User Story

The widely-used technique for classifying the requirement is size-based SMC (Small, Medium and Complex) or based on various factors like dependency, unknowns, risk and uncertainty

In Agile, requirements are primarily defined based on value to business (ROI-Return on Investment) and how they evolve over the project execution timespan

Theme:

- A Theme is very large requirement and sometimes encompass entire product vision
- Set of interrelated requirements which can be considered for Release planning
- Large theme can be subdivided in multiple sub-themes
- Development of a theme can span from few months to a year

Example: 'Xyz Bank wants to allow customers transact online and be able to do fund transfer, open fixed deposits, and request check book online.'

Epic:

- Very large requirement that can't be completed in a short iteration (Sprints)
- Needs to be broken down into User stories
- Can be considered as a small theme by virtue of its magnitude

Example of Epic: 'Xyz Bank wants to allow funds transfer between own accounts in different branch, other accounts in same bank, and preferably, accounts in other banks.'

User Story:

- A feature small enough to be delivered in iteration and represents unique business value
- Epic is divided into multiple user stories
- Mostly a vertical slice cutting across all phases of Software development life cycle and has tangible business value.
- Represents technical debt which doesn't represent business value directly but is required for better and maintainable code quality

Example of Epic: As a customer, I want to schedule automatic fund transfer among my accounts.'

Effective Requirement Definition

Requirement Workshop

- Carried out from a few hours to a few days depending on size of the release
- Outcome of these workshops is Epics and Stories on which stakeholders have enough clarity
- Have stakeholders from even extended teams such as Tech Architect, Solution Designer, IT Managers, Capacity Managers and BAs
- Requirements are prioritized, detailed out, estimated

Requirement Accuracy

- Requirements taken up for development should have enough clarity to help team estimate them to a satisfactory and agreeable level
- Complex Epic requirement should be broken down to sub-epics till it is possible to define multiple short feature/story for each one
- Becomes accurate when it's smaller as it will have lesser unknowns, dependencies and associated risks
- Adding testable acceptance criteria in each requirement helps in a to increase requirement clarity

User Story

- written in perspective of user and how it would help business
- always capture AT (Acceptance Test) which should be testable
- Each story should have DONE criterion to consider the story for acceptance
    - Story is developed and Unit-tested
    - Peer reviewed and passes all unit tests
    - QA complete and no defect open
    - Acceptance Criterion satisfied

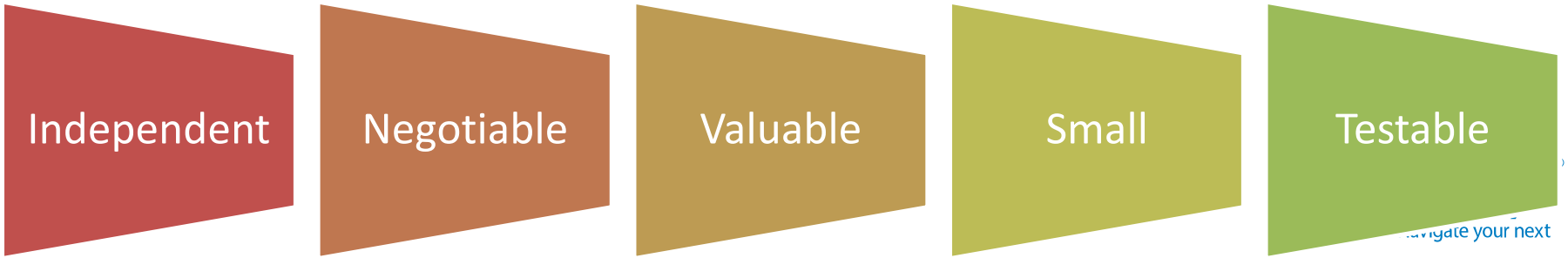
User Story Id: UX-14 **Requirement Id:**	**Description: Search feature in Control Panel**
As a (User Role) Control Panel user **I want to:** Search all the values in the Control Panel based on Product code/ Channel Code/Location/ Stability Profile combination **So that :** I can filter the different values for further updates	**References:** Control Panel Page Acceptance criteria/how to verify story completion: Verification test to be performed from Control Panel Page Verify that the correct values are displayed based on the filter criteria applied during search

User Story

Three C's for writing good user stories

Card	•'Card' came from XP world where practice of writing requirement on Index cards •Writing requirements in short
Conversations	•brings required details •conversation summary must be recorded on the backside of the card with date
Confirmations	•Acceptance tests provide ways to confirm if story card needs are met •Can prove as excellent artifacts to be considered as part of documentation deliverables

INVEST Method: Define User Story



Prioritization Techniques

Cont…

Prioritizing By Value:	Value stream visualization, cost to build the feature vs. efforts, feasibility, infrastructure cost (if any), risk, skills, segment/audience
	Include requirements are essentially to improve operations or for statutory compliance reasons and do not add business value; rather they are cost realization
MMF (Minimum Marketable Feature)	Focused on the set of features that needs to be clubbed and released together to gain business value
	Team focuses on completing smallest functionality of real importance to business rather than releasing a theme or epic which may not directly lead to delivering most important need
MoSCow	Must Have
	Should Have
	Could Have
	Wont Have
Cost od Delay (COD	Impact of not completing a user story or feature to business
	Used with Kanban method to effectively prioritize requirements that have more value over others because of cost of non-completion
Certainty	Certain requirement has more weightage while prioritizing
	Helps in fast-tracking the decision process of business stakeholders
Feasibility	Can be gauged based on the cost/benefit analysis (CBA)
	No matter which technique is used, list of project requirements must be sorted from most to least valuable

Architecture and Design

- High-level architecture generally involves
  - Depicting the system into modules
  - Representing the interfaces and interactions between these interfaces
- Detail design outlines the design internals for each of the modules
- For OOAD (Object Oriented Analysis and Design) project high-level design includes system architecture with details about interfaces, components, and networks, data flows and interactions between component systems
- Detail design further divides high-level design into class diagram with all the methods and relation between the classes

Factors for Bad Design

Factors	Description
Rigidity	The design is difficult to change
Fragility	The design is easy to break
Immobility	The design is difficult to reuse
Viscosity	It is difficult to do the right thing
Needless complexity	Overdesign
Needless repetition	Mouse abuse
Opacity	Disorganized expression

Agile projects believe in You Aren't Gonna Need It (**YAGNI**) & No Big Design Up Front (**NBDUP**):

- YAGNI is an engineering principle of Extreme Programming (XP)
- Focuses on working on the features which are deemed necessary
- Used in combination with other XP practices of refactoring and TDD

Rationale:

1. Any code or design which is not required for the current set of features might not be required in the future
2. More time is spent in designing/ developing feature which does not add any value to the current set of features required by the customer
3. Additional unrequired feature can lead to software becoming larger and complicated which in turn can impose constraints on adding new features

Define System Architecture

- Create visual architecture and infrastructure (AIP) planning diagram defining components, interfaces and other characteristics of system or components
- Security or other NFRs in the visual AIP
- Decided in workshop or project initiation (Sprint 0) phase

Guidelines:

- Create System architecture diagram for implementation in different layers and the main components
- Decide on platform level details (programming language, database)
- Identify risk areas that need POC (Proof of Concept)

Techniques:

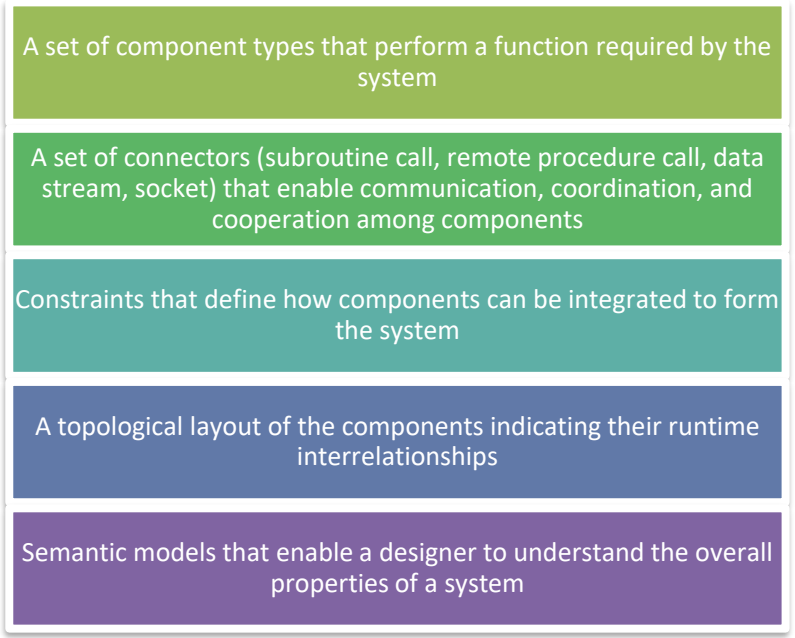
- Expert Judgment
- Whiteboard, AIP diagram
- Existing Enterprise Guidelines & Constraints

Define Architecture Stories

- Software Architecture is the structure which comprise the software components, the externally visible properties of those components, and the relationships among the components
- Represents the structure of the data and program components that are required to build a computer-based system
- Refined at the initiation of the project

- Risk areas are identified as part of the 'Define System Architecture' phase (i.e. Sprint-0 or Discovery Phase). These are defined as architecture stories and taken up as POC (Proof of Concept) during this phase, which is in-line with the idea of fail fast of Agile principle
- Any other POC as required for identified risk areas or new technology implementation should be covered during this phase and has to be in agreement with all stakeholders
- Some of the high priority stories can be identified for implementation, during first 2-3 Sprints (assuming 2 weeks iteration for project duration of more than 6 months) which can set up the technical foundation (e.g. Data Access Framework, Error Logging Framework, common utilities) for the system being developed

Architectural styles describes a system category that encompasses



Objectives:

- Reduce project risk due to technology by validating assumptions
- Provide foundation for future Sprints

Techniques:

- Expert Judgment
- Whiteboard
- Team Discussions

Just-in-Time Design

- Focus on feature implementation and design should be given at start of story
- Design evolves iteratively along the Sprints
- Sets the direction for the team for implementing the stories in conjunction to the defined architecture
- TDD to be followed to ensure easier code refactoring
- Pair programming for code

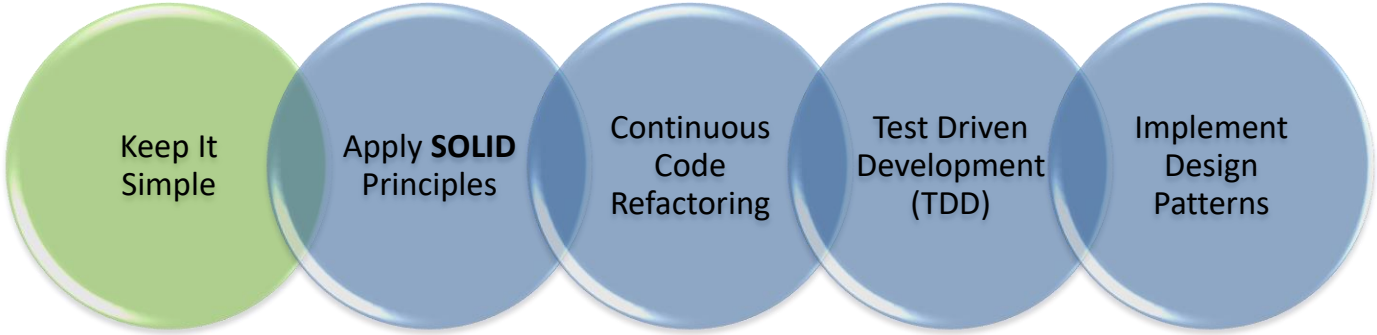
Visual artifacts for current story:

