Chapter 4. Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences

This chapter is currently in press:

Abstract
In this paper we present a summary of factors that must be taken into account when applying continuous integration to software-intensive embedded systems. Experiences are presented from two study cases regarding seven topics: complex user scenarios, compliance to standards, long build times, many technology fields, security aspects, architectural runway and test environments. In the analysis we show how issues within these topics obstruct the organization from working according to the practices of continuous integration. The identified impediments are mapped to a list of continuous integration corner-stones proposed in literature.

4.1 Introduction

Continuous integration is widely promoted as an efficient way of conducting software development. The practice is said to enable that tests can start earlier, that bugs are detected earlier and to increase developer productivity [Duvall 2007, Ståhl 2013]. Martin Fowler's popular article [Fowler 2006] is often referred to as a summary of the practice of continuous integration. Paul Duvall summarizes continuous integration in a similar way into a list of seven corner-stones [Duvall 2007]. The corner-stones (here labeled C1-C7) are presented in Table 13. Applications of continuous integration and other agile practices on large, complex systems have been presented by Craig Larman and Bas Vodde [Larman 2009] and Dean Leffingwell [Leffingwell 2010]. There are also reports describing various experiences from introducing continuous integration practices, often together with other agile practices [Downs 2010, Karlström 2002, Miller 2008, Roberts 2004, Stolberg 2009]. However, these reports do not describe experiences from applying continuous integration to software-intensive embedded systems (software systems combined with electronic and mechanical systems).

The topic of this paper is an overview and discussion of factors specific for software-intensive embedded systems that could constrain a full adaptation of continuous integration (as defined by Duvall's corner-stones). The authors of this paper have under a long period of time been involved in software development projects for large-scale and complex systems. Through our work in various roles related to integration and testing, we have gained experiences of problems and issues related to the practices of continuous integration.
The main contribution of this paper is a summary of factors that we believe must be taken into account when applying continuous integration to software systems combined with electrical and mechanical systems. Further work could examine solution approaches that can be applied in multiple case-studies.

The remainder of this paper is organized as follows. In the next section the study cases are described. Subsequently in Section 4.3 we present the problems and issues that we have experienced regarding seven topics. In Section 4.4, we present an analysis of how the topics described in Section 4.3 are related to the corner-stones for continuous integration that were presented in Section 4.1. The paper is concluded in Section 4.5 where we summarize those relationships.

<table>
<thead>
<tr>
<th>Id</th>
<th>Continuous Integration Corner Stone</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>All developers run private builds on their own workstations before committing their code to the version control repository to ensure that their changes don't break the integration build</td>
</tr>
<tr>
<td>C2</td>
<td>Developers commit their code to a version control repository at least once a day</td>
</tr>
<tr>
<td>C3</td>
<td>Integration builds occur several times a day on a separate build machine</td>
</tr>
<tr>
<td>C4</td>
<td>100% of tests must pass for every build</td>
</tr>
<tr>
<td>C5</td>
<td>A product is generated that can be functionally tested</td>
</tr>
<tr>
<td>C6</td>
<td>Fixing broken builds is of the highest priority</td>
</tr>
<tr>
<td>C7</td>
<td>Some developers review reports generated by the build, such as coding standards and dependency analysis reports, to seek areas for improvement</td>
</tr>
</tbody>
</table>

Table 13: Duvall's seven corner stones of continuous integration.

### 4.2 Case Study Companies

In order to discuss impediments for continuous integration, we will compare experiences from two study cases, which both are companies developing large-scale and complex software for products which also include a significant amount of mechanical and electronic systems.

#### 4.2.1 Study Case A

Study Case A is a telecommunications company with a wide range of products that serves the B2B market. The products are highly software-intensive, but also include significant electronic and mechanical parts.

Study Case A has an advanced system of automated build and test, which has been implemented to support continuous integration. Build, test and analysis of varying system scope and coverage run both on event basis and on fixed schedules, depending on needs and circumstances. A wide range of physical target systems as well as a multitude of both in-house and commercial simulators are used to execute these tests.
**4.2.2 Study Case B**

Study Case B is developing airborne systems and their support systems. The main product is the Gripen fighter aircraft, which has been developed in several variants. Gripen was taken into operational service in 1996. An updated version of the aircraft (Gripen C/D) is currently operated by the air forces in Czech Republic, Hungary, South Africa, Sweden and Thailand. The next major upgrade (Gripen E/F) which will include both major changes in hardware systems (sensors, fuel system, landing gear etc) and a completely new software architecture.

