2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)

Continuous integration in validation of modern, complex, embedded systems

1st Mateusz Kowzan
Intel Technology Poland
Intel Corporation
Gdańsk, Poland
mateusz.kowzan@intel.com

2nd Patrycja Pietrzak
Intel Technology Poland
Intel Corporation
Gdańsk, Poland
patrycja.pietrzak@intel.com

***Abstract* — This paper covers the challenges in software validation, especially in the context of complex, embedded systems. Problems related to limited validation environment resources (DUT - Devices Under Test, workload and traffic generators, measurement setups, test probes), stability and complexity of systems will be highlighted. Authors will describe validation strategy which responds to quality assurance and risk management requirements applied for real products provided to the market. Presented technologies covers solutions based on Continuous Integration and Continuous Delivery process of modern, complex, embedded systems. Authors present both known and new solutions and approaches to known challenges, also suggest possible further development directions to improve efficiency and optimize product development cost.**

Keywords* — *Continuous Integration, Continuous Delivery, Build Verification Tests, Basic Acceptance Tests, Validation Strategy, Automation, Embedded Systems, Software Quality Assurance.

I. Introduction

The production of modern products based on embedded systems is a multi-stage and complicated process. The continuous increase of the level of product complexity have been observed from many years. This trend forces changes in product development process. Advanced systems consists of numerous different components like hardware, firmware or software. Each of them is usually delivered by a separate team of engineers (who often works in another location across the world). Often schedules of releases for system sub-components are not synchronized with each other. From the manufacturer's point of view, the process of delivering a product that meets the defined quality expectations that are defined for each phase of the project is not trivial. Even the successful validation of individual system components does not guarantee the achievement of quality expectations at the system level, where we deal with specific interactions between components including critical time dependencies. There is also second aspect: product validation process increases product development cost. So even if designed strategy allows to meet quality requirements, there is still a room to optimize the cost and/or full validation cycle execution time. The methodology that meets such scenarios is the Continuous Integration and Continuous Delivery approach [1-3]. The very concept is that the team responsible for the integration of all delivered components works on system validation from the very beginning of the product definition and development process. Early engagement of validation team helps to track product quality from the beginning and react appropriately (identifying and fixing the defects in the early phase is much less expensive than at a later time). With the availability of subsequent versions of system components (theoretically of higher quality), the team cyclically collects a set of the latest versions of components and creates the so-called Best-Known-Configuration, on which it bases its validation work. The above approach coincides with popular, modern product development methodologies such as Agile Software Development [4] or Kanban Board. All kinds of problems at the system level are quickly identified, which makes the work of teams providing components is more predictable.

In many industries, the market requires delivery to various clients of the same system in different configurations, or the same system that works with various external types of hardware, components or environments (e.g. customer infrastructure, often in different versions, which requires backward compatibility). For this reason, development teams often apply Trunk Based Development methodology. In such approach there is a main master branch in code repository, the so-called Trunk, which contains all product versions, usually (but not in all cases) from the same generation. Additionally, for a short time before release (or milestone) a release branch is created in order to focus on quality assurance. Fig. 1 depicts a code repository model in the Trunk Based Development approach. Such code repository model allows to decrease code development cost. However, validation of Trunk builds requires testing for all products contained in Trunk, what increases cost of validation.

Another common approach to the system development process is the so-called V-model. This model assume that the system design is created from general to detail and the system integration is made from detail to general. Important aspect of such approach is that the feedback information about system quality from system integration is quickly obtained and are included in system design process. Fig. 2 depicts an example of V-model of the system development process. This approach has significant impact on practical projects. Defects made at early development stages will be populated to the next phases. The cost of defects fixing rises with time of development and becomes tremendous when are detected after product release to the market.

