WHITEPAPER

Continuous Integration Build-Test-Delivery (CI-BTD) Framework in compliance with ISO26262

Manish Patil
Sathishkumar T

September 2015

L&T Technology Services

Contents

Abstract

Continuous Integration (CI) is a methodology, a mind-set change and a leadership practice that focuses on how to achieve rapid building, testing and delivery throughout the software development lifecycle. This relies on the adoption of automation to streamline manual processes and enforce consistency for frequently repeated tasks in the software delivery pipeline.

This paper demonstrates the steps and advantages of implementing a "Continuous Integration — Build, Test & Delivery (CI-BTD)" framework in the automotive industry to describe how Continuous Integration can be used to turn manual processes to an automated manner for faster and reliable results. This will enable automatic generation of the artefacts for the compliance with ISO26262 functional safety standard, part 6 - Product development at the software level that results in savings of about 70% of manual effort during the implementation and testing stages on the SDLC (Software Development Life Cycle). The complexity level handled using CI-BTD framework for the considered use case is for three different vehicle platforms with each platform project having about 20-25 software components or modules with seven stages of quality gates with possibility to run Jenkins jobs in parallel based on the infrastructure availability.

Keywords: Continuous Integration (CI), Build, Test & Delivery (BTD), Mainline, ISO26262, Jenkins

1. Introduction

The concept for Continuous Integration (CI) is driven by the factor that human brains are better at inventing new solutions, but can make mistakes in executing repetitive manual tasks. So the driving force for continuous integration is any activity, where there is a clear process being observed for repeated execution in a consistent pattern, which can be executed by an automated system running continuously on a machine.

The automation is achieved in CI by integration of various tools which includes configuration management tool (subversion), Jenkins with wide range of plugins, Matlab, Simulink, Embedded Coder, Compilers, QAC and Tessy.

2. Industry Challenges

In today's vehicles, electronics account for about 40% of total cost with emphasis on software and about 200 million lines of code which are developed in close collaboration with Service Providers, Tier 1's and OEM's. In today's vehicle development programs, the development teams are placed globally, increasing the geographical diversity resulting in bigger challenges to maintain and ensure the stable build software always.

Increasingly, proliferation of electronics has resulted in ever increasing complexity. Main factors behind the increase in electronics content includes increasingly stringent environmental regulations, advanced safety features, better drivability, increasing comfort and convenience, information and entertainment systems. In order to make the system safer and more reliable to the malfunctions, ISO26262 functional safety standard plays a vital role, which in contrary increases the overall efforts towards ensuring the complexity with as much as 40% of increased effort which in turn impacts the overall vehicle program cost and time-to-market.

In the competition that is present in the automotive industry, coming out with a product faster than your competitor while ensuring you have taken every measure to make the vehicle safe by reducing the time-to-market is important. In SDLC (Software Development Life Cycle), integration, build and testing are the stages where it takes a large amount of time. There is always a high possibility of missing out something when it is done manually, we can say automation with defined framework is more reliable in this case.

In order to cope with this challenge, we have introduced a “Continuous Integration - Build, Test & Delivery (CI-BTD)” framework (hereafter referred as “CI-BTD framework”), which can perform the processes automatically ensuring reliable and stable results in each iteration with as much as about 70% saving of efforts in comparison to manually doing the same processes, along with support towards generating the artefacts for functional safety compliance.

3. CI-BTD Framework — The Big Picture

The big picture of the CI-BTD framework with various levels of process maturity in an organization is outlined in the below figure.