- Interaction Diagram
- ER (Entity Relationship) diagram
- Sequence Diagram

Techniques:

- Expert Judgment
- Whiteboard

Design Principles



Self-Explanatory:

- Code should be self-documenting
- Should use unabbreviated words i.e. full names
- Standard naming convention should be followed
- Use comments or assertions for better understanding of code

No duplication:

- Each and every declaration of the behavior should appear OnceAndOnlyOnce
- Developers should not implement unrelated ideas in the same method
- Code without duplication is easier to maintain and change

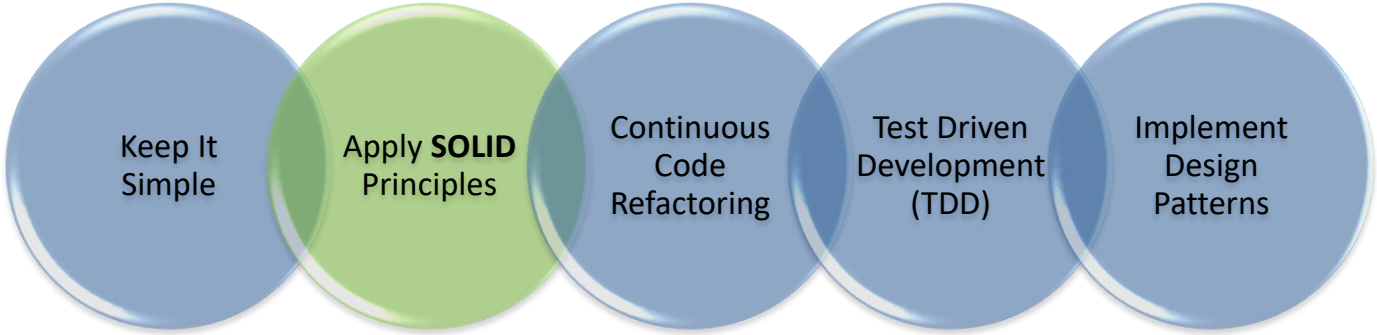
Remove superfluous content:

- Code should not have any unnecessary content
- Support the required functionality
- Any functionality which is not required should not be coded (YAGNI principle)

Cover all Test Scenarios:

- Write code that satisfies unit tests covering all functional areas
- TDD helps the developer with the same
- Minimal coding is done to keep it simple and flexible to maintain

Design Principles

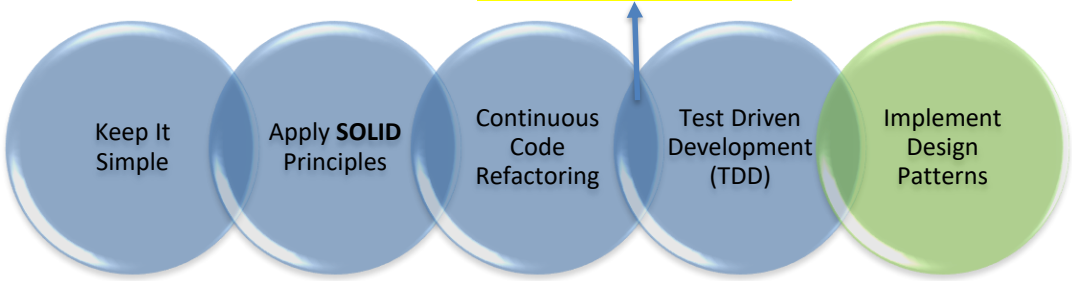


- **Single Responsibility Principle**: The programmer should maximize cohesion so that each class does only one thing. This in turn ensures that behavior of a class is not accidently changed
- **Open/Closed Principle** :The open/closed principle states "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification"; that is, such an entity can allow its behavior to be modified without altering its source code
- **Liskov Substitution Principle**: If a system is using a base class, the reference to that base class can be replaced with the derived class without changing the functionality of the system
- **Interface Segregation Principle**: Each interface should have a single responsibility and should not be overloaded. Interfaces should be kept small and cohesive and exposed only when required
- **Dependency Inversion Principle**: Idea is to isolate our class behind a boundary formed by the abstractions it depends upon. If all the details behind those abstractions change, then our class is still safe. This helps keep coupling low and makes our design easier to change

Please refer to page 11 to 16 of Requirements, Architecture and Design for code example

Design Principles

To be read from study material



- Experienced Object Oriented (OO) developers build up a collection of both general principles and idiomatic solutions that guide them in the creation of software
- These principles and idioms codified in a structured format describing the problem and solution are called as patterns
- Design Pattern names, abstracts, and identifies the key aspects of a common design structure that makes it useful for creating a reusable object-oriented design
- Describes a design structure that solves a particular design problem within a specific context and amid forces that may have an impact on the manner in which the pattern is applied and used

For example, here is a sample pattern:
Pattern Name: Information Expert
Problem: What is a basic principle by which responsibilities are assigned to objects?
Solution: Assign a responsibility to the class that has the information needed to fulfill it

A good pattern is a named and well-known problem/solution pair that can be applied in new contexts, with advice on how to apply it in novel situations and discussion of its trade-offs, implementations, variations and so forth. The intent of each design pattern is to provide a description that enables a designer to determine:

- Whether the pattern is applicable to the current work
- Whether the pattern can be reused
- Whether the pattern can serve as a guide for developing a similar, but functionally or structurally different pattern

Helps to

- Document design decisions and rationale
- Reuse wisdom and experience of master practitioners
- Reuse successful design
- Design alternatives
- Get right design faster

Agile Testing Techniques

Testing: Overview

- Tester’s involvement is necessary right from the initiation of the project so that they can actively participate in defining Definition of Done, exploration of User Stories
- Follow up on validation criteria in order to know what's expected of a new feature or a change being made in the system
- Both testing and development are carried out in the same sprint
- Tester’s work on how the system would behave after the new changes are incorporated and whether it satisfies the business needs or acceptance criteria
- When the development team works on the design and code changes, testing team works on the testing approach and comes up with the test scenarios and test scripts
- Testing team is expected to work on smaller modules and test them as and when provided rather than the whole functionality to be developed completely

Testing: Concepts

Test Strategy and Planning

- Structured and definite testing approach and strategy in the project
- Recommended to update the test strategy at the beginning of each sprint
- Documentation of this test strategy starts from the Release initiation phase and acts as a living document throughout the Release
- Test plan has to be updated in each Sprint and is continued till the release
- test one functionality at the same time write a test plan for another and at the same time review the design for another story and carry out many similar kinds of activities all at the same point of time
- Testers participate in design discussions in Agile projects
- In sprint review, testers prepare the test data sheets and run the tests in front of the Business Stakeholders, Product Owner and show them how the product is working and whether or not it's meeting their Acceptance Criteria

Test Coverage in Agile

- Inadequate test coverage might result in missing of critical tests for some requirements
- Test coverage for a user story is usually discussed and finalized during the backlog grooming sessions and later during the Sprint Planning meetings
- Creating a traceability matrix between requirements, user stories and test cases avoids missing out a few test scenarios. A link can also be created between test cases and user stories
- Analyzing the impact, review of source code to identify modules that are to be changed and ensuring that all the changed code has been properly tested ensures test coverage
- Use of code coverage tools must be leveraged for verifiable thoroughness as part of the automation effort

Testing: Concepts

Test Data Management in Agile

- Agile Testing in Sprints gets difficult when the system is dependent on another team for test data. Sprint timelines do not give enough room for any delays in the data setup
- Setting up the test data is done along the Sprint to make available them at least, ‘just in time’ (JIT) if not earlier, by the time when the developer checks-in his code for claiming it ‘as done’
- Developers usually practice test automation for unit testing and integration testing, use TDD approach to complete user stories, the synchronized supply of the test data by testing team is essential
- Analysis of the data needs is done during the backlog grooming sessions and the Sprint planning meetings to ensure the complete data set up needs
- If data is complex and huge, we may not get the adequate support from the team owning the data. Hence the discussion should ideally start with the business, at least a Sprint early or as per the applicable SLA to get the data at the appropriate time
- members (from business) who are responsible for the data should be included as part of the Scrum team for the Sprint where their support is required
- test data created in earlier sprints can either be directly reused or used with minor modifications in order to avoid redundant data requests/set up and reduced turnaround time for the test data creation and manipulation

Impact Analysis in Agile

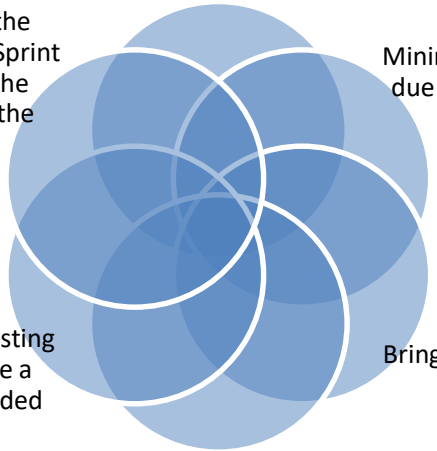
- Significant role of testing team in impact analysis and design reviews as they are involved in the design discussions happening for a story
- Helps the developer in the impact analysis and debugging the issues
- Testing team members go to the code level to analyze or debug the issue along with the developer

Importance of Automation for Testing in Agile

Importance of Automation for Testing in Agile

Agile needs continuous testing of the functionalities developed in past Sprints. Use of Automation helps complete this regression testing on time

Helps in minimizing the cost of the project in the Release Sprint (last Sprint in the Release) - as majority of the testing can be handled through the existing automations



Minimizes the rework that can happen due to error prone manual processes

Testers can put their effort into testing areas of new functionality where a manual intervention is really needed

Brings predictability to the tests run as part of Sprint testing

Helps to cover more regression tests and reduces effort than it could have been done manually because of higher number of builds and frequent changes

Testing: Types

Unit Testing	• Part of every Sprint for every story being developed • Allows developers identify as many problems as possible at an early stage • Doing it in a repeatable and automated fashion helps them make more frequent and stable code changes • JUnit, NUnit, Jmock etc. are some tools to effectively and efficiently test the code being developed
Integration Testing	• Iterative and continuous in nature • After few units or functions are developed and tested by the team, if integration of systems have to be done, then that story is planned for the sprint on priority and integration testing is performed by running tests that were done in isolation previously • A project team should have outlined the capacity for integration testing based on current scenario of the project to seize any opportunity for integration • Team should create stories and functions to integrate and plan tasks to carry out the sequence of the integration of user stories to complete integration testing in every sprint For example, imagine a new data feed has to be sent from System A and consumed by System B. There will be stories to create a feed from System A. There will be stories to consume the feed in System B. Now both these work happen in isolation. After these are tested by the team, there should be one more story to actually integrate the two systems for the new feed and test the same. This helps them to address any integration issues between the two systems. To summarize, in Agile it is not expected to wait till the end of development phase to carry out Integration testing.
Smoke Testing	• Done to confirm whatever is dropped to QA for testing is good and can move ahead with system testing • Subset of system test cases are used for targeting some of the critical features • Happens once in a release path prior to the start of system testing • Important when there are multiple release paths catering to different releases • It happens after every code backfill into the current test environment from the release path • Carried out after a code drop, by all team members irrespective of whether he is developing or testing • For better usage of time, since code drops happen very frequently in Agile, there should be a base-lined set of test scripts for Smoke testing in a particular sprint. This set of test scripts/cases should be distributed to all the team members irrespective of their work. This way when a code drop happens, the team members can complete the smoke testing quickly.