High volume validation of modern, complex embedded systems requires considerable amount of hardware and human resources, and also a significant amount of time (in comparison to the time needed to manufacture the product). Taking this into account, as well as complexity of modern, embedded systems, number of potential scenarios that can happen in field, developed by many groups of engineers from different locations working simultaneously on the same code, it is easy to imagine that credible validation strategy should be based on many dedicated tests performed frequently on several DUTs in well controlled test environment. From product development efficiency perspective it is crucial to automate the execution of as many tests as possible. Massive

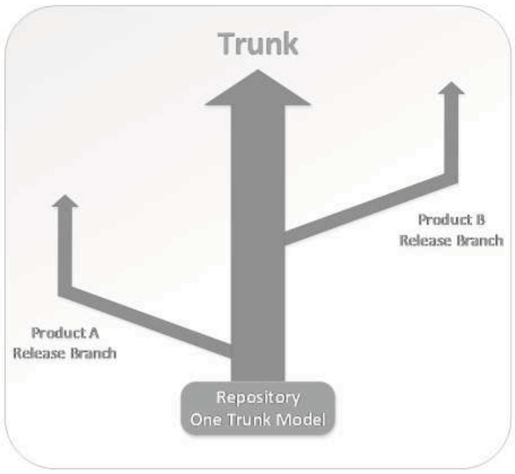

Fig. 1. Code repository model in the *Trunk Based Development* approach.

test execution automation process will be covered as crucial element to improve quality assurance process.

However, even with 100% automation of tests, some of the challenges such as optimizing the cost of maintaining the test environment (among others high cost of hardware and its maintenance) or the time of the single validation cycle are still valid. All kinds of savings in the field of hardware maintenance or time of the validation process clearly translate into the producer's profit and the time of product delivery to the market.

In this paper authors will try to address the following high volume embedded systems validation challenges:

- *product scale and complexity,*
- *high amount of daily code changes,*
- *test environment configuration multiplicity,*
- *limited resources.*

II. Solution

The base required in order to overcome the challenges in high volume validation of modern, complex, embedded system, is automation. Created validation environment is based on fully automated tests. Implemented Continuous Integration and Continuous Delivery process consists of 5 stages of testing:

- *Unit Tests (UT),*
- *Build Verification Tests (BVT),*
- *Basic Acceptance Test (BAT),*
- *Nightly Tests Environment (NTE),*
- *First Article Inspection (FAI).*

Each type of testing stage is different from other in terms of its purpose and test suite (various amount and types of tests). Real life example of innovative Continuous Integration and Continuous Delivery process for firmware development for embedded system is presented in Fig. 3.

The Continuous Integration and Continuous Delivery model has been created to increase quality of products. This

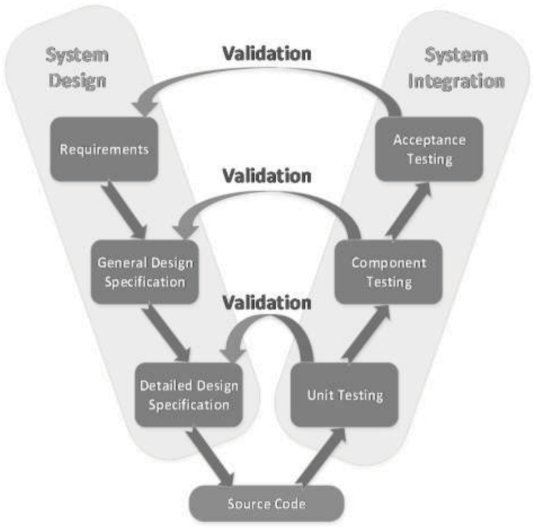

Fig. 2. An example of *V-model* of the system development process.

new process could bring results only if stability of environment will be on very high level. For this reason lots of effort was put to create automatic self-recovery procedures in case of hardware or software failure and implement proper redundancy policy. During whole process stability metrics are gathered and analyzed in terms of further environment improvements.

A. Pre-review buid check

Generating pre-review build takes place even before the code review process, but immediately after the code commit. Above all, this action is aimed at checking the correctness of the code compilation. At this point we are also able to start the initial testing.

Not every code change or code fix needs a generation of the pre-review build, but there are cases when the use of pre-review build allows to get information about changes as soon as possible or eliminates various errors form the whole process at the very beginning.