Continuous integration practices such as automated testing, private builds and integration build servers are applied in development of software for the Gripen computer systems. The software teams commit to a common mainline. Testing is conducted in simulated environments, rigs and test aircraft.

**4.3 Problems and Experiences**

In this section we will compare the conditions at Study Case A and Study Case B regarding seven topics (derived from the characteristics of the companies' products). The seven topics are shown in Table 14. In general, our experiences of applying continuous integration practices are positive, but we present challenges related to applications with complex software systems together with mechanical and electronic systems.

<table>
<thead>
<tr>
<th>Id</th>
<th>Topic Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Complex user scenarios need manual testing</td>
</tr>
<tr>
<td>T2</td>
<td>Compliance to standards shifts focus away from working software</td>
</tr>
<tr>
<td>T3</td>
<td>Longer build time due to tightly coupled systems</td>
</tr>
<tr>
<td>T4</td>
<td>Complete system a secondary concern due to many technology fields</td>
</tr>
<tr>
<td>T5</td>
<td>Restricted access to information due to security aspects</td>
</tr>
<tr>
<td>T6</td>
<td>End-to-end testing impossible without architectural runway</td>
</tr>
<tr>
<td>T7</td>
<td>Test environments often a limited resource with bespoke hardware</td>
</tr>
</tbody>
</table>

Table 14: The seven topics discussed in Section 4.3.

**4.3.1 T1: Complex User Scenarios Need Manual Testing**

Study Case A is developing communications solutions where systems interact with other systems. The user experience is limited to measurable capabilities such as quality and data transfer speed. Every other aspect of the user experience is linked to the user interface of products that are provided by other companies.

Study Case B on the other hand develops a product where the pilot cockpit is a vital part of the product. The pilot's judgment is critical with regards to whether the presentation and manoeuvring of sensors, weapons and other systems on the displays can support the pilot to fulfill the assigned missions.
Our experience is that usability testing for a product such as the Gripen fighter (Study Case B) is very difficult to discuss in terms of automated testing. Testing with the purpose of checking if for example a symbol is presented after a button is pressed can be automated, but the pilot's judgment when evaluating a complex user scenario is extremely difficult to replace with an automated test case. Our experience is that the challenges of testing which include subjective experiences are clearly valid for Study Case B, but are much less pronounced (if present at all) at Study Case A.

4.3.2 T2: Compliance to Standards Shifts Focus Away From Working Software

Development of airborne systems follows standards like DO-178B or specifically in Sweden RML-V-5. Development is to a great extent requirement-driven, where high-level requirements are broken down into low-level system requirements. Specific roles are responsible for quality assurance through reviews and audits. The telecom industry also has rules and regulations, but often not to the same extent as avionics software systems.

If evidence that the product is compliant to a standard is at the same importance as the product itself, however, a document review can be seen as time-critical and be given higher priority than software problems. Our experience is that Study Case B (fighter aircraft) to a greater extent than Study Case A (telecom systems) has milestones and project progress connected to audits (on system design or software) or formal documents (a document is issued that is required at a certain stage in the process).

4.3.3 T3: Longer Build Time Due to Tightly Coupled Systems

The Gripen aircraft (Study Case B) is a highly integrated system which uses rate-monotonic scheduling with a cyclic execution pattern. Both execution within a computer and communication between the central computers are scheduled. Our experience is that when working with a highly integrated (tightly coupled) system, a small delivery to the main track may cause building and linking of a large part of the computer system which implies long build times.

Study Case A’s telecom systems have varying degrees of real time characteristics, typically depending on the level of abstraction with regards to the underlying physical interfaces. Similarly, the degree of coupling and ability to modularize also varies. Study Case A has had (where possible) very positive experiences of increasing "integration time" modularity – in other words, building and testing the systems in smaller, independent pieces. This approach is impeded by the tighter coupling of Study Case B.

4.3.4 T4: Complete System Secondary Concern Due to Many Technology Fields

Development of a product requires knowledge of all technology fields that the product covers. The Gripen aircraft (Study Case B) covers technology fields spanning from for example aerodynamics, engine control and electrical power system to communication system, navigation and mission planning. The telecom products of Study Case A also cover many technology fields, such as network optimization or handling of customer data.

Our experience is that a large number of technology fields fosters silo behaviors. The organization tends to establish tailored ways of working for each system (technology field) and also tends to see it as "our system", and treating the complete system as a secondary concern. This is
arguably as a consequence of limited understanding of the unique challenges and requirements governing the many parts of the complete system. Silo mentality in this scenario, but we find it severely exacerbated when these silos operate in separate engineering disciplines with little or no understanding of one another's unique characteristics or challenges.