Testing: Types

System Testing	• Scenarios where system testing of a story needs a completion of another story, prioritization of stories keeping an eye to the end product and the testing needs is critical • Since it's iterative in nature, the same scripts have to be executed as and when different interlinked stories are developed • The system test scripts have to be designed in line with the Story under test and not the complete feature • Ensure that the scripts are designed in such a way that they require minimum modification when the end product/feature is ready for testing
Regression Testing	• Done prior to the release of new changes into production basically towards the end of system testing • Done in every Sprint to ensure that all the previous functionalities developed till the previous sprints are working fine • Having an automated suite of test cases is important to help the team with their regression testing. Unless there is automation, regression testing objective to cover all prior Sprint functionalities in a given sprint cannot be realized • For maintenance kind of projects, regression testing of all the existing functionalities is carried out in the last Sprint prior to the release (also known as the Hardening Sprint or Release Sprint) • Based on the volume of changes, team should plan to execute the existing regression suite in small portions in every sprint before the Release or Hardening sprint. This way, the team would avoid maximum issues in the Release or Hardening sprint and minimize the risk to the Release
Production Implementation Testing	• When the product is released to production or to the end users of the product, the development and the testing team is involved in the implementation • Testing team prepares a checkout test plan and list the cases to be tested once the code is implemented in production • During this phase, if the team finds something which is not working as expected, they fix it immediately, thereby by ensuring quality of the product • In the last sprint prior to the release, the capacity for the team should have factored in the activities that has to be done for the implementation starting from deployment to testing the changes in production and carrying out the required preparedness for the same

Testing: Types

Performance Testing	• Determine the behavior of the system at a particular load and the breakpoint of the system • Done at the release sprint after the functional testing is completed • However, if the performance is a critical aspect of the project and there is high risk involved when performance issue identified may lead to significant changes in the application, Team is advised to run performance testing after few iterations • Team needs to plan and work on the performance testing couple of Sprints prior to the Sprint where performance testing is planned. This helps to setup environment for the performance tests • Team must be aware of the 'Done' criteria of performance testing upfront. This is applicable for many of NFR (Non-Functional Requirements) tests like availability, security, scalability etc. and not just limited to performance testing only
Exploratory Testing	• An unorthodox means of testing with no specific plan or testing approach • Exploring and identifying what is working and what is not in the software. Time is a key factor in this testing • Most of the standard test cases for the Sprint Stories are automated. Even though Automations capture the majority of functionalities, they won't be able to identify the usability, reliability, compatibility and other quality areas. Manual verifications that need to happen to ensure the product is behaving as expected • Testers identify areas that can have issues or impacts and then design manual test cases, run the tests, analyze the results and decide on the next phase of test cases to be run • During the story development asking the right set of questions can bring more clarities to the acceptance criteria and help developers to design the product in a right way For example: Web page is getting developed and as part of a story there must be all the fields that need to be present in the web page. The story normally won't have the details of the upper and lower limits of all the data entry fields, the browsers it is supported or the load it can take. However all those questions bring the right kind of clarity to the story and hence story can be developed in the right way from day one
Client/User Acceptance Testing	• There are regular Sprint Reviews (at the end of the Sprint) where team get inputs from the Business Stakeholders on the product that is being developed • Formal User Acceptance Testing is planned in the Release Hardening Sprint when all functionalities are fully developed • Chances of identifying defects during this phase gets low because of the regular feedbacks received as part of Sprint Reviews

Testing: Defect Management

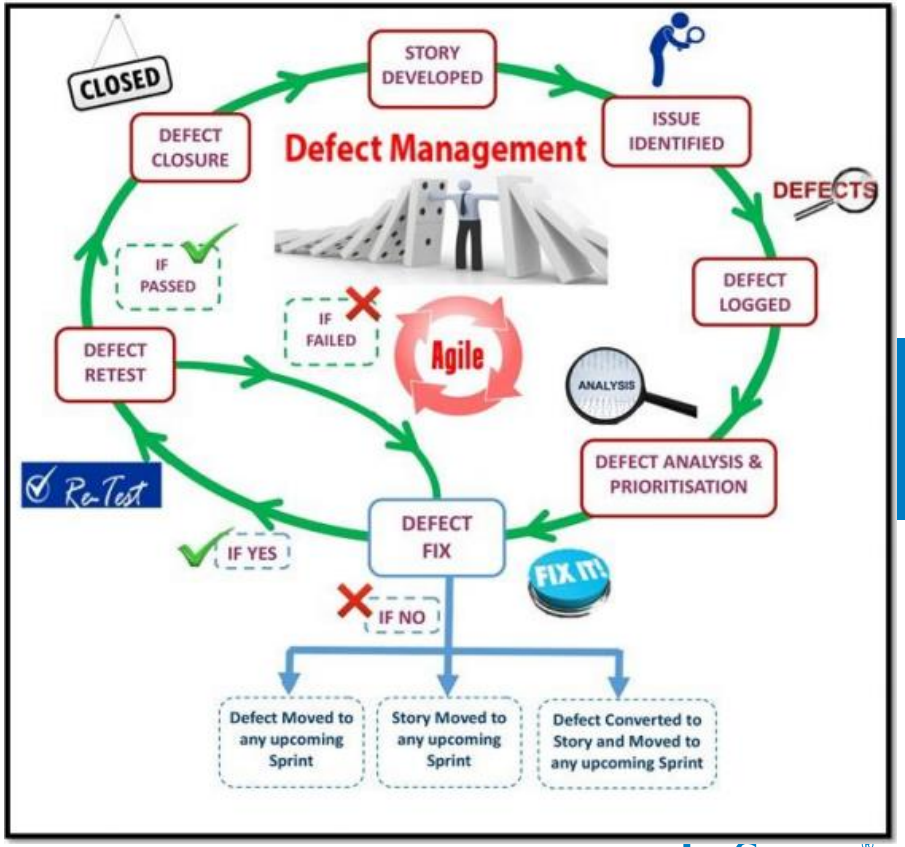
- Whenever the stories are available to test, the testing team starts testing and finds bugs
- The usual practice is, if the bug or the issue can be fixed in a day, defects are not raised, and the development team is communicated to work on the issue. If the issue which is found is not fixed in a day, then a defect is raised to make the team and the relevant stake holders aware of the issue
- This has to be tracked formally in a defect management tool
- The critical defects are discussed with the team along with Product Owner who finalizes on the criticality of the defect and the criticality of the story under which the defect has been logged

The defects are prioritized on the following basis

1. Impact on the test coverage
2. Impact on meeting the acceptance criteria of the story

What if a defect comes up late during the sprint or some defect which is directly not related to any of the stories taken up in the sprint?

- In such cases, a defect is raised and discussed whether it can be fixed in the same sprint and also how much of effort is required to fix and retest it
- If the effort is too high and if forecasted that it cannot be delivered as a part of the current Sprint, it is either moved to a story related to the defect which is then planned in future Sprint, or it is converted to a new story and gets assigned to a future sprint



Infosys
Navigate your next

Testing: Metrics

Defect Removal Effectiveness (DRE)

Defect removal effectiveness is a metric that tells that how many defects testers could identify in their testing and how many of them got slipped to next stage i.e. UAT or Production or Warranty period etc. DRE is expressed as %.

$$\textit{Defect Removal Effectiveness (DRE)} = \textit{(No: of In-process defects / Total no. of defects)} * 100$$

(Total number of defects = In-process defects + Defects found in Sprint/Release review)

Automation Test Coverage

Provides an insight to the percentage of test cases being automated as a part of the project. Each Sprint covers certain test cases to be automated which can be run in an automated fashion in the future Sprints. In this way, the number of test cases being automated increases Sprint by Sprint reducing the number of manual test cases thereby providing higher percentage of automation which can cover system cases during release level testing, rather manually executing the cases

$$\textit{Automation test coverage metrics} = \left(\frac{\textit{Number of automation test cases}}{\textit{total number of test cases}}\right) * 100$$

Automation Coverage Effectiveness

Defines how effective the Automation test cases are by having the number of defects caught through automation testing. The more the defects caught through running automation test cases, higher is the effectiveness. With each Sprint, team can work on the defects caught manually in the earlier sprint to incorporate the changes in automation so that the effectiveness of the automation test cases increase. If the effectiveness remains the same/decreases, then the team need to strive to have some changes in the automation testing strategy

$$\textit{Automation coverage effectiveness} = \left(\frac{\textit{Number of defects caught by automation}}{\textit{total number of defects caught}}\right) * 100$$

Testing: Metrics

Test Coverage (using code coverage)

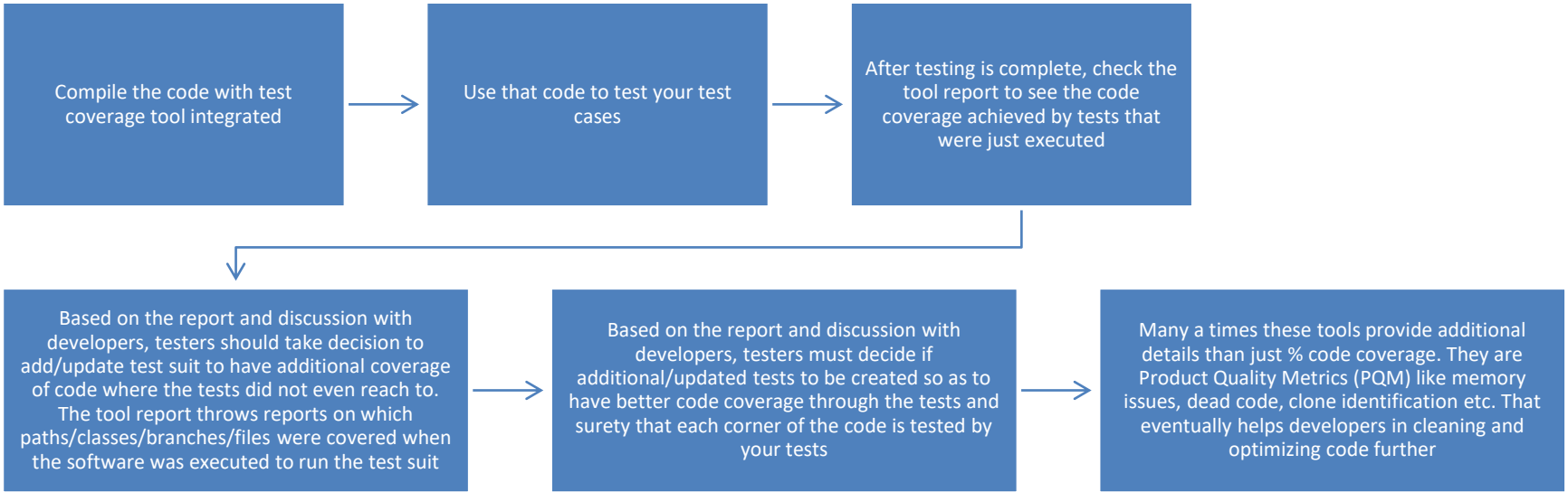
- How much of the source code is actually exercised by my test system?"
- Testers can look inside the Black box code and analyze how much of the code was actually exercised by the test set
- Tools like like ‘Semantic Design’ test coverage tool, ‘Bullseye Coverage’ can be used

Unexercised code can mean either

- Test system is deficient
- Source has "dead code" (code that cannot be reached by any path)
- Extra code is in the codebase that is not necessary (perhaps a feature was removed but not all of the associated code was deleted)

Test Coverage % = % of Code covered through the Test Suit

Test coverage targets should be 90% to 100%



Testing: Challenges

Challenge	Code broken accidently due to frequent Builds	Performance Testing Bottlenecks	Wastage of Automation scripts after couple of sprints
Description	Since code gets changed and compiled daily, the likelihood of code breaking the existing features is much higher	Agile involves regular builds and subsequent testing, and hence environment availability becomes difficult for the performance testing activities	There is always a risk of lot of rework and wastage of automation scripts in case they are not developed properly. With change in project course is quite normal in Agile, automations would need modifications which may impact its ROI (Return on Investment)
Solution	Have a smoke test bed created, to be executed after every code build. As resource constraints prevails, so it is not practical to have team members do this daily thereby having efficient automated test cases built to confirm the environment stability after code drops	Performance testing is mandatory for projects undergoing changes which have architectural impacts. It is very essential for Agile projects to carry out performance testing. To overcome the challenge, team should always have a separate environment for performance testing. The setup should be backfilled with the correct code base and then a round of smoke testing has to be done on the same. There should be separate stories to track the performance testing progress	Automation scripts should have high reusability factor with data sheets driving the automation execution. This way team can reduce the code changes to the automation scripts for any change in the Business functionality. It's an industry wide best practice, but required more in Agile since it's prone to more frequent changes.