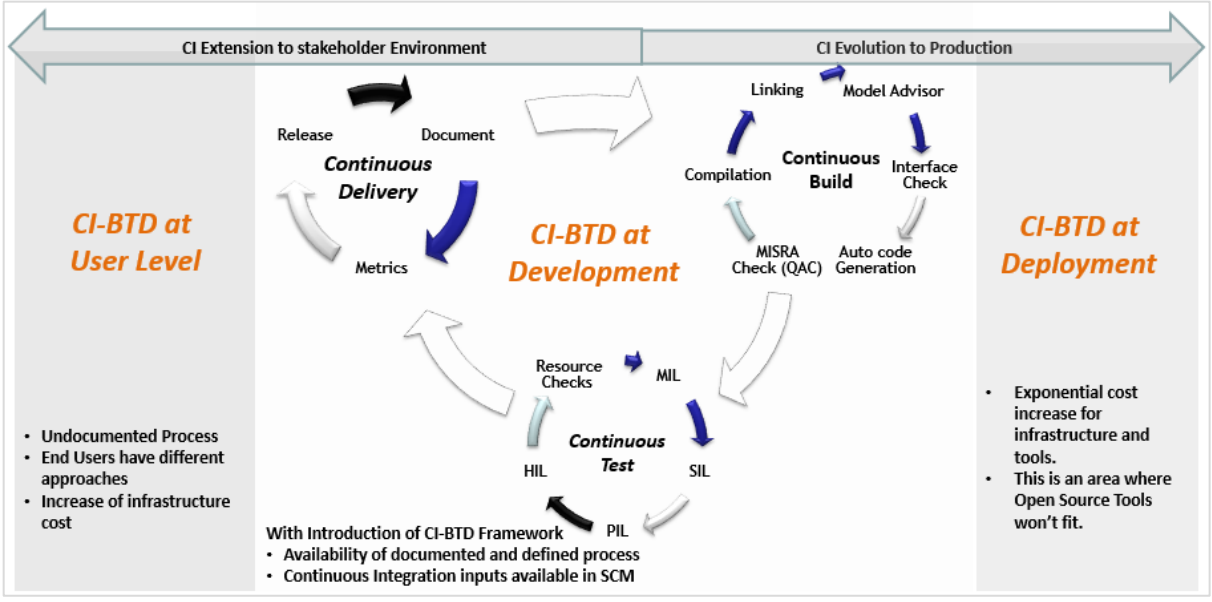

Figure 1 : CI-BTD Framework - The Big Picture

There are three strategic levels of CI-BTD framework which can be incorporated into any organization-

3.1. CI-BTD at Development

The best place to start the CI-BTD framework introduction is this level and in most organizations the CI-BTD also end into this level without going to other levels. This is the level where there is already process maturity and documentation available in any organization or departments. It will be clear what are the Quality checks or Gating criterions for the release of software.

The below can be considered as sub categories for this phase:

i. **Continuous Build:**
   Continuous build includes the Quality criterions or gating decision which can be directly executed on the software code. Depending on the organizations, different Quality Assurance (QA) stages can be added or removed.

ii. **Continuous Test:**
   This phase is executed mostly on the artefacts generated from the Continuous Build. Again the different test phases used will depend on the organization, but most of the use cases need testing on the binary or executable files generated from the continuous build.

iii. **Continuous Delivery:**
   This is the phase in the CI-BTD framework which will bind the results of the Continuous Build & Test phases. This phase is responsible for combining the Software changes from the software configuration management system and the feature or bug reference from the project issue management systems. This phase is also termed as the Release Management Automation with CI Framework. Creation of metrics, release notes, all required documentation and making the final software available for end users- is considered part of this phase.

3.2. CI-BTD at Deployment

This is the level where the software generated will be incorporated into a production environment continuously. For an automotive domain, this is the stage where software changes could be directly flashed into the ECU after certain quality criteria are passed. But there will be an exponential increase in cost to support the infrastructure required for this deployment. Also in safety critical applications like automotive domains, CI-BTD at deployment should be attempted only after complete maturity and reliability of the initial level.

3.3. CI-BTD at User Level

This level is an extension of the CI-BTD framework principle to the end users. CI-BTD at pre-development phase gets initiated when the code is available in the Software Configuration Management (SCM) Systems, but activities carried out in the manual way for the development stages by each developer separately. There exists undocumented processes with end users having different approaches during the development cycle. This level is also considered as a pre-step for CI-BTD framework with configuration for each developer separately. So to define a generic documented process which is a requirement for CI is difficult to be achieved at this level.

For the rest of this paper, we will concentrate on the CI-BTD at Development which is termed as CI-BTD framework. It is an ideal place for the CI-BTD to be introduced in any organization, if reliable CI-BTD is achieved that can be the base for the further levels.

4. Overview of CI-BTD Framework

In today's Agile Age CI-BTD framework (CI-BTD at Development) plays a critical role, wherein the test, support and development teams work closely and collaboratively to automate and streamline the build, test and release process. Iterations are short, manual handoffs are eliminated, risks are minimized, and releases are fast and reliable. As a result, a successful CI-BTD process can reduce the time-to-market for the product. The below figure shows the stages of SDLC covered using CI-BTD framework:

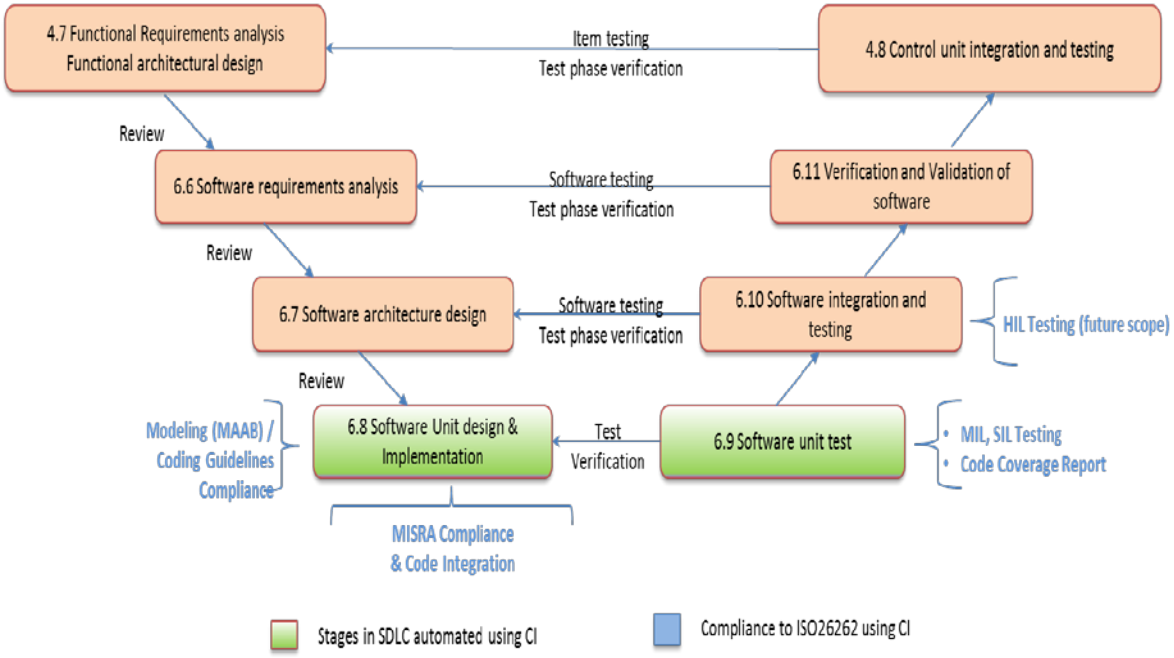

Figure 2: CI-BTD Framework in SDLC V-model

The below figure shows the high level CI-BTD framework topology along with some references of the tools used:

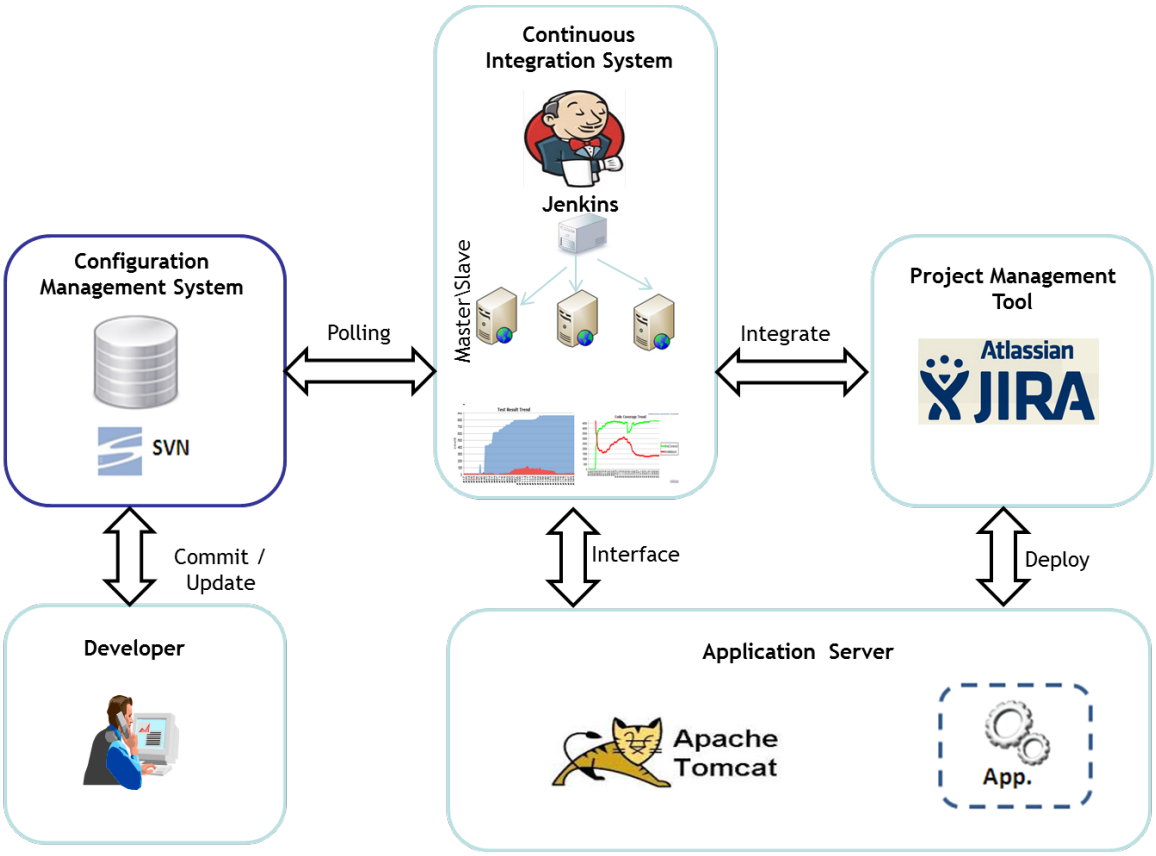

Figure 3: CI-BTD Framework Topology

5. CI-BTD Framework Approach

In L&T Technology Services, CI-BTD framework approach has been designed to create an automation environment ensuring that every change to the software results in a releasable version, and that any version can be built automatically at the push of a button. At a broader level, CI-BTD aims to make the entire end-to-end release process automated where the build software is delivered frequently with automated testing.