B. Engineering builds integration

Developer is pushing his code into separate engineering branch in version control system. This event is noticed and information is passed to continuous integration application. It does code compilation from this branch for all supported products. If all compilations are successful request is sent for code review. Review is done manually by experienced developers (experts) who checks if code is written properly and is compliant with coding style. If review is accepted continuous integration application is one more time compiling branch on top of latest master and informs test management framework to start BVT scope for all supported products. Scope is basic, so that all tests do not last more than 20 minutes. The main purposes of BVT scope is to verify that product starts up properly and all interfaces and communication channels are working. BVT ends with one of two results:

- *PASSED - all tests from BVT scope passed for all supported products:*

  Continuous integration application is integrating engineering branch to master branch and notified

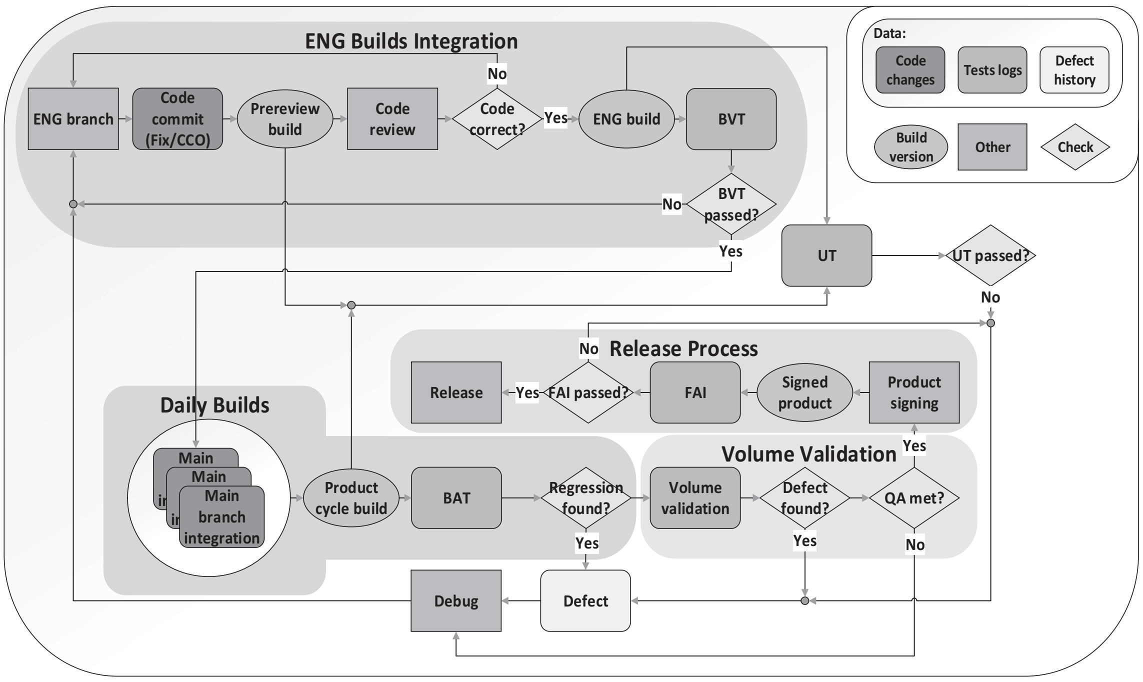

Fig. 3. Real life example of innovative *Continuous Integration and Continuous Delivery* process for firmware development for embedded system.

developer that his commit is integrated with master branch.

- *FAILED - at least one of tests from BVT scope failed for any supported products:*

  Continuous integration application notifies developer that commit failed and his actions is required. In addition an email is send to developer with detailed information about failure. Commit will be not integrated.

In this stage of integration process has been implemented one of the most important rule of this model. It assume, that all commits are integrated as engineering builds in serial queue, which is stored in continuous integration application and it contains not integrated commits with positive result of review process. Queue is scheduled by first in, first out (FIFO) rule. However, there is possible to manually change the sequence of commits in the queue, if needed.

C. Daily builds

Every few hours (frequency depends on project stage, closer to major release frequency is higher) latest master branch is compiled for all supported products by continuous integration application. Each daily build contains set of engineering builds created during BVT with positive result. After this, notification is sent to test management framework to run BAT scope on compiled products images. When tests are finished is performed an analysis of the results searching for potential regressions. At this stage there are 3 options:

- *Regressions not found:*

  No actions are taken.