Testing: Challenges

Challenge	Wrong selection of Sprint to start Automation of test cases	Wrong selection of Automation Tool
Description	It's always a challenge to decide on the correct Sprint for starting with the automation of manual test cases. If Automation is planned at the early stages of the Release (i.e. from the initial Sprints onwards), it leads to a lot of wastage of effort as the team is not settled properly and the output is meagre to do any automation. This leads to the automation work getting redone in subsequent Sprints, leading to wastage of effort. In case of late start of automation (i.e. during the last one or two Sprints in the Release), the team would have more manual test cases than automated ones. This would significantly impact the team's velocity and delivery capabilities.	In Agile, User Stories get reprioritized and nothing is constant if compared on a day to day basis. Hence if the Automation Tools are not selected properly, there is a chance of the project team running into higher costs and less ROI for the automation.
Solution	Although it varies from project to project, but planning the automation ideally after 3 Sprints is a good idea, since by then there would have been significant application development and stability to carry out the automation work.	For Agile Teams, it is always recommended to go for Open Source Tools without any licensing costs. This would give automaton access to all the team members ensuring better/higher outputs at low cost. However for projects, where there are members dedicated for Automation work, tools should be selected based on the kind of project being developed. Ex: A tool with Command Line Support is perfect for projects which involve testing for backend systems rather than User Interface.

Testing: Best Practices

- **Project/Release Initiation Phase**:

  - Testers should get involve right from the beginning of the Project initiation to understand the requirements better
  - Understand the high level requirements and highlight the issues (if any) in the very early phase of the project
  - Provide inputs to Product Owner or Business Analyst while setting business priorities in Product Backlog
  - Understand the business priorities and help in the Release Planning

- **Project/Release Planning and Sizing**:

  - Both, developers as well as testers should participate in the Product Backlog grooming sessions
  - Developers and Testers should together participate during Planning Poker Game which helps in accurate estimates and Story Sizing
  - Customized Estimation Template is used to capture defect rate, Injected defect rate and rounds of Regression Cycle based on past agile project experiences

- **Sprint Planning**:

  - Testers should mandatory attend the Requirement walkthrough sessions to capture more number of requirement/design findings & production issues in early phase
  - It is always recommended to revisit and revise estimates after Sprint level requirement analysis
  - Modular approach for Test Planning and Scripting should be adopted
  - Test scenario walkthrough with the team at end of planning phase will always help every member to get a broader idea

- **Sprint Execution**:

  - Consolidated/modular approach for Test Data setup activities
  - Test data repository to increase the reusability
  - Metrics Based test execution for better tracking and coverage
  - Risk Based testing in the critical situations
  - "Sprint Traceability Metrics" to maximize the test coverage
  - Agile Review Checklists to increase the quality of deliverables
  - Strong collaboration with waterfall team for workload balance during crisis

Automation

Automation: Overview

- Automating the repetitive procedures can provide real value to the projects
- Continuous integration/builds, unit, functional & integration test execution and continuous/automated deployment are common examples of applying automation

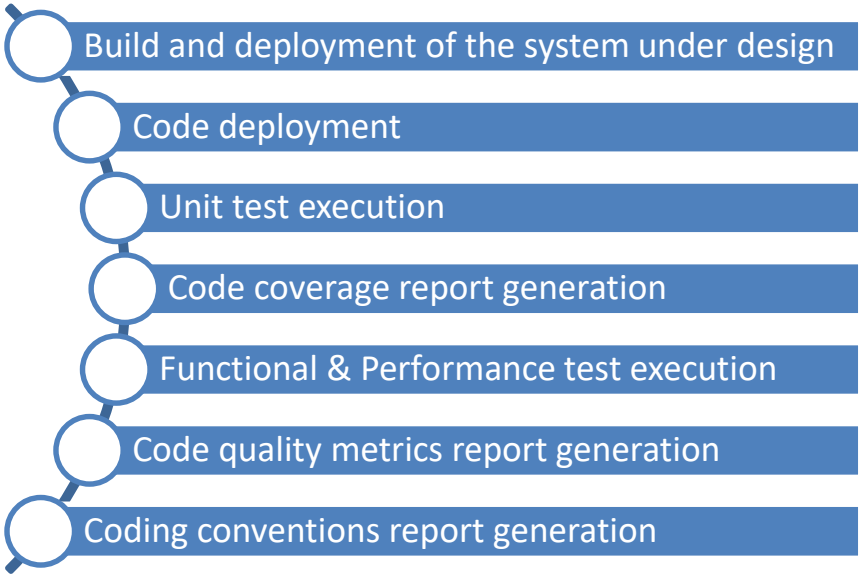
Benefits:

- Automation in software design and testing eliminates risks associated with human error
- 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
- Ensures reusability and scalability, which in turn translates into significant cost savings

Team needs to

- Investigate on how to automate the process and identify the tools that could assist with automation
- Estimate the level of effort required to implement the automation against the total cost and risk of manual procedures

Areas where automation can play an important role



Automation: Concepts

Test Data Management in Agile

- Agile Testing in Sprints gets difficult when the system is dependent on another team for test data. Sprint timelines do not give enough room for any delays in the data setup
- Setting up the test data is done along the Sprint to make available them at least, 'just in time' (JIT) if not earlier, by the time when the developer checks-in his code for claiming it 'as done'
- Developers usually practice test automation for unit testing and integration testing, use TDD approach to complete user stories, the synchronized supply of the test data by testing team is essential
- Analysis of the data needs is done during the backlog grooming sessions and the Sprint planning meetings to ensure the complete data set up needs
- If data is complex and huge, we may not get the adequate support from the team owning the data. Hence the discussion should ideally start with the business, at least a Sprint early or as per the applicable SLA to get the data at the appropriate time
- Members (from business) who are responsible for the data should be included as part of the Scrum team for the Sprint where their support is required
- Test data created in earlier sprints can either be directly reused or used with minor modifications in order to avoid redundant data requests/set up and reduced turnaround time for the test data creation and manipulation

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.

Automation in Development

Unit Test suite

- 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

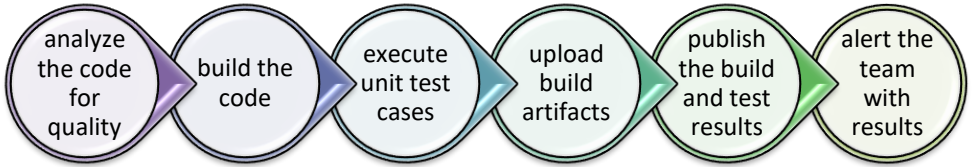
- Static code analysis needs to be performed to identify and quantify the code quality metrics like standards, security risks, and adherence to best practices
- As part of the development practice, the code quality checks should be run
- Only after attaining satisfactory quality metrics, the code should be checked into the version control tool
- Some of the key tools for Static Code Analysis are CAST, SONAR.
- Developers can make use of the IDE(Integrated Development Environment) integration capabilities of these tools to integrate static code analysis in their development process

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.

Following is the process:



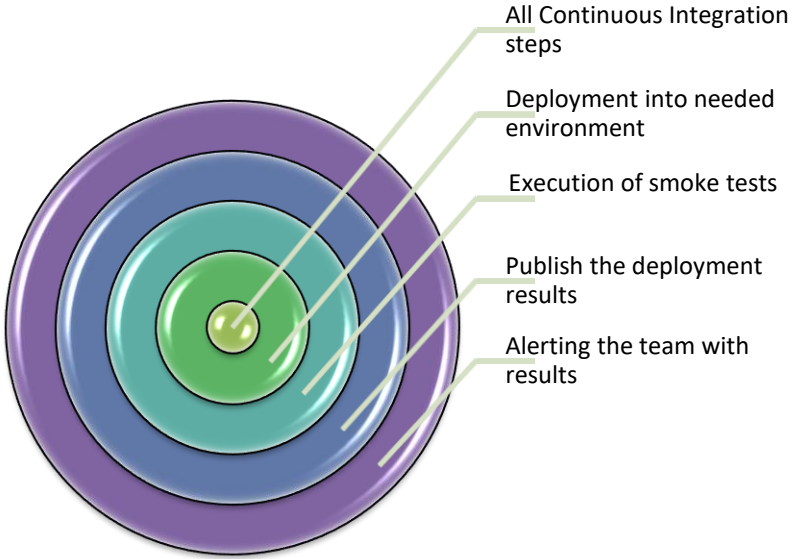
To aid in this practice following support systems are required

- **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

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

A typical CD process includes



- 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

Automation in Testing

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

Smoke testing

- Few sanity test cases can be handpicked and a suite of tests can be created
- Test suites will be executed from the build server after every deployment tasks
- Ensures that the basic features are validated automatically

System Testing

- 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
- Ensure test case library is always updated
- Future maintenance/iterative work on the same module will not need excessive testing effort

Regression Testing

- To keep 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
- 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

