Continuous integration and delivery applied to large-scale software-intensive embedded systems
Martensson, Torvald

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date: 2019

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
Chapter 6

Continuous Integration Behaviors in Large-Scale Industry Projects

This chapter is based on: Mårtensson, T., Ståhl, D. and Bosch, J. (2017). Continuous integration impediments in large-scale industry projects. IEEE International Conference on Software Architecture, ICSA 2017, pp. 169-178. The material in Sections III-E and III-F of the ICSA conference paper is also presented in the journal paper included in Chapter 8, and is therefore not included here in this chapter.

Abstract: Based on interviews with 20 developers from two case study companies that develop large-scale software-intensive embedded systems, this paper describes the continuous integration behaviors in projects where up to 1,000 developers commit to the same mainline. We find that the developers who commit to a branch look at the branch as "their mainline", where they implement and test their function step by step. Continuous integration is not necessarily black or white, but there is a grey zone where one must consider what the system which one integrates is, and that arguably there are levels of continuous integration. One might be practicing continuous integration at one level, but not at another, and this may have different pros and cons depending on the context.

6.1 Introduction

Continuous integration is often described as an efficient way of conducting software development. The practice and its benefits are described in both books and other research papers (Duvall 2007, Humble and Farley 2010, Rodriguez et al. 2016, Ståhl and Bosch 2013). Still, many projects struggle to realize their implementations of continuous integration (Roberts 2004, Owen Rogers 2004, Ståhl and Bosch 2014a, Ståhl et al. 2016b). In previous work (Ståhl et al. 2017b), we have analyzed the continuity of different continuous integration implementations in organizations with hundreds or thousands of developers, and investigated the correlation between the continuity of continuous integration and size of the organization. Related work (Humble 2014) also describe that large-scale continuous integration is challenging.

But the question still remains why the implementations of continuous integration in industry projects differ so much from the practice as it is described in literature, for example in Martin Fowlers popular paper (Fowler 2006). We have, as one small step to answer this question, in previous work reported our own experiences of factors that could constrain a full implementation of continuous integration (Mårtensson et al. 2016). We believe that filling the gap between large-scale industry projects and continuous integration as described in literature is a very topical issue, especially as continuous integration is now implemented by many large-scale organizations (often together with continuous delivery).
The topic of this paper is to answer the following research question: What are the continuous integration behaviors of developers in large-scale industry projects, and how do the developers look at pros and cons regarding committing to branch or directly to the mainline? We will focus on continuous integration implementations for software-intensive embedded systems (software systems combined with electronic and mechanical systems).

The contribution of this paper is two-fold. First, it provides interview results from two large-scale industry projects regarding the developers’ continuous integration behaviors. Second, it sheds light on how developers describe pros and cons regarding committing to branch or directly to mainline (also called trunk). The remainder of this paper is organized as follows. In the next section we present the research method, including a description of the case study companies. This is followed by an analysis of the interview results in Section 6.3. Threats to validity are discussed in Section 6.4. The paper is then concluded in Section 6.5.

6.2 Research Method

In order to answer the research question stated in Section 6.1, a series of interviews were conducted with interviewees from two companies. The interviews were conducted according to the selected interview design, and were followed by an analysis of the interview results.

6.2.1 Case Study Companies

The case study companies are developing large-scale and complex software for products which also include a significant amount of mechanical and electronic parts. Both companies are multi-national organizations with more than 15,000 employees. In this paper, the companies are referred to as Company A and Company B. Continuous integration practices such as automated testing, private builds and integration build servers are implemented in various ways in different parts of the companies. The software teams commit to a common mainline for each software product. 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.

The interviewees from the case study companies were purposively sampled in line with the guidelines for qualitative data appropriateness that is given by Robson and McCartan (2016) (pp.166-168): “interviewing ‘good informants’ who have experienced the phenomenon and know the necessary information”. The individuals were sampled for scope and variation to represent a wide range of sub-systems (with different characteristics) and project types (size and development context).
6.2.2 Interview Design

Twenty individual interviews were held with participants from two case study companies (ten at each company). The interviews were conducted as semi-structured interviews, held face-to-face or by phone using an interview guide with pre-defined specific questions. The interviewer was transcribing the interviewee’s responses during the interview, and each response was read back to the interviewee to ensure accuracy. The interview questions were sent to the interviewee at least one day in advance to give the interviewee time to reflect before the interview.

The first part of the interview queried the software developer for contextual information, and how the interviewee currently commit their software. This information was gathered as the interviewee’s answers may depend on their background and current work situations. In the second part of the interview the interviewee was asked questions about how the interviewee would prefer to commit software, and about the interviewee’s opinions about continuous integration practices.