4.3.5 T5: Restricted Access to Information Due to Security Aspects

All companies have to take into account how to protect company confidential information. Almost every company has a strategy for how to avoid information leakage. Another aspect is the ability to protect customer data. That is, to ensure that information about one customer's performance or available functionality is not exposed to other customers. Both Study Case A and Study Case B must make allowances for this.

A third aspect is defense-related security. Defense-related security includes safeguarding of national security and foreign policy objectives for all (military) customers, but also to follow export control regulations for parts or sub-systems supplied by a foreign vendor. US arms regulations demand that it is secured that only specified individuals have access to software included in defense-related items, which increases the difficulty of a common understanding of the product. Export control of US technology (especially arms regulations) is regulated by The International Traffic in Arms Regulations. Our experience is that these regulations are affecting Study Case B (fighter aircraft), but are not relevant for Study Case A (telecom systems).

4.3.6 T6: End-to-end Testing Impossible without Architectural Runway

Platforms like .NET or Java Virtual Machine enable a developer to rapidly produce software that includes both user input/output and communication with other software modules. Embedded systems developed by Study Case A (telecom systems) and Study Case B (fighter aircraft) are not built on a commercially available platform like .NET. Instead, the development of an entirely new product includes a long period of in-house construction of a platform with all infrastructure functions. When you start from a clean slate you give up the luxury of a platform with working infrastructure including for example communication between systems, functional monitoring or data registration.

Dean Leffingwell defines the term architectural runway as infrastructure sufficient to allow incorporation of new requirements (new functionality) [Leffingwell 2010]. Development for bespoke hardware with tight dependencies to the physical interfaces miss out the benefits from a commercially available platform. Consequently, the architectural runway is much longer.

Our experience from both study cases is that at the initial phase of development of a new product (lasting for a significant part of the project) the sub-systems cannot be integrated. Due to this, the product cannot for a long time be functionally tested end-to-end to expose any problems.

4.3.7 T7: Test Environments Often a Limited Resource with Bespoke Hardware

Development of embedded systems is highly dependent on bespoke hardware, both mechanical and electronic parts. The telecommunication equipment delivered by Study Case A (for example network nodes) often contain specialized internally developed hardware, and is deployable in a large number of variants. The equipment may also coexist with a wide variety of topologies, including equipment developed by Study Case A and/or any competing vendor. The computer
system in the Gripen aircraft (Study Case B) is built on internally developed hardware and equipment developed for aeronautical applications. Gripen is designed in different variants, and each variant has sub-variants. Simulators with models of hardware are used by both Study Case A and B, but have limitations regarding for example timing.

When the system is based on bespoke hardware (not running on any standard computer) and hardware is considered expensive or in short supply, the test environments often become a limited resource. Further on, a large number of hardware configurations (caused by customer-specific hardware) increases the test effort needed for every build. Our experience is that both Study Case A and Study Case B are highly dependent on bespoke hardware, with Study Case A having to handle a greater degree of differences in hardware configurations.

### 4.4 Analysis

In the previous section we compared the conditions at Study Case A and Study Case B regarding seven topics (T1-T7 in Table 14) related to product characteristics, based on our experiences. In this section we will analyze how this relates to the seven-bullet summary of continuous integration (C1-C7 in Table 13).

#### 4.4.1 C1: All Developers Run Private Builds

The first cornerstone (C1) states that "All developers run private builds on their own workstations before committing their code to the version control repository to ensure that their changes don't break the integration build".

Test environments easily become a limited resource if the system is based on bespoke hardware (T7). We argue that if the developers build and test in a simulated environment, they cannot fully ensure that the exact same test cases will not expose problems during test activities that run on real hardware.

#### 4.4.2 C2/C3: Commit Code and Build Often

As we find the two corner-stones "Developers commit their code to a version control repository at least once a day" (C2) and "Integration builds occur several times a day on a separate build machine" (C3) related they will be jointly discussed.

Build time is correlated with the size of the code base. If a product can be divided into several parts that are built and linked in parallel as separate binaries, build time can be reduced. If the product is a tightly coupled system, such sectioning is more difficult or even impossible which implicates a longer build time. We argue that a long build-and test-time (T3) reduces the developer's interest in committing to the main track often, and the developers will not commit their code to the repository at least once a day. Kent Beck quite simply states that "if integration took a couple of hours, it would not be possible to work in this style" [Beck 2000]. If build- and test-time for the integration build (T3) extends to several hours, this severely limits the number of integration builds that can be produced in a day.
4.4.3 C4: 100% of Tests Must Pass for Every Build

To use automated tests to support the practices of continuous integration is a far more effective approach than manual testing [Duvall 2007]. We find automated tests to be a prerequisite for the continuous integration of any non-trivial software system.