CI-BTD framework implements a set of principles and practices to ensure consistency and high quality by delivering fast feedback in order to reduce the cost, time, and risk of delivering incremental changes to users. The below figure shows the typical development, build and test cycle in CI:

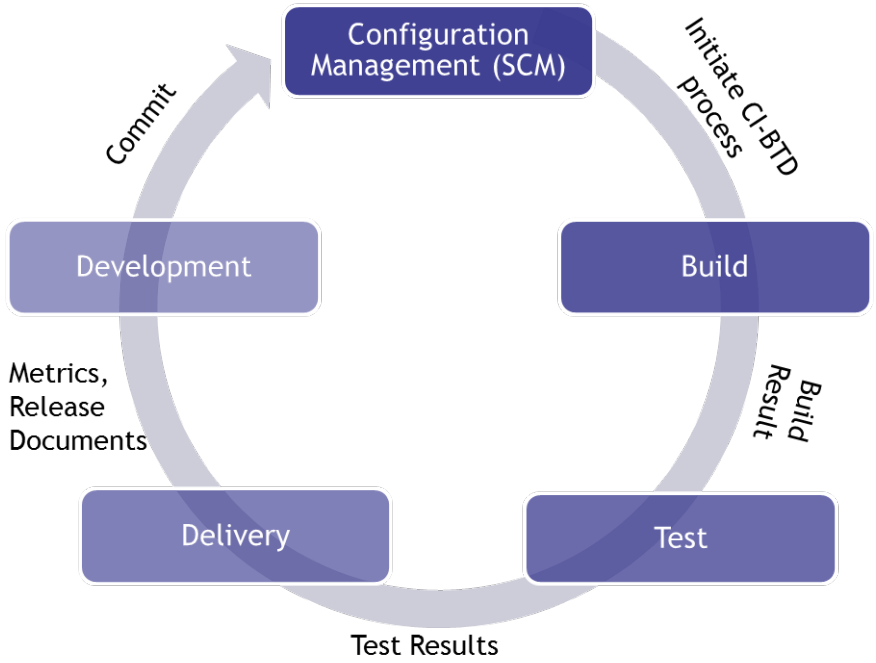

Figure 4: Typical CI-BTD framework cycle

CI-BTD framework pipeline consists of the following discrete steps:

i. **Development:** When a developer finishes a change in implementation, he or she commits it to a central source code repository (e.g. Subversion).

ii. **Build:** The change is checked out from the repository using Jenkins jobs and the software is built. In this phase, many quality gates are incoporated to get the compliance reports relevant to ISO26262 standards like modelling or coding guidelines, MISRA compliance, interface checks etc.

iii. **Testing:** This is where the change is tested against the set of test plans for the build software at different test levels to ensure that it works and that it does not break anything else. The tests are carried out at different environments such as MIL (Model-In-the-Loop), SIL (Software-In-the-Loop), PIL (Processor-In-the-Loop) and HIL (Hardware-In-the-Loop) providing back-to-back tests on the build software.

iv. **Delivery:** This is the stage in which software release, documentation and metrics reports are generated. Metric reports for memory and CPU utilization are generated.

CI-BTD framework is a practice that definitely helps developers identify the errors at an early stage. It is a practice of integrating, whenever a developer makes any changes and check-in to source code repository, automatically and continuously it starts the process such that no errors can arise without the developers noticing them. The below figure shows the CI-BTD framework workflow:

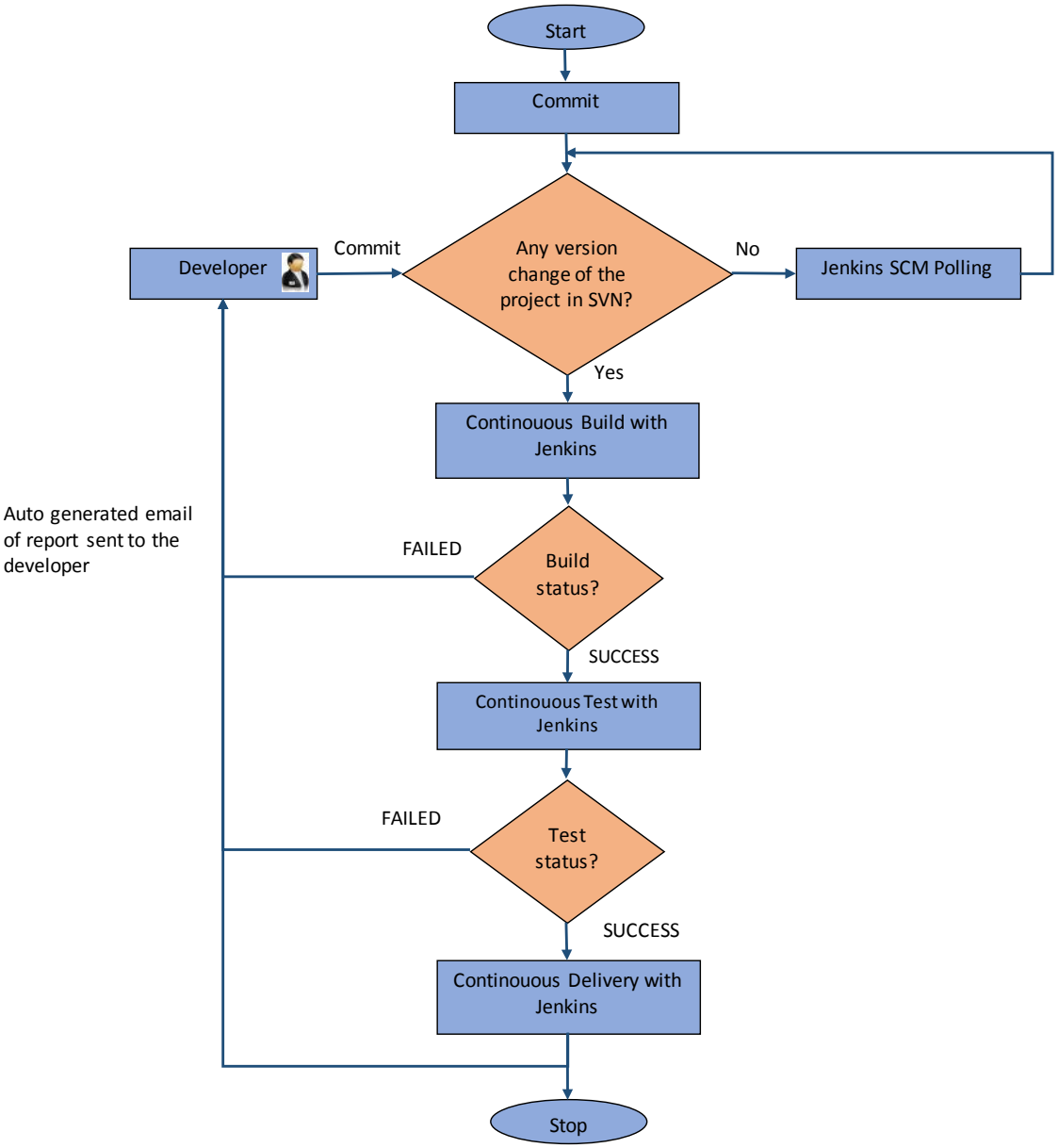

Figure 5: CI-BTD Framework WorkFlow

CI-BTD workflow is implemented to support the three types of builds:

i. **SCM Build (Commit Builds)**: It is triggers whenever any changes are committed in the source control repository in the component / module. This build is designed to execute in the best possible time and with highest frequency. The implementation considers whether the changed code is good for the build along with basic sanity testing.

ii. **Nightly Build (Full Builds):** It occurs when developer wants to validate all the modules at a time. This is designed to run for a longer duration where all the components are validated along with any special consideration including the regression testing of the Tool chain changes and architectural changes.

iii. **Release Builds (Delivery Builds):** This is a promotion build which can be executed on the Full Builds which automates the software delivery process on a qualified build. The build is responsible for documentation, Metrics creation, Project management system update along with the release of the qualified software for end stakeholders.

When the code needs to be production-ready at all times, managed by strict version control policies and the development team adopts an agile development with back-to-back testing practices, organizations can move to implement CI-BTD framework. The below figure explains the flow of stages involved in a CI-BTD framework:

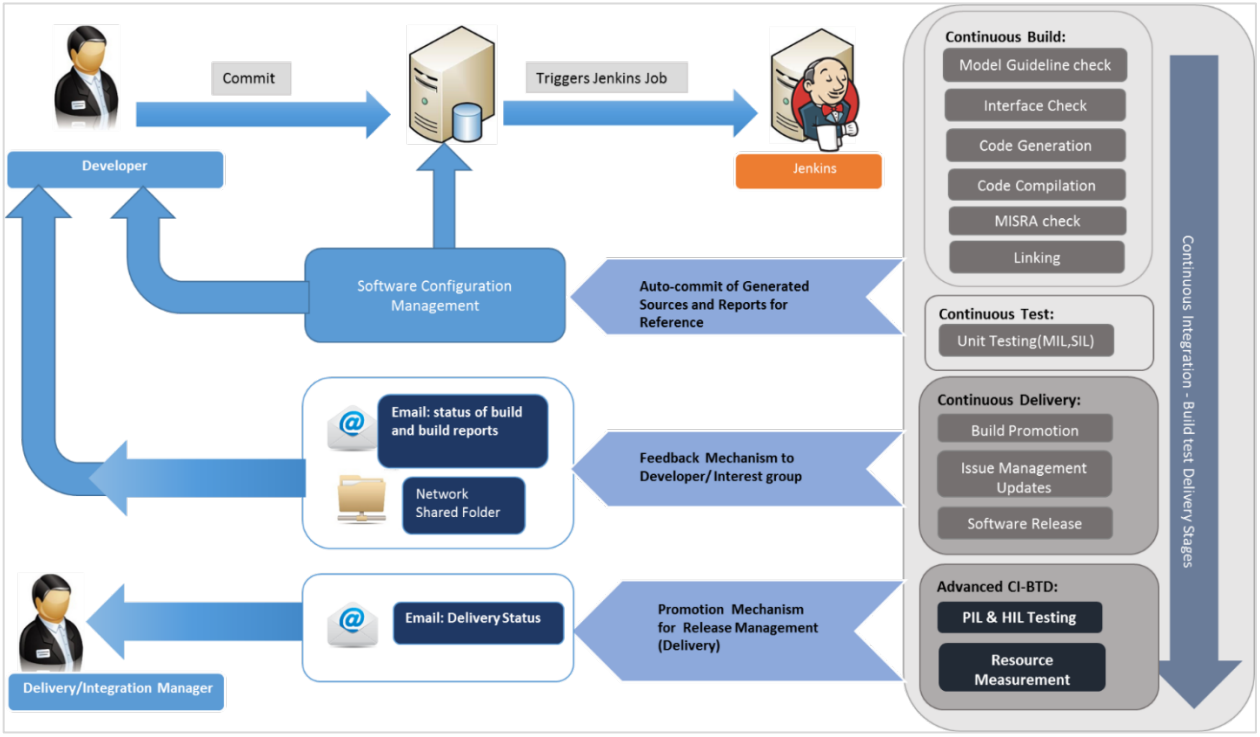

Figure 6: CI-BTD Framework Architecture

To illustrate a CI-BTD framework workflow, consider the scenario:

i. A repositoy is where all the stabilized or baselined work is comitted, this is called mainline or trunk in a subversion repository.
ii. A developer 'checks-out' the mainline in his working environment which is called working copy or subversion branch. The developer starts working and makes changes.
iii. All the changes made need to be integrated and tested, as soon as the developer commits the changes to a subversion repository. From here the CI-BTD framework process is initiated.
iv. Jenkins detects the changes in the subversion repository and the pre-configured Jenkins job is triggerred automatically.
v. During the execution of Jenkins jobs, quality gates are verified for the following scenarios:
    a. **Modelling Guidelines Check**: In this stage, the modelling guildelines are verified using Matlab for MAAB compliance. The MATLAB guildeines compliance report is then generated.
    b. **Interface Check**: In this stage, the inputs and outputs of the signals are compared in the Matlab against the data dictionary and interface check report is generated.
    c. **Autocoding Check**: In this stage, the auto-code generation is invoked and verified for the successful code generation using defined configuration.
    d. **Compilation**: In this stage, the auto-code compilation is carried out and checked for any errors.
    e. **MISRA Compliance Check**: In this stage, MISRA C coding guidelines are checked using QAC with generation of the QAC report.
    f. **Linking**: In this stage, all the objects files are combined together to create the hex and a2l files to be flashed into the ECU.
    g. **Unit Testing**: The unit testing in MIL (Model-in-the-Loop) and SIL (Software-in-the-Loop) environments is carried out against the test cases. The tests are run in Matlab and Tessy environments and reports are generated for the coverage based on statement (C0), branch (C1) and MCDC configuration.
vi. During the execution of above stages, email will be sent to the developer as either passed or failed, alongwith link for the execution status of each stage for the detailed report.

vii. JIRA- a planning and ticketing tool is integrated into the Jenkins CI-BTD framework. The respective JIRA ticket state is updated based on the results of the Jenkins jobs.

viii. For the errors reported, the developer performes the analysis and rework for the reported issues. Then the changes are re-commited to the working copy or subversion branch. Then again the Jenkins job will be triggered and the process is repeated.

ix. Once the changes are stabilized in working copy or subversion branch, it is integrated into mainline or trunk and Jenkins job is triggered and the process is repeated in the mainline or trunk.

The new feature adopted in CI-BTD framework is the tracing and debugging which helps to provide the information of the currently running Jenkins jobs in different stages of its execution.

6. ISO26262 Compliance using CI-BTD Framework

CI-BTD framework is developed to integrate tools such as Matlab with Model Advisor and Embedded Coder, Compilers, QA-C and Tessy, in order to provide the compliance for respective stages of the SDLC. This helps to provide quicker and faster results repeatedly during each run of the Jenkins jobs with reports creation at the end of each stage. The CI-BTD framework could also be extended for future requirements and integration of organization specific tools.

The figure “CI-BTD framework in SDLC V-model” in section “Overview of CI-BTD Framework” outlines the ISO26262 compliance being incorporated into CI-BTD framework mainly towards part 6 for stages of 6.8 and 6.9, software unit design and implementation and software unit test, respectively. During these stages of the SDLC, CI-BTD framework supports the compliance for modelling or coding guidelines, MISRA checks and code integration, MIL and SIL testing with code coverage reports generation. The below figures outline the compliance ensured by CI-BTD framework using various tools integration into the framework during the various stages of the SDLC with respect to part 6 of ISO26262.