- *Regression found:*

  Tests which found regression are rerunning sequentially on engineering builds contained in the daily build until engineering build which introduced regression is found. Defect is created and is assigned to developer who introduced regression in engineering build. Then email notification is sent to developer with information that he has 24 hours to fix or to remove faulty commit.

- *Regression from previous build is fixed in current build:*

  Status of defect which was created for this regression is changed to Verified.

BAT scope could be bigger than BVT scope even by order of magnitude. It mainly contains acceptance tests and is use as regression scope. Average duration of BAT is between 5 to 8 hours, depends on product.

D. Volume validation

Every night, for every product, NTE is started on last daily build corresponding to specific product. NTE scope could contains much more tests than BAT scope (even by order of magnitude). However, increased tests scope requires a lot of more hardware and human resources to be are involved. This is the most accurate tests scope. It could be so big that it is necessary to schedule it on couple of NTE runs (it means, couple nights), in parts. After the tests are finished, same as in the case of BAT, the results are analyzed and searched for potential regressions.

NTE is a powerful, but long lasting process. It also requires plenty of resources and because of that it is used by night when most of resources are unoccupied. NTE is a kind of regular validation approach.

E. Release process

It is an indispensable part of testing every release build. It is required to check if uploaded build package is ready to be download, has correct contents and license, and if release collaterals are double checked. This tests are crucial to release signed product version. FAI tests scope is rather small and usually do not last more than one hour. Because of many complex flows which assumes hardware configuration changes in scope of single test case this part of process is hard to fully automate.

F. Unit Tests

To maintain high quality of the code UT are developed as an integral ingredient of the project and covers each line of the code. Some of them are auto-generated by VectorCAST framework. UT are performed automatically for each engineering build and each daily build compilation. They are also performed during compilation of pre-review builds. As well as BVT, UT protect code repository against faulty commits. In case of any failure in UT scope, the integration is stopped.

III. Results

Introducing Continuous Integration and Continuous Delivery as a process to product testing process shifted left regression finding burden from regular testing approach such as Nightly Tests Environment. Because automatic mechanisms such as Build Verification Tests and Basic Acceptance Tests reduced time needed to identify and fix issues introduced by code and/or components, and/or hardware changes. It allows to make huge savings in project development and time to market, and also time needed to release fixed issues to customers.

So far, introduced part of designed Continuous Integration and Continuous Delivery process results in:

- *faster and more efficient defect detection,*
- *improved product quality assurance,*
- *increased development and validation teams performance.*

IV. Further Steps

Many processes in Continuous Integration and Continuous Delivery model in the validation of embedded systems already have been implemented. However, there are still areas that could be even more automated. In this section are listed ideas for further development.

A. AutoBAT

AutoBAT will introduce a brand new approach. The design has been already defined. Next stage is implementation of AutoBAT service which will automate the steps that until now have been manually raised. AutoBAT service would be aimed at automating the search for regression, reporting and verification process. Each daily build contains set of commits included in it. Each of those commit is also represented by engineering build created during BVT run on it. AutoBAT, based on current and last BAT results, will be looking for regressions. When regression is found in BAT, continuous integration application will start AutoBAT, which will try to find shortest reproduction steps. In most cases only single test is required to reproduce a regression, but occasionally combination of two or more are required. After finding reproduction steps AutoBAT will extracting set of engineering builds included in daily build and will run reproduction steps on each engineering build (starting from the oldest) until it finds the one that is introducing regression. Defect will be automatically created with proper reproduction steps and will be assigned to author of commit that introduced regression.

The next step could be an email notification sending to author of commit with information, which code changes needs to be reverted or fixed in 24 hours. This information will be verified in next BAT run. If regression is removed, the defect status will be changed to Verified.

Innovative system, which is AutoBAT, allows to reduce human involvement in regression recognizing process almost completely and to increase level of automation of whole system.