Testing should include different categories of tests, from unit tests and component tests to functional tests and tests of load/performance and other capabilities. Tests of Human Machine Interaction (HMI) differ from other types of testing, as the purpose of the tests are to check that the usability is considered at least good enough by user representatives. Manual usability tests take longer time to execute and are less predictable than automated tests, which means they cannot be repeated for every integration build (at a build rate of several builds a day or more).

When the system is based on bespoke hardware (not running on any standard computer) and hardware is considered expensive or in short supply, the test environments soon become a limited resource. Further on, a large number of hardware configurations (caused by customer-specific hardware) increases the test effort needed for every build. With a wide range of hardware configurations it is no longer clear what "100% of tests must pass" actually means – does it mean testing on all valid configurations or a representative subset?

Test environments more easily become a limited resource with bespoke hardware, especially if the product uses many hardware configurations which increases the test effort (and consequently the demand of test environments). A large number of hardware configurations also increases the risk for flaky tests, as there are more test environments to maintain.

We argue that both a product with complex user scenario testing (T1) and many bespoke hardware configurations (T7) can be impediments when trying to adhere to the rule that "100% of tests must pass for every build".

4.4.4 C5: A Product is Generated that Can Be Functionally Tested

Before a first version of all infrastructure for the complete product has been developed, the developers don't have a minimum viable product which then can be incrementally expanded upon. That is, before the architectural runway is established, the product cannot be generated (assembled) and cannot be functionality tested end-to-end (T6).

Another aspect is that is important that all participants have common understanding of the desired functionality of the product. We argue that if the product has a large number of technology fields (T4) and especially if the technology fields are not adjacent, it becomes more difficult to agree on the content and meaning of functionality tests. Security aspects (T5) can also be an impediment, such as when developers are hindered from communicating freely regarding the exact content of the functions they have built. This further increases the difficulty of a common understanding of the product, which also becomes an impediment related to testing the product end-to-end.

4.4.5 C6: Fixing Broken Builds Is of the Highest Priority

Fixing broken builds fast restores the confidence for a stable and sound main track. If status of the software is undisputed as the full picture of status in the project, it is easy to keep focus on fixing broken builds fast.
"Working software over comprehensive documentation" is one of the values in the agile manifesto, which also fully applies to continuous integration. This might be seen as a value that collides with the principles of development of safety-critical, highly regulated software such as medical devices, nuclear power stations or flight-critical software. This conflict is also discussed by Janet Gregory and Lisa Crispin [Gregory 2015].

Regulated environments typically apply one or several standards that require that the developing organization should "show evidence" of compliance to the standard, which should be done in written documents. We argue that the obligation to show compliance to a standard (T2) can be an impediment in relation to the intention of fixing broken builds as the highest priority.

4.4.6 C7: Developers Review Reports to Seek Areas for Improvement

The last corner stone states that "Some developers review reports generated by the build, such as coding standards and dependency analysis reports, to seek areas for improvement". We argue in the same way as for corner-stone C5 (Section 4.4.4) that a large number of technology fields (T4) and security aspects (T5) make it more difficult to achieve a common understanding of the product. Only a few people have an overview of the whole product, and in many cases information cannot be shared due to security restrictions. To some extent, this affects how developers review reports on other parts of the product than where they are working themselves.

4.5 Conclusion

The analysis in the previous section relates the seven topics to the corner stones for continuous integration that were presented in the introduction. The analysis is summarized into the following bullets:

- If the developers run tests in a simulated environment, they cannot fully ensure that the same tests will pass for the integration build that runs on real hardware
- Tightly coupled systems (causing long build- and test-time) implies additional challenges related to frequent deliveries and integration builds several times a day
- A product with complex user scenarios and/or bespoke hardware (especially a large number of hardware configurations) implies that the rule "all tests must pass for every build" must be replaced with other testing approaches
- In a highly regulated environment, "fixing broken builds" must be balanced against other project objectives
- At the initial phase of development of a new product (before the architectural runway is established) the sub-systems cannot be assembled in order to test the system functionally end-to-end and expose any integration problems
- It is more difficult to achieve a common understanding of a product with a large number of technology fields or security aspects, which affects tests and reviews

The relations that were found are summarized in Figure 8.
We believe that these experiences represent an area of further work of high relevance to large segments of the software industry. Any research promising to mitigate the discussed impediments would be of great value in the embedded software development community.

Figure 8: Relations between corner stones and impediments.