Table 8 — Design principles for software unit design and implementation

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	Tools integrated in CI-BTD framework for ISO26262 Compliance
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++	✓ QA-C
1b	No dynamic objects or variables, or else online test during their creation[a,b]	+	++	++	++	
1c	Initialization of variables	++	++	++	++	
1d	No multiple use of variable names[a]	+	++	++	++	
1e	Avoid global variables or else justify their usage[a]	+	+	++	++	
1f	Limited use of pointers[a]	o	+	+	++	
1g	No implicit type conversions[a,b]	+	++	++	++	
1h	No hidden data flow or control flow[c]	+	++	++	++	
1i	No unconditional jumps[a,b,c]	++	++	++	++	
1j	No recursions	+	+	++	++	

Table 9 — Methods for the verification of software unit design and implementation

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	Tools integrated in CI-BTD framework for ISO26262 Compliance
1a	Walk-through[a]	++	+	o	o	✗ Human Intervention Required
1b	Inspection[a]	+	++	++	++	
1c	Semi-formal verification	+	+	++	++	✓ Model Based Design (Matlab, Simulink, E-Coder)
1d	Formal verification	o	o	+	+	
1e	Control flow analysis[b,c]	+	+	++	++	✓ Advanced CI-BTD (Polyspace)
1f	Data flow analysis[b,c]	+	+	++	++	
1g	Static code analysis	+	++	++	++	✓ QAC
1h	Semantic code analysis[d]	+	+	+	+	✓ Compilers

Table 10 — Methods for software unit testing

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	Tools integrated in CI-BTD framework for ISO26262 Compliance
1a	Requirements-based test[a]	++	++	++	++	✓ MIL-SIL Test Framework
1b	Interface test	++	++	++	++	
1c	Fault injection test[b]	+	+	+	++	✓ Advanced CI-BTD
1d	Resource usage test[c]	+	+	+	++	
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++	✓ Test Framework

Table 11 — Methods for deriving test cases for software unit testing

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	Tools integrated in CI-BTD framework for ISO26262 Compliance
1a	Analysis of requirements	++	++	++	++	✓ Tessy & MIL-SIL Test Framework
1b	Generation and analysis of equivalence classes[a]	+	++	++	++	
1c	Analysis of boundary values[b]	+	++	++	++	
1d	Error guessing[c]	+	+	+	+	

Table 12 — Structural coverage metrics at the software unit level

	Methods	ASIL A	ASIL B	ASIL C	ASIL D	Tools integrated in CI-BTD framework for ISO26262 Compliance
1a	Statement coverage	++	++	+	+	✓ Tessy & MIL-SIL Test Framework
1b	Branch coverage	+	++	++	++	
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++	

Figure 7: ISO26262 compliance using CI-BTD framework

Source: Road vehicles — Functional safety — Part 6: Product development at the software level

7. CI-BTD Framework Benefits

A key benefit of a CI-BTD framework is that the software will have a always stable build, with back-to-back automated testing and ready for delivery. Many of the industry challenges can be resolved by making CI-BTD framework as part of the SDLC process workflow. Some of the challenges being resolved using CI-BTD framework during SDLC process are outlined as below:

i. Getting the immediate feedback of the changes being carried out and reporting the same to the developer
ii. Consistency and high quality of the deliverables with quality gate checks and metrics reporting
iii. Back-to-back continuous testing for the changes in the subversion repository
iv. Reduction in manual effort and cost
v. Faster and stable releases with errors detected early in the development cycle
vi. Higher model or code quality with automated documentation of quality metrics
vii. Employee motivation by delegating manual & monotonous activities to machines.
viii. Readily available Metrics on the Code quality and feature deliverance.

The below figure shows the statistics analysis earlier and after CI-BTD framework implementation:

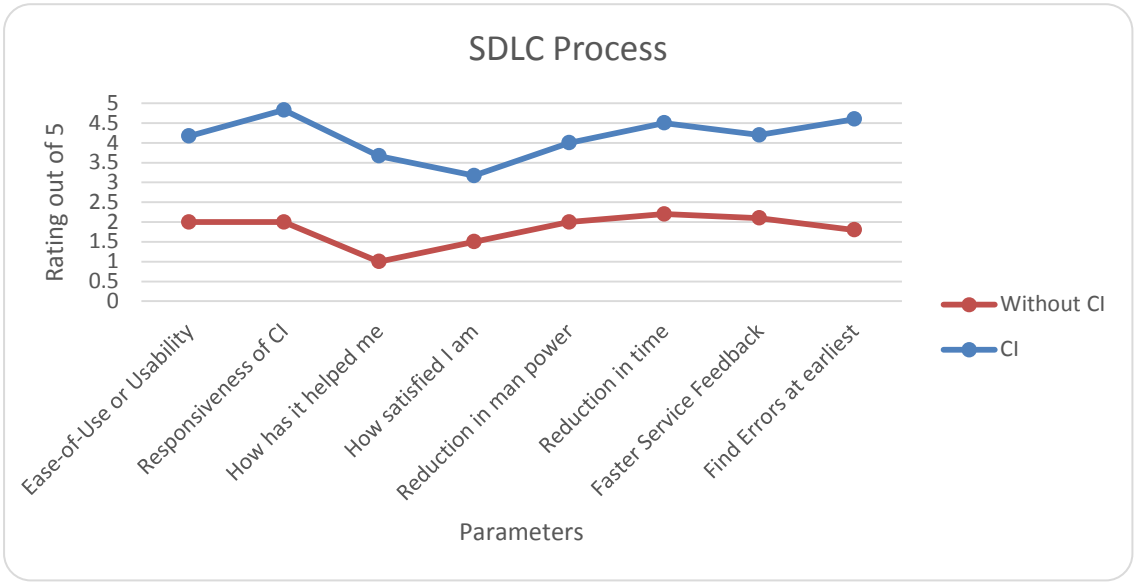

Figure 8: Statistical Analysis of SDLC

8. Advanced CI-BTD Framework

A CI-BTD framework is continously getting upgraded with new features and tools integration to support across different platforms. The advanced CI-BTD framework will be incoporated with the following features and relevant tools integration:

i. Integration of dynamic analysis tool like Polyspace
ii. PIL Testing using Tessy with flashing of code on the ECU
iii. HIL Testing in dSpace environment
iv. Resource measurement and metrics generation for RAM, ROM, Stack & CPU utilization
v. Timing analysis and run-time measurements

9. Conclusion

In today's environment, it is critical to act and react fast in order to keep up with the rapid pace of change in SDLC and tight competition landscape within the automotive industry. As such, CI-BTD framework is more than just a new working process, it is a strategic capability that provides organizations with a strong competitive advantage. This 'advantage' is rapidly becoming the new benchmark. Companies that do not adopt agile processes like CI-BTD framework risk falling beneath in today's standards of acceptability. With the usage of fully automated process into the software development lifecycle, there is no room in agile processes for manual handoffs and quality compromises. The continuous test and continuous delivery will pave the way for the future automotive CI-BTD framework process. CI-BTD framework is a need of time for complex projects due to the benefits it provides regarding early detection of errors and flexibility to integrate different tools and quickly adopt into pre-existing development environments.

10. References

i. Continuous Integration: Improving Software Quality and Reducing Risk by Paul M. Duvall with Steve Matyas and Andrew Glover
https://www.assembla.com/spaces/cie/documents/c5Sfa6Lq8r3yLWab7jnrAJ/download/c5Sfa6Lq8r3yLWab7jnrAJ
ii. Continuous Software Engineering a book edited by Jan Bosch
https://books.google.com/books?id=JURTBQAAQBAJ&pg=PA2&lpg=PA2&dq=Continuous+Software+Engineering+a+book+edited+by+Jan+Bosch&source=bl&ots=pVaxEdLSW0&sig=A2GzX_6aXPdGWqEpf135Puki00I&hl=en&sa=X&ved=0CB4Q6AEwAmoVChMIl__F7K_2xwIVx9QaCh29bQsN#v=onepage&q=Continuous%20Software%20Engineering%20a%20book%20edited%20by%20Jan%20Bosch&f=false
iii. Road vehicles — Functional safety — Part 6: Product development at the software level
https://www.iso.org/obp/ui/#iso:std:iso:26262:-6:ed-1:v1:en

11. About The Authors

Manish Patil

Manish Patil has 13 years of industry experience in the design, development and validation of embedded systems majorly with automotive domain. He is currently part of Transportation Business Unit in L&T Technology Services Limited focussed on Embedded Systems solutions in MBSE, ISO26262, Continuous Integration and Software Development for the automotive industry.

Sathishkumar T

Sathishkumar T has 4 years of industry experience in continuous integration. He is currently part of Transportation Business Unit in L&T Technology Services Limited focussed on Continuous Integration solutions for the automotive industry.

About L&T Technology Services

L&T Technology Services is a wholly-owned subsidiary of Larsen & Toubro with a focus on the Engineering Services space, partnering with a large number of Fortune 500 companies globally. We offer design and development solutions throughout the entire product development chain across various industries such as Industrial Products, Medical Devices, Automotive, Aerospace, Railways, Off-Highway & Polymer, Commercial Vehicles, Telecom & Hi-Tech, and the Process Industry. The company also offers solutions in the areas of Mechanical Engineering Services, Embedded Systems & Engineering Application Software, Product Lifecycle Management, Engineering Analytics, Power Electronics, and M2M and the Internet-of-Things (IoT).

With a multi-disciplinary and multi-domain presence, we challenge ourselves every day to help clients achieve a sustainable competitive advantage through value-creating products, processes and services. Headquartered in India, with over 10,000 highly skilled professionals, 12 global delivery centres and operations in 35 locations around the world, we constantly find flexible ways of working, tailored to our assignments and customer needs.

For more information, visit us at www.lnttechservices.com