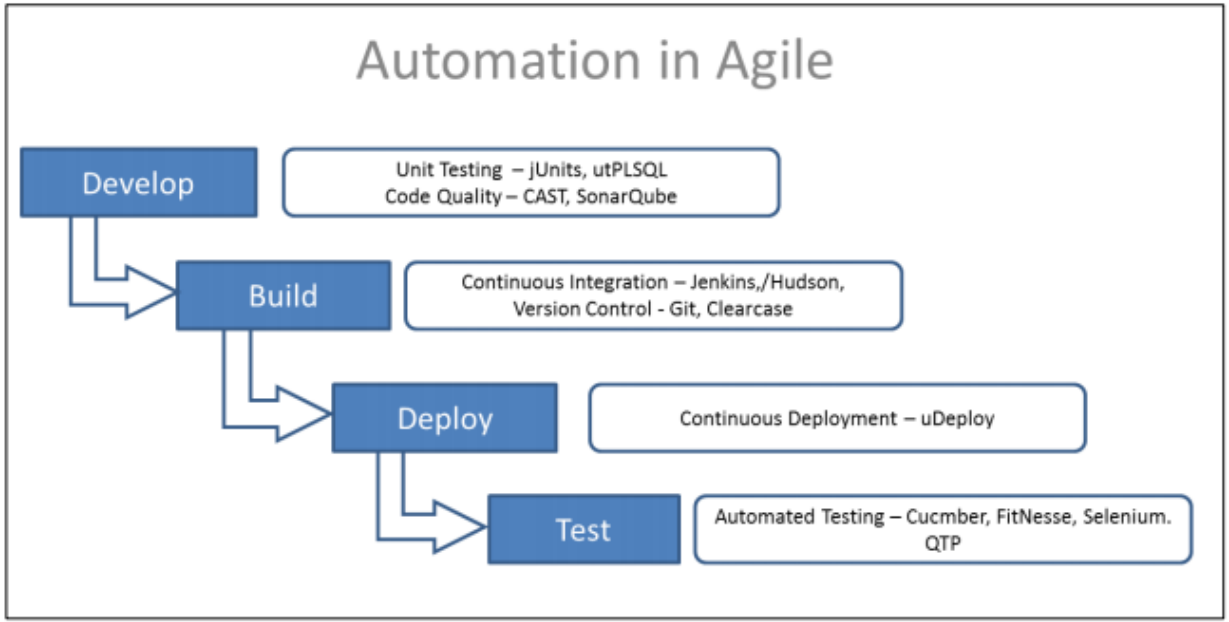
- Aim of defining the acceptance criteria is to automate them easily
- Multiple frameworks/tools exists to support these acceptance testing including Cucumber, FitNesse etc.
- This method of defining the acceptance test cases initially and then developing the code based on it is known as Acceptance Test Driven Development (ATDD)

Automation: Emerging trends

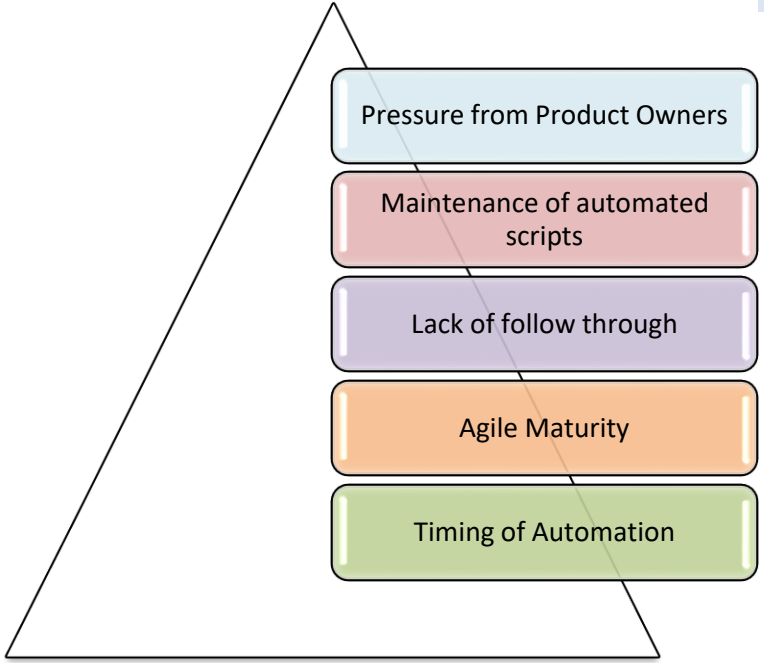
LiquiBase:

- A tool that tracks the changes that are made to the database
- Changes to the database are maintained in an XML
- Helps in subsequent Rollback of changes or applying a change in one environment to another
- Can be used to create/rollback test data for unit test cases or to move the changes from DEV environment to higher environments

Chef: A tool that automates deployment and management of infrastructure. Infrastructure is treated as versionable, testable, and repeatable similar to the application code.



Automation: Challenges



- 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
- 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

- Automation scripts have to be 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

- 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

- 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

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

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.

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

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

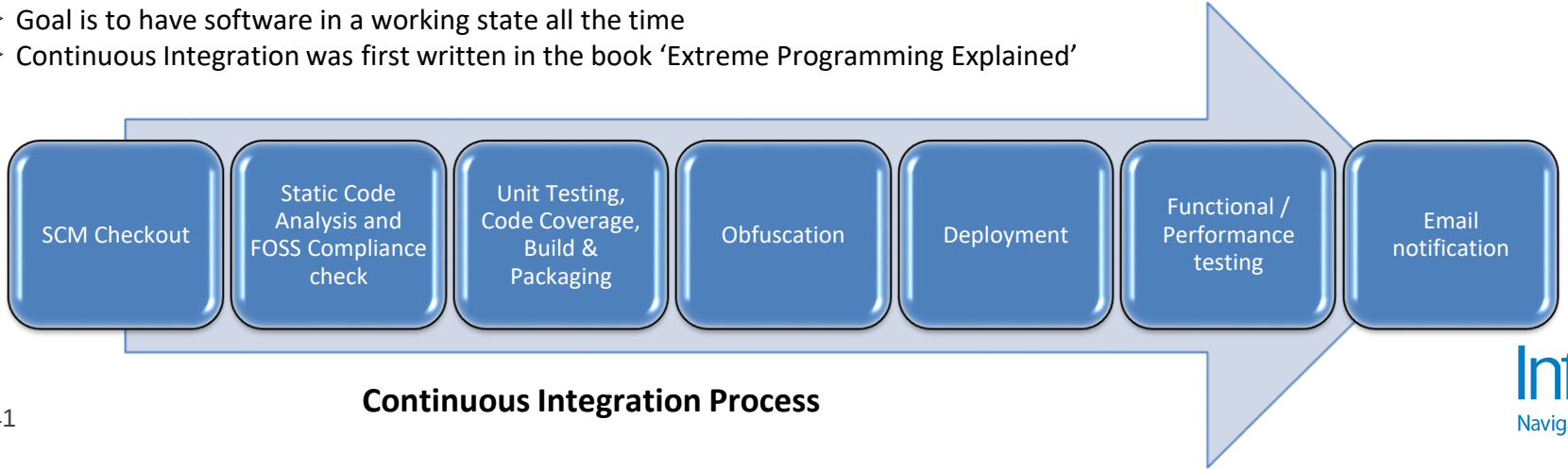
Continuous Integration and Associated Tools

Concepts

Continuous Integration is continuously integrating source code into a central repository wherein each instance of integration is verified by the Automated Build process to ensure the stability of the project (working software) at all times

- Use of Continuous Integration effectively will deliver software much faster, and with fewer bugs
- helps to identify the integration defects much early in the software development lifecycle
- Reduces risk by providing faster feedback
- Facilitates coordination between team members and encourages collaborative approach for problem solving and process improvement in the project

- Requires that every time somebody commits any change, the entire application is built and a comprehensive set of automated tests is run against it
- If the build or test process fails, the development team stops whatever they are doing and fixes the problem immediately
- Goal is to have software in a working state all the time
- Continuous Integration was first written in the book 'Extreme Programming Explained'

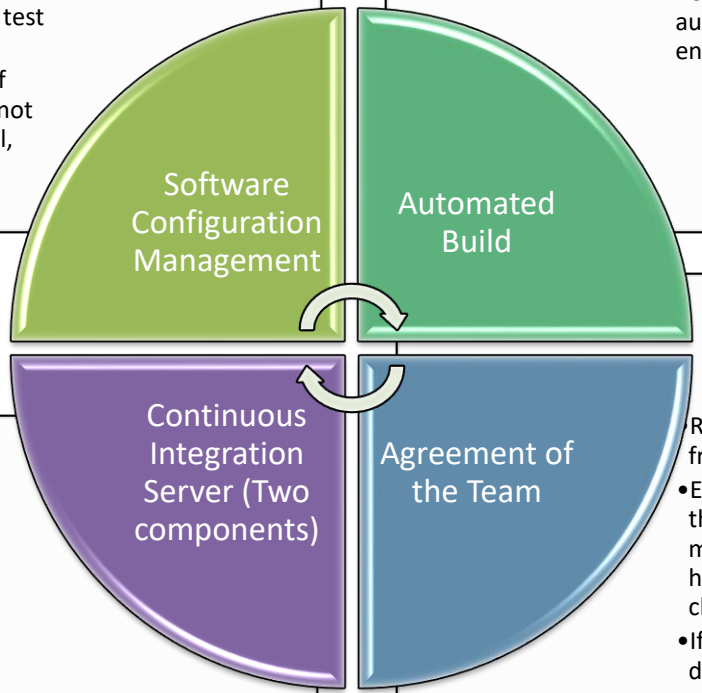


Continuous Integration Process

Infosys®
Navigate your next

Things needed before getting started with CI

- Everything in the project must be checked in to a single version control repository i.e. code, tests, database scripts, build and deployment scripts, and anything else needed to create, install, run, and test the application
- There will be projects that don't use any form of version control considering that their project is not big enough to warrant the use of version control, which is incorrect

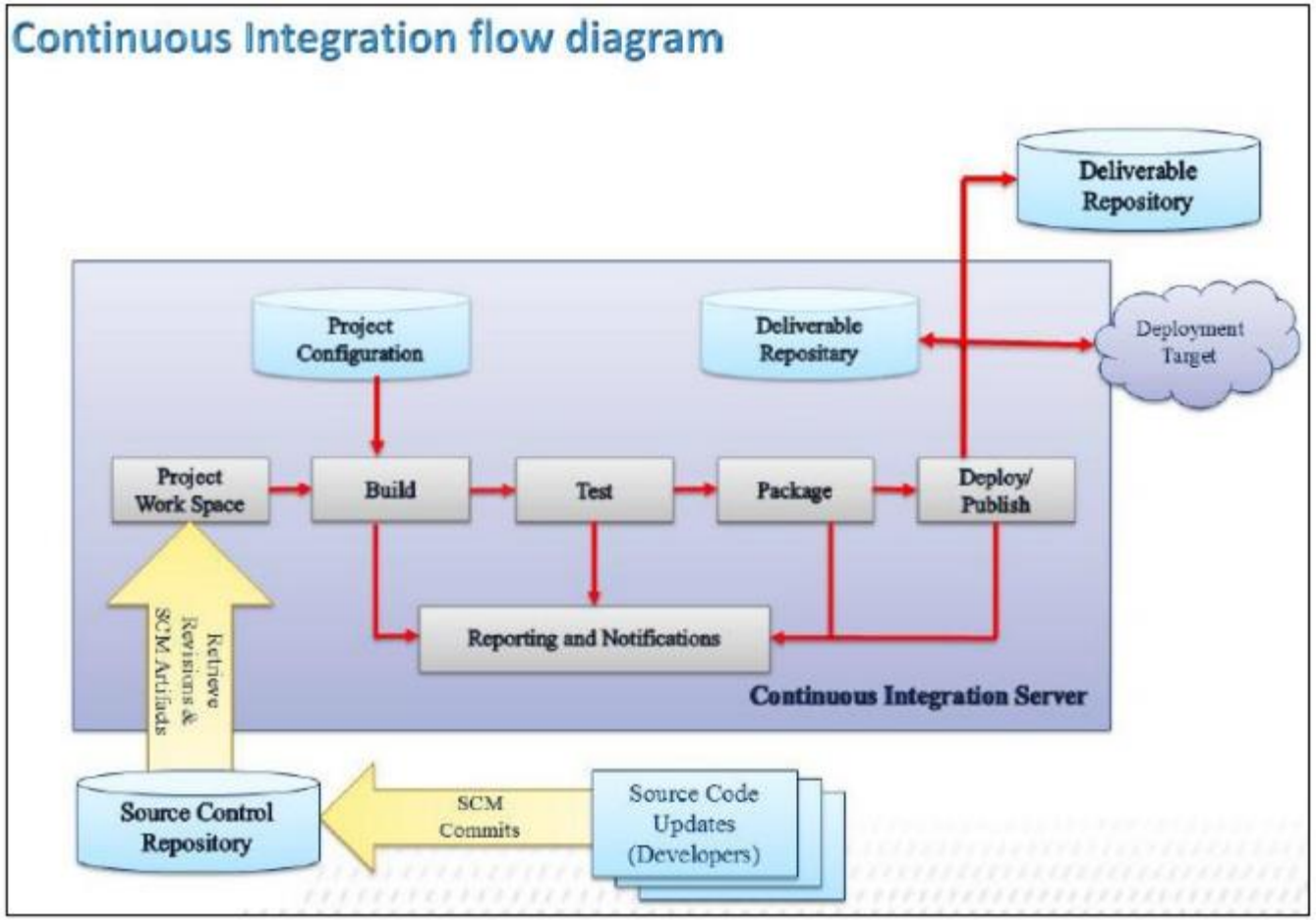


- Project build, test and deployment process should be run in an automated fashion
- Team must be able to run the build process in an automated way from the continuous integration environment