B. External Components Integration

Defined design of a new process that will integrate external components is based on known solutions such as BAT. It will be used to verify the product resulting from the combination of several different elements.

Implementation will provide that the latest product build package will be validated against the external component. They will be built and as a pre-review build would be tested. Here could be used AutoBAT process to verify the result of this integration. Base on the results analysis, it would be possible to decide whether the integration was successful and whether each of the components is correctly co-operating with each other.

C. Test suite analysis and optimization

The aim is to develop methodologies and algorithms that allow creating an effective validation strategy, optimal for the current phase of product development and a set of changes that were introduced in comparison with the previously validated version, guaranteeing achievement of the expected level of product quality with assumption of limited validation resources and acceptable time for a single validation cycle. This solution should meet the requirements of the real life product development environment, in which components are delivered cyclically at intervals that are not synchronized with each other.

In order to solve the problem, a detailed analysis of the requirements is necessary. On the basis of the analysis results, will be created intermediate metrics, on the basis of which it will be possible to assess the quality of the system that generates optimal test strategies.

Elements on the basis of which one can make a decision about the selection of a valid validation strategy are data about:

- *the results of previous test cycles,*
- *a list of changes made to components of the validated system,*
- *availability of validation resources,*
- *a list of available tests,*

- *historical data regarding changes and defects that were identified after their introduction.*

As part of the work, an analysis of the above factors in the context of their significance will be carried out. Will be designed and implemented the architecture of the expert system generating the optimal strategy. The implemented solution will be verified cyclically, and after obtaining the expected results, the system will be implemented in the production environment. The implemented solution will be verified and improved cyclically, and after obtaining the expected results, the system will be fully implemented in the production environment.

V. Summary

High volume validation of modern, complex embedded systems is a challenging topic. Process complexity comes from multiplicity of possible ingredients configurations. Nowadays trends forces changes in product development process. Crucial elements, like hardware accessibility, total cost of operation and time to market have to be reconsidered.

So far, implementation of Continuous Integration and Continuous Delivery process is usable and stable, but it is still under development and improvement process. However, main features, such as BVT, BAT and FAI have been implemented completely. Furthermore exist complete design and partial implementation of AutoBAT and External Components Integration process.

Introduced Continuous Integration and Continuous Delivery model in validation of modern, complex, embedded systems provides quick feedback information about new regressions and overall product quality. It also significantly increased the level of system automation, which resulted in faster and more efficient defect detection, improved product quality assurance and increased development and validation teams performance. Solution is scalable and portable to any firmware or software development project.

Current implementation of presented solution addresses mentioned high volume embedded systems validation challenges. High amount of daily code changes validation is addressed by UT and BVT processes which protect code repository from faulty changes. NTE and BAT processes help to manage the scale and complexity of product. NTE covers the multiplicity of test environment configuration. BAT provides quick feedback about introduced regressions and allows to revert faulty code change immediately and efficiently in automated way. In future, AutoBAT solution will decrease time required to identify faulty code changes. Finally External Components Integration flow will simplify and automate integration process on system level.

In next steps improvements and new features, such as AutoBAT and External Components Integration process, will be implemented and integrated with Continuous Integration and Continuous Delivery system in order to increase system automation level and decrease total cost of operation. Correctness and efficiency of implemented model of Continuous Integration and Continuous Delivery solution will be evaluated. Also, further analysis and optimization research will be perform, to allow a precisely definition of test suite. Definition of suitable test scope will optimize the amount of tests necessary for quality assurance with limited resources.

Acknowledgment

We would like to acknowledge all teams and individuals from Intel Technology Poland for their help, support and contribution to design and implementation of this Continuous Integration and Continuous Delivery model.

References

[1] "A Brief History of DevOps, Part III: Automated Testing and Continuous Integration". CircleCI. 2018-02-01.

[2] P. M. Duvall "Continuous Integration. Improving Sotfware Quality and Reducing Risk" Person Education Inc., 2008.

[3] V. Melymuka "TeamCity 7 Continuous Integration Essential" Packt Publishing, December 2012.

[4] Gruver G., Young M., Fulghum P. "A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware", Addison-Wesley Professional, November 2012.