- First: Long-running process which can execute a simple workflow for regular intervals.
- Second: Provides a view of the results of the processes that have been run, notifies the team about the success or failure of the build and test runs, and provides access to the test reports, installers and so on.

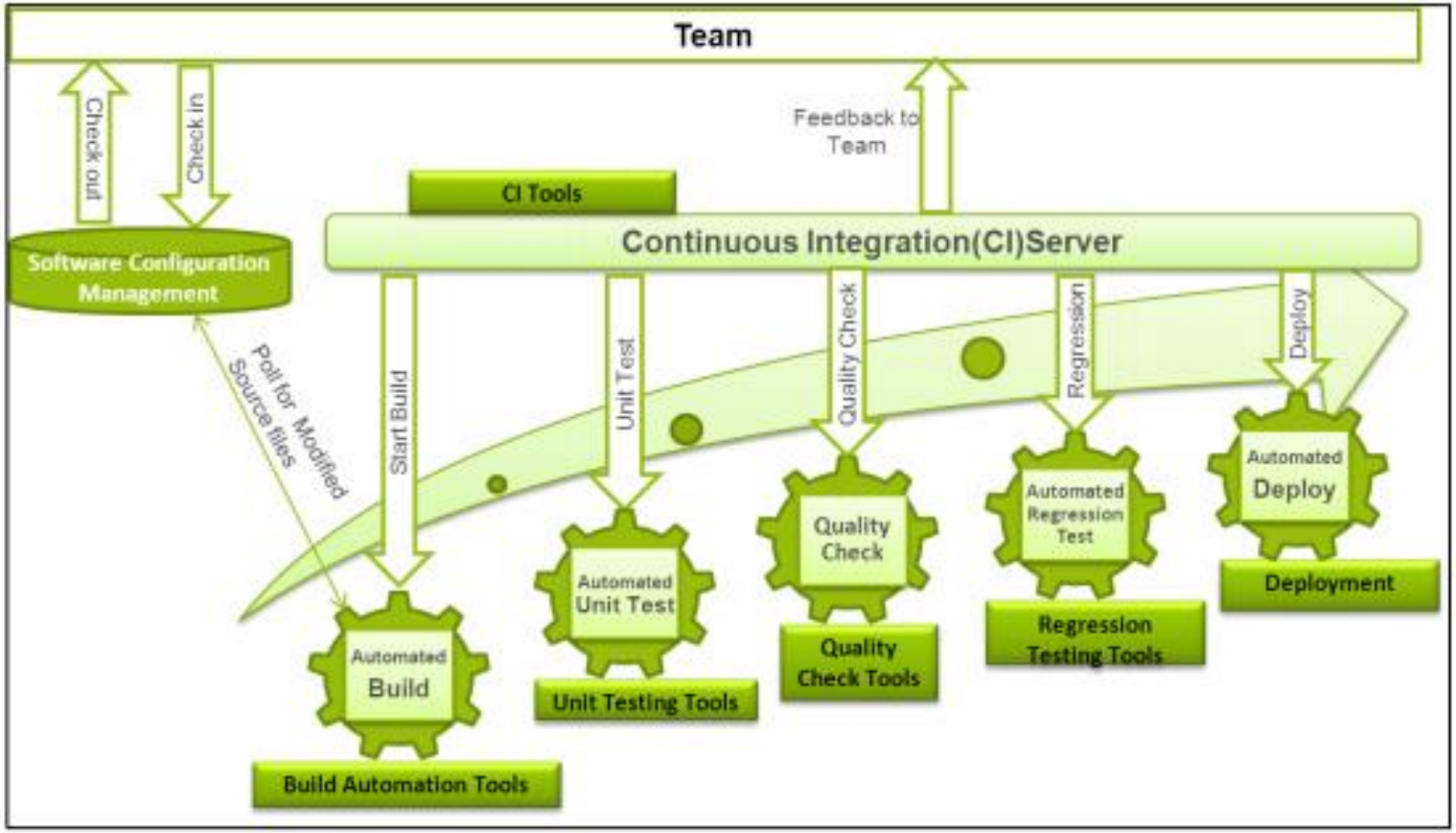
- Requires a degree of commitment and discipline from the development team
- Entire team must follow the process of checking in the small incremental changes frequently to mainline and must be in agreement that the highest priority task on the project is to fix any change that breaks the application
- If any of the team members don't adopt the discipline necessary for it to work, then the attempts of Continuous Integration will not lead to the improvement in quality

Continuous Integration Flow Diagram



Infosys®
Navigate your next

Stages in Continuous Integration



Infosys®
Navigate your next

Stages in Continuous Integration

Stage for Build, Unit and Integration Testing and Quality Check

Build gets initiated when the developers commit their individual work/code to the version control repository system. The CI server which is installed as part of project environment, continually polls this repository (e.g. frequency could be every few hours based on the need of the project) for detecting any changes made by the development team

The CI server then detects changes, if any, in the version control repository, and initiates the process of retrieving the latest copy of source code from the repository and executes the build script using build automation tools to perform integration of the software

Once a successful build is generated, CI server triggers the process of validation of test cases/scenarios and code quality inspection. The process of validation and inspection is automated using the automation tools for unit and integration testing and inspection

Though these steps are automated, there would be some manual intervention required in these stages. Post this activity, feedback is generated by the CI server and is communicated to the team members through various mechanisms such as email and text message

Stages in Continuous Integration

Stage for System/Functional/Non-functional Requirement Testing

- Component Tests: Verify portions of a system such as database, file systems, or network end points
- System Tests: Exercise a complete software system including external interfaces
- Functional Tests: Executed to test the functionality of the application or product from the client perspective

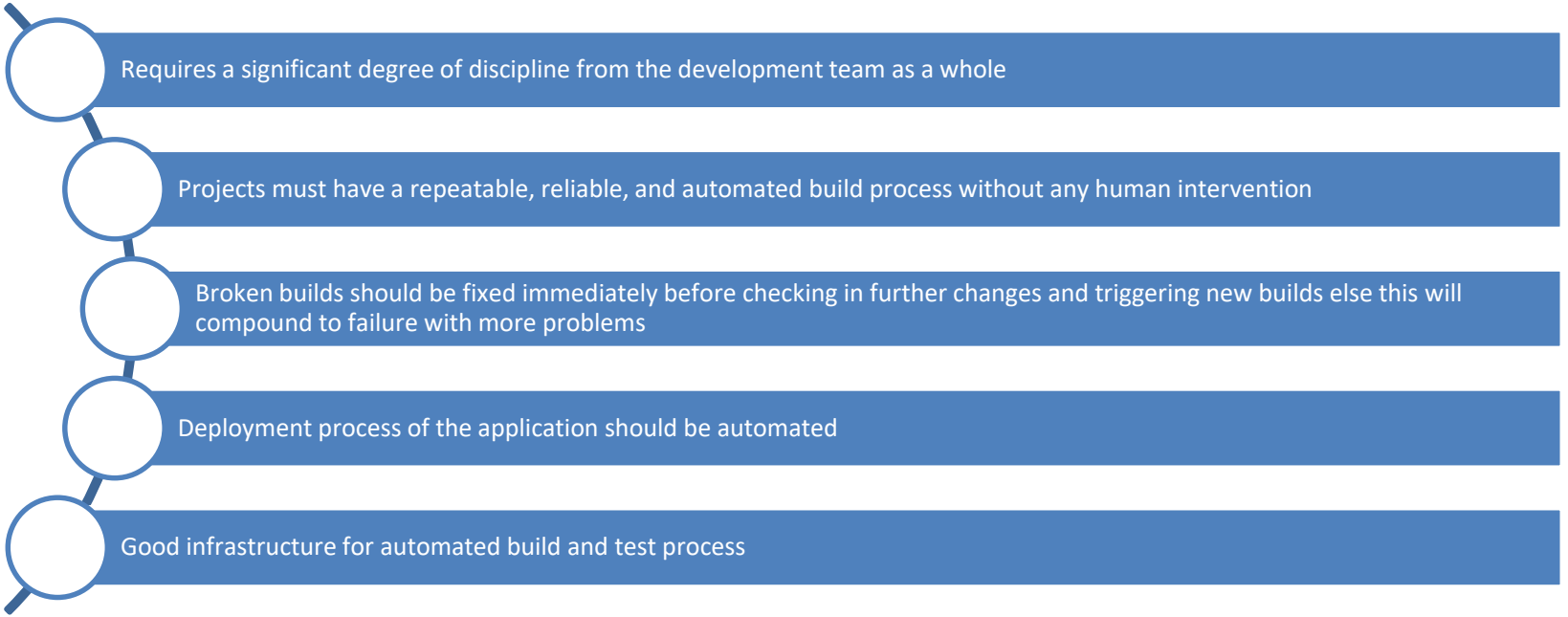
Component tests take a bit more time to run when compared with Unit testing because of multiple dependencies whereas the System and Functional tests take the longest time to run

- CI system would be configured to automate and run all these types of tests
- Tests can be categorized into distinct buckets and run the slower running tests (example, Component, System) at different intervals than the faster running tests (example, Unit)
- CI system Configuration would run each of these test categories with different staged builds
- Slower running tests (example, System and Functional) can be automated through the tools interfaced with the CI server

Stage for Automated Deployment

- Continuous deployment enables to deliver working, deployable software at any point in time
- By running a fully automated build including compilation, all tests, inspections, packaging, and deployment, the team acquires the capability to release working software at any time and in any known environment
- CI needs CD for the culmination of all engineering practices which enables the release of working software at any time and any place with minimal effort
- CD should include the capability to automatically roll back all changes applied in the deployment

Practices



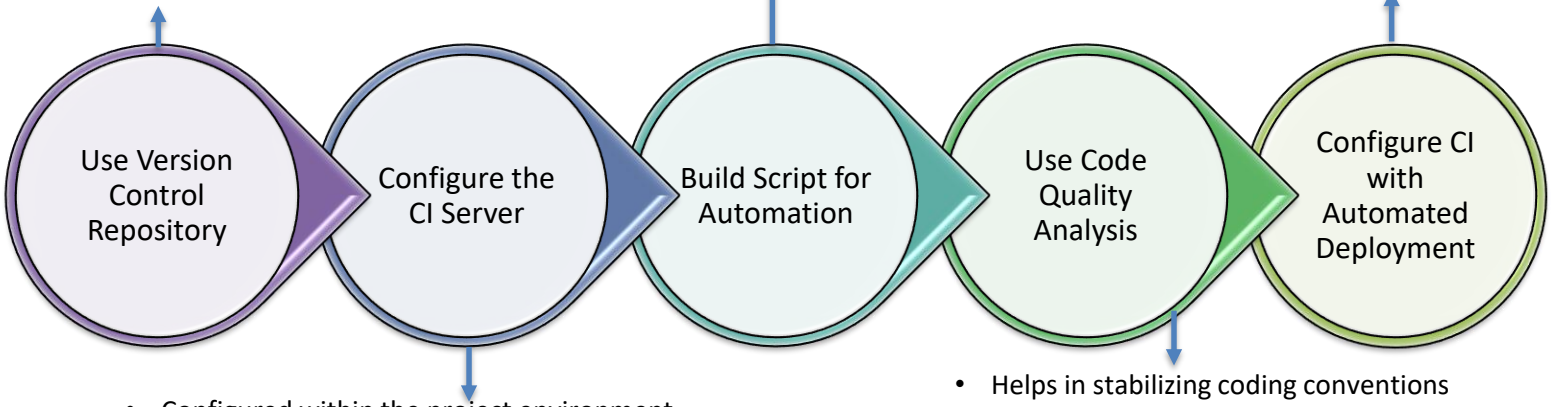
- The most basic functionality of CI server is to poll the version control system for new commits, running build scripts for compilation, running the tests and notifying the results to the team
- Teams commonly determine test coverage, code duplication, adherence to coding standards, and other indicators of health, and have the results displayed in the CI server's summary page for each build

Step-by-step Process of Implementing Continuous Integration

- Manage all the source code changes and other software artifacts
- Uses a controlled access repository
- “single source point”, all source code is available from one primary location

- Developed and implemented (which could be a single script or a set of scripts)
- Automate the software build cycle which includes compiling, testing, inspecting, and deployment processes
- Developers run Unit, Component, System, Functional, and Regression tests to speed up the build process

- After the creation of each build, new version of binaries should be automatically deployed to test servers
- Helps in improving integration with customer systems
- After each fix, customer may verify new versions



- Configured within the project environment
- Continuously check for any changes in the version control repository
- Run an integration build by retrieving the source files whenever a change is committed to the version control repository

- Helps in stabilizing coding conventions and constantly monitors code for

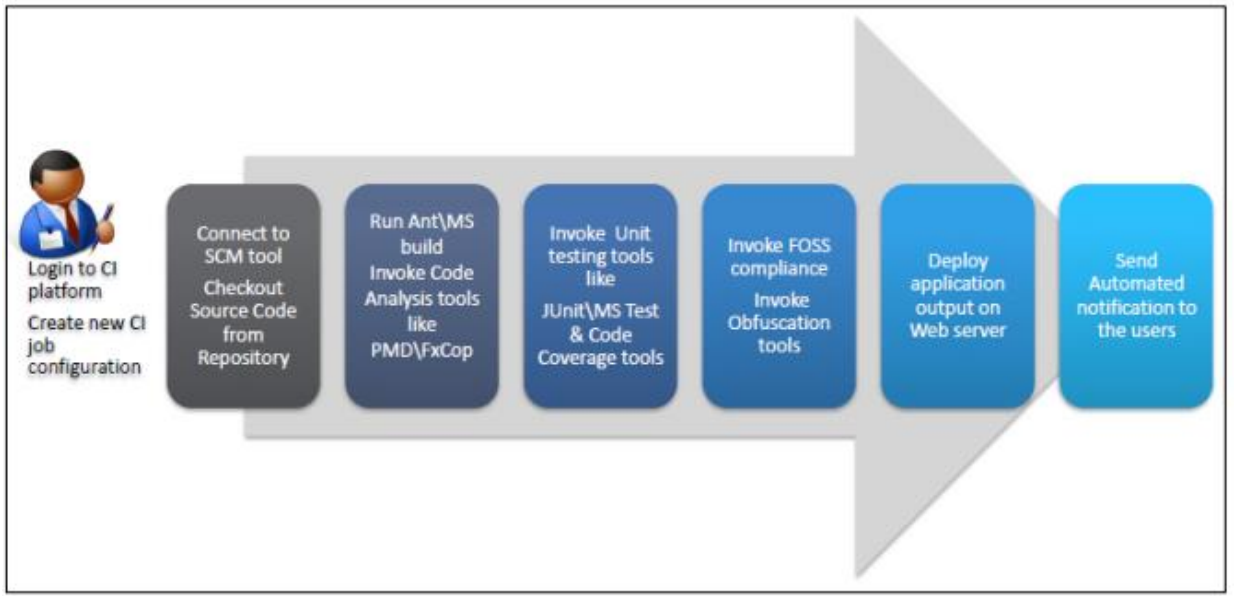
1. Average code complexity
2. Coverage of code
3. Duplication of code
4. Performance

ICIP & its Workflow

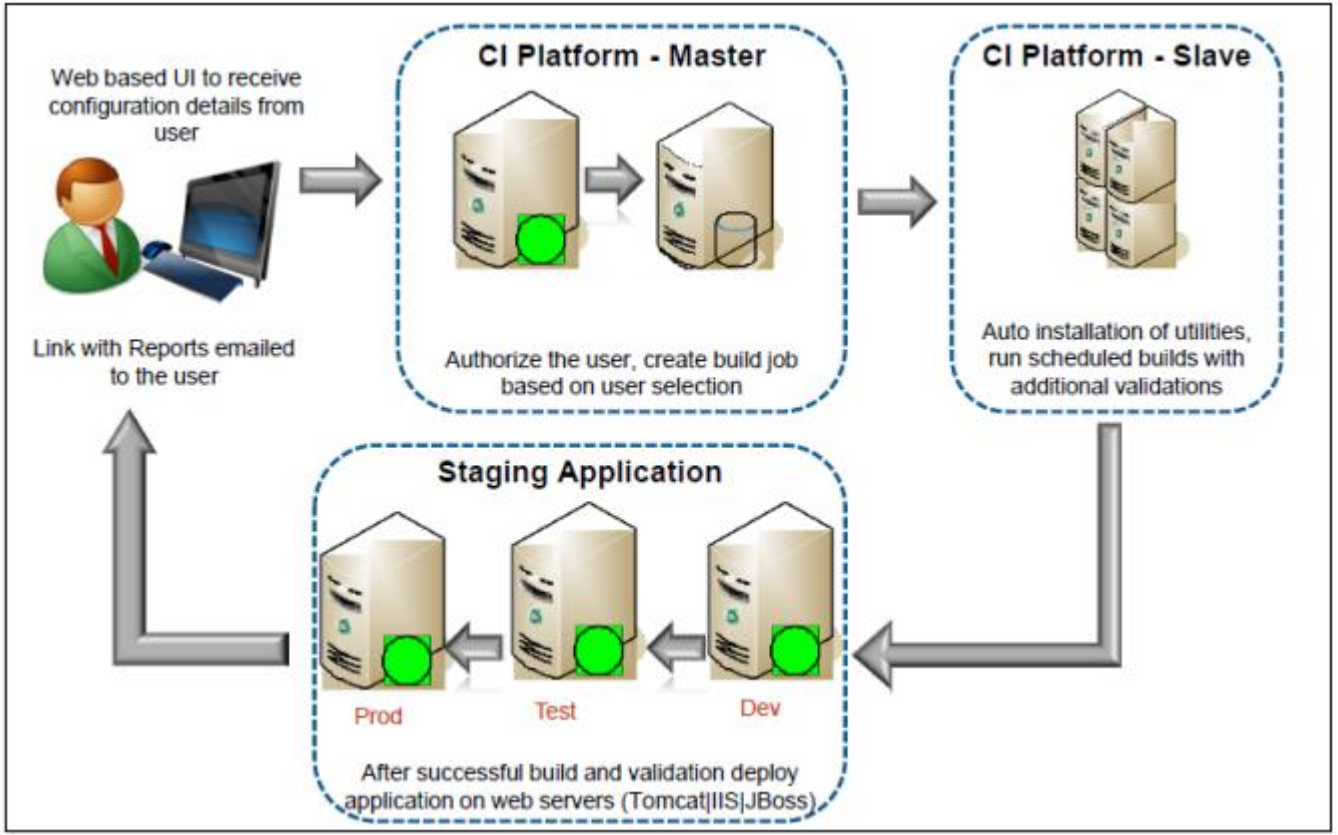
Infosys Continuous Integration (CI) Platform is a centralized hosted platform, which enables the project teams to leverage a predefined continuous integration workflow for build and deployment. It is web based solution designed to automate build for JAVA/J2EE and .NET based application projects.

- Enables teams to schedule the builds to run at regular intervals or based on source control action performed by the developer
- Leverages the plugins such as Custom Tools, Source control management (SCM) plugins, Build tool plugins, Code analysis plugins, unit testing plugin, Code coverage plugins, and Authentication and notification plugins
- Infosys internal team has developed custom plugins such as createuser plugin, copy-slave plugin, save-job plugin to define the continuous integration workflow

- Leverages the batch script execution to ensure activities such as Free and Open Source Software (FOSS) compliance and securing intellectual property of the source code through obfuscation
- Identifies the cross project dependency for Java/J2EE projects and build multiple projects at one go and support parallel deployment on multiple web servers



Architecture Diagram of ICIP

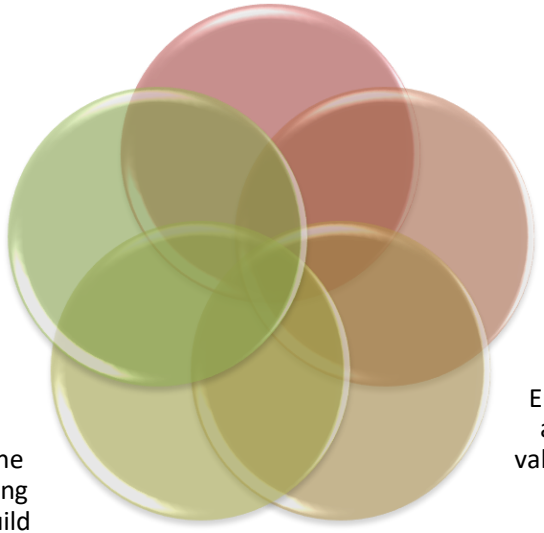


Infosys®
Navigate your next

Infosys Continuos Integration Platform (ICIP): Benefits

A centralized platform available as-a-Service that reduces project teams effort for set up & administration

Significant reduction in cycle time through automated build & deployment



Implementation of Standard CI process that leverages Industry best tools & practices

Improved adoption of the CI System due to deskilling of administration and build script creation

Enhanced compliance & automation of several validations - code analysis, unit testing, code coverage, FOSS compliance, obfuscation, packaging, deployment, etc.

Tools Associated with Continuous Integration

CATEGORY	TOOL NAME
Software Configuration Management Tools	CVS
	Clear Case
	SVN
Continuous Integration Servers	Cruise Control
	Jenkins
	Rational Team Concert
	TFS Team Build
Code Quality	Cobertura
	SonarQube
	Parasoft
Build Tools	ANT
	Maven
	NANT
Unit Testing Tools	JUnit
	NUnit
	Parasoft Jtest / Parasoft dotTEST
	Quest Code Tester
Integration and Functional Testing	Rational Functional Tester
	HP Unified Functional Testing
	Parasoft SOA Test
	Rational AppScan
	Microsoft VSTS
	Selenium
Performance Testing	HP Load Runner (HP Performance Center)
	Jmeter
	Rational Performance Tester

CI Server Tools

Cruise Control:

- This server automates the process of integration by monitoring the source code repository of the team directly
- Every time a developer commits a new set of modifications, the server automatically launches an integration build to incorporate the changes

Key features:

- Allows remote management
- Reports using email, exe, RSS
- Builds multiple projects on a single server
- Integrates with other external tools such as Ant, Nant, MS build

Jenkins:

- Java-written Open Source CI tool
- Server-based system running in a servlet container such as Apache
- Supports Software Configuration Management (SCM) tools that include CVS, Clearcase
- Can execute Apache Ant and Apache Mavenbased projects as well as shell scripting

Rational Team Concert:

- Built under Jazz platform by 'Rational' brand of IBM
- Available in both Client version and Web version
- Licensed according to the number of users working with the software as well as the actions these users are allowed to execute with the system

TFS Team Build:

- Provides the functionality of a public build lab and is part of Visual Studio Team Foundation Server
- With Team Foundation Build, enterprise build managers can synchronize the sources, compile the application, run associated unit tests, perform code analysis, release builds on a file server and publish build reports
- Build result data is propagated to the warehouse for historical reporting

Code Quality Tools

SonarQube

- Web-based application that can be used to manage the code quality
- Very efficiently, navigates a balance between high-level view, dashboard, TimeMachine and defect hunting tools
- Due to this ability of the tool, it becomes easy to uncover projects and / or components which are in technical debt, so that action plans can be established effectively

Parasoft®

- Integrated development testing tool that automates various practices that have proven to improve productivity and software quality
- Used for static code analysis, data flow static analysis, and metrics analysis
- Facilitates automated peer reviews, Cactus test creation, execution, optimization and maintenance, along with runtime error detection

Cobertura

- Open source tool for Java that can calculate the percentage of code accessed by various tests
- Identifies those parts of the Java program that lack test coverage
- Based on jcoverage tool

Build Tools

ANT

- For Java Project
- Uses XML for defining the build process
- Open Source software released under the Apache Software License

NANT

- Open Source software tool
- Similar to ANT but needs .NET environment to work rather than Java
- The name NANT came from the fact that the tool is NOT ANT
- Released under GPL License

Maven

- build automation tool usually used for Java
- Used to build and manage projects written in C#, Ruby, Scala
- Uses XML to define the software project being built
- Downloads Java libraries dynamically and plug-ins from one or more repositories
- Released under Apache License 2.0

Unit Testing Tools

Junit

- Unit testing framework for the Java programming language
- Important in the development of test-driven development
- One of a family of unit testing frameworks which is collectively known as xUnit that originated with sUnit

Nunit

- Open source unit testing framework for Microsoft .NET
- Serves the same purpose as JUnit does in the Java world
- One of many programs in the xUnit family

Parasoft Jtest

- Automated Java software testing and static analysis product
- Includes technology for Unit test-case generation and execution, static analysis, regression testing, runtime error detection, and Design by contract

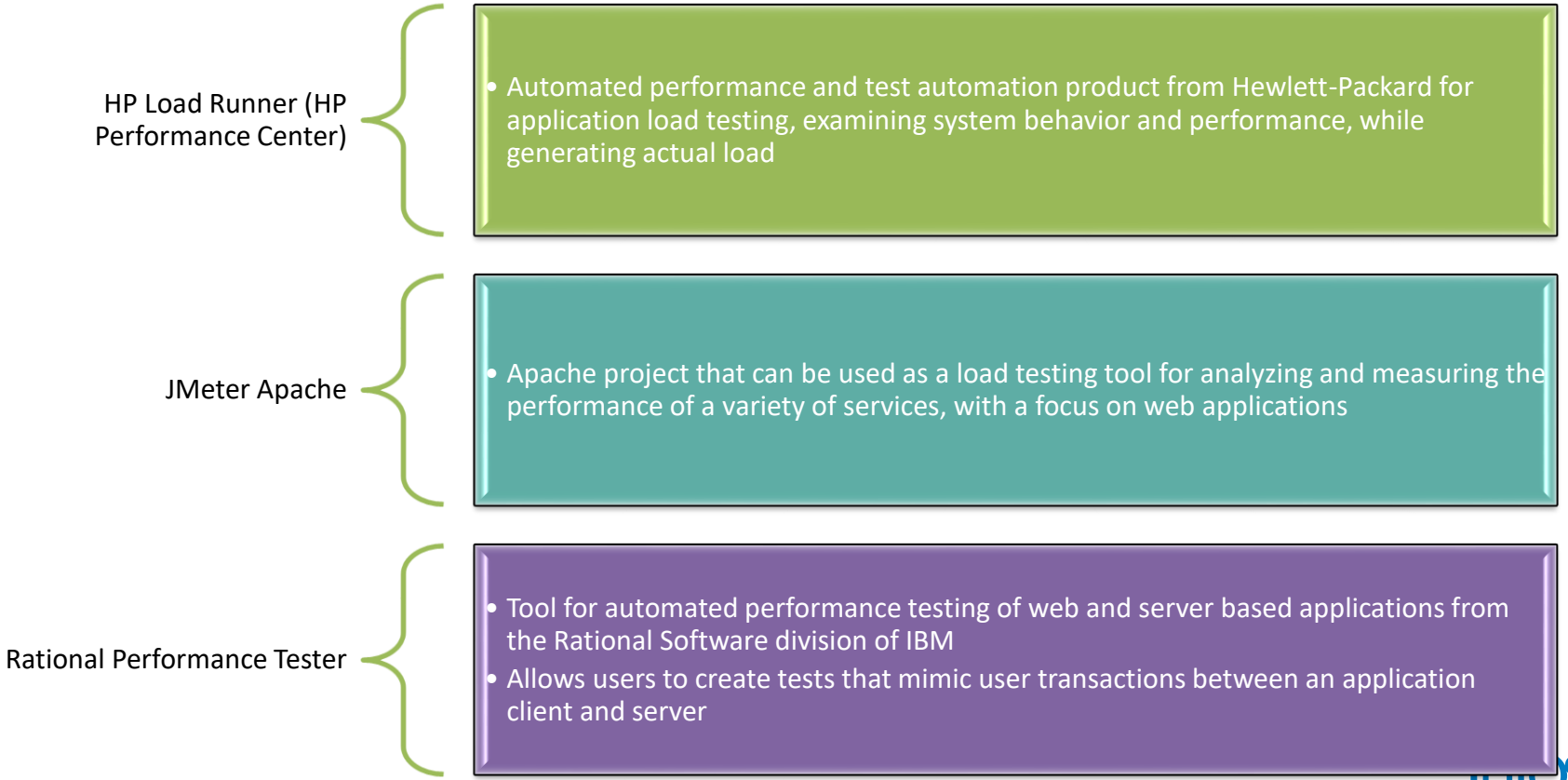
Quest Code Tester for Oracle

- Component of the Toad Development Suite for Oracle
- First and only automated PL/SQL code-testing tool available
- Provides an efficient and reliable way to perform thorough PL/SQL code tests

Integration and Functional Testing Tools

Rational Functional Tester	•Tool for automated testing of software applications from the Rational Software division of IBM •Allows users to create tests that mimic the actions and assessments of a human tester
HP Unified Functional Testing	•Formerly known as HP QuickTest Professional (QTP) •Provides functional and regression test automation for software applications and environments •Can be used for enterprise quality assurance
Parasoft SOA Test (Service-Oriented-Architecture)	•Testing and analysis tool suite for testing and validating APIs and API-driven applications (e.g., cloud, mobile apps, SOA) •Basic testing functionality include functional unit testing, integration testing, regression testing, system testing, security testing, simulation and mocking, and load testing
Rational AppScan	•IBM Security AppScan previously known as IBM Rational AppScan is a family of web security testing and monitoring tools from the Rational Software division of IBM •Test Web applications for security vulnerabilities during the development process, when it is least expensive to fix such problems
Microsoft VSTS Visual Studio Test Professional	•Integrated testing toolset developed by Microsoft to facilitate a plan-testtrack workflow for collaboration between testers and developers
Selenium	•Set of different software tools each with a different approach to supporting test automation •Most Selenium QA Engineers focus on the one or two tools that most meet the needs of their project, however learning all the tools will provide many different options for approaching different test automation problems •The entire suite of tools results in a rich set of testing functions specifically geared to the needs of testing of web applications of all types •These operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior •Support for executing one's tests on multiple browser platforms

Performance Testing Tools



Case Study

Cont…

Following is the Case Study from a project which has overcome various quality challenges through the implementation of Continuous Integration Platform.

Project Description:

Assist Edge Smart User Environment (SE) is an Infosys Product aimed to achieve desired operational efficiency for customer care function with the help of process automation, application integration and data handling techniques. It provides an intuitive interface for the customer care representatives to view the right information of customers from disparate enterprise systems and applications. It can configure and automate the information flow comprising data search, validation and aggregation to reduce manual efforts of agents thus increasing the productivity.

Technology details:

Database: Oracle

Language: .NET, Java;

Anticipated Challenges:

This project deals with product development where the quality of the product is a key factor. Also early detection of issues and timely fix ensures there are no defects that are slipped into the product. Some of the typical scenarios are illustrated below:

- Multiple member are working on single project. The team would need to work on the common visual studio projects. There are chances that changes made by one developer may break the code which is written by the other developer. There are chances that significant time is required to correct this type of issue.
- As multiple developers are working on the project it is very important to follow the pre-defined coding standards to avoid the security/logical type of issues.
- It is very important to ensure that all the added code is compiling properly so that if any other developer will take the code from TFS then s/he will not face any compilation issue.

Case Study

Benefits Perceived:

Following are the benefits observed by implementing Infosys Continuous Integration Platform (ICIP):

- **Daily Build:** ICIP allowed team to create automated daily builds which helped in tracking the code quality. Automation of daily builds is an important facet of agile development.
- **Code Quality:** Sonar dashboard shows the test execution status at a glance, an indicator of the quality of the changes made to the solution. Also ensures coding standards are getting followed.
- **Lines of code:** Sonar dashboard also gives the statistics of the lines of code, duplicate code, code documentation and code comments. Saves time spent in calculating the same.
- **Violations using FxCop**: The automatic FxCop analysis performed by the ICIP shows the rule violations introduced in the code on Sonar dashboard indicating the compliance.
- **Reporting:** ICIP also helped to send daily build report to stake holders through email.
- **Early error Detection:** Helps in executing the automated test cases to ensure newly added/existing code will not fail in any unexpected situations.
- This platform helps to ensure entire project is compiling properly on the daily basis. This helps developers to ensure the code added by them are not conflicting with the code added by other developers.

THANK YOU

Infosys®
Navigate your next