The word “commit” is used in the research study as the term for integration of a software developer’s code into the mainline. As there is no unified terminology, the use of “commit” (or any other word) could be seen as problematic. The word “integrate” could be seen as an alternative, but might create confusion as it could also be interpreted as rebase. The alternative “deliver” has connotations of formality and process, which is not a limitation we like to include. A third alternative is “push”, but is really a term connected to a specific family of tools (distributed version control systems). That being said, we consider “commit” to be the best alternative.

6.2.3 Analysis and Validity

The data on how often the interviewed developers currently commit software and would wish to commit software (collected from the interviews) were collated into a table. Then min, average and max values et cetera were simply calculated from the information in the tables. The interviewees were quite generous with comments regarding both their opinions about benefits of continuous integration practices and about committing to branch versus directly to mainline. The comments were collated and processed with exploratory analysis. Representative quotes were selected from the transcripts to convey the interviewees’ opinions and discussions.

Construct validity, internal validity and external validity have been considered during research design, during the interviews and as part of the analysis (as described in Section 6.4). The 12 threats to internal validity presented by Cook, Campbell and Day (1979) are used to discuss internal validity.
6.3 Analysis of Interview Results

6.3.1 Background Information

The interviewed developers were generally very experienced, with an average of 12.15 years of experience in industry software development. Company A interviewees had experience spanning from 4 to 20 years, with an average of 11.6 years. Company B interviewees experience spanned from 4 to 32 years, with an average of 12.7 years.

The developers from Company A were all working in the same project and committed software to the same mainline, although all interviewees were working in different teams and with different subsystems. The project included around 500 software developers, of which about 150 worked with the product code on a daily basis. The scope of the Company A project was to develop a new product (with a new platform) during a time-span of several years.

The interviewees from Company B worked in several different projects, with the number of developers (committing to the same mainline) spanning from 50 to 1,000. The average number of developers for the projects in the Company B interviews was 626 developers. The scope of the Company B projects was in six of ten cases described by the interviewees as adding new features or systems to an existing product (with an established architecture). Three interviewees described the project as development of a new product with a new platform. One interviewee described the project as both.

6.3.2 Rating Continuous Integration

The developers generally had a very positive attitude towards continuous integration. The ratings (1-5) of how software development of large-scale systems benefits from the practices of continuous integration were generally 4 or 5, with an average of 4.13. One of the interviewees refrained from answering the question. Many of the developers added comments, which in most cases were very positive, for example “I see only benefits from it” or “I don’t really see what the alternative would be”. Some commented on the benefits of shorter feedback cycles, for example “Getting rid of uncertainty, how much merge work do we have ahead of us?” Several interviewees also commented on the difficulties to make continuous integration work ("extremely important, but it’s difficult to make it work"). The most commented area was difficulties to establish test suites for regression tests and test environments, for example “Potentially it could be great. […] Our test environments are damn slow.” Other voices, however, looked at it from a different angle, stating for example “Particularly the testing aspect I think I have experienced the greatest improvement from. You can always test on the latest.”

Two of the interviewees rated the benefit from continuous integration practices as 3, and one interviewee as 2. They all provided comments on the relation between continuous integration and different aspects of system design, e.g. “You don’t want to deliver so quickly that you don’t have time to think. It’s often the interfaces that aren’t settled. I like to say that I think we kid ourselves. We think we move faster because we deliver frequently, but often it’s an illusion, because what we do deliver isn’t thought through.”
6.3.3 Current Frequency of Commits

The interviewees’ responses on frequency of commits were interpreted quite conservatively, e.g. “several times a day” was interpreted as three times a day. The values are also calculated with regards to that a week has seven days (not a five day work week). For example, “every six weeks” is interpreted as 42 days. The interview results on how the interviewees commit to branch and mainline, respectively, were:

- Committing to branch: 16 developers, of which 3 are mixed committers
- Committing directly to mainline: 7 developers, of which 3 are mixed committers

Interviewees who were committing to branch commit to a team or a feature branch with a frequency spanning from several times a day to every second day. In this group, two developers were divergent compared to other branch committers, and seem to not yet have established a way of working that could be related to as continuous integration. One of the outliers committed to a branch once every two weeks, a figure quite different from other interviewees. The interviewee commented: “We recently reorganized. Before we didn’t do continuous integration […].” The other outlier interviewee committed to the branch similar to other interviewees, but the branch was committed to mainline the first time after “four months” but was rejected and the next time “after a year”.

Interviewees who were committing directly to mainline commit software with a frequency spanning from “several times a day” to “once a week”. Three of the interviewees are mixed committers, meaning that they responded that they commit to both branch and directly to mainline. The frequency of their commits to both branch and mainline are similar to the other interviewees’ responses, with the exception of one interviewee who committed the branch to mainline less frequent than other interviewees (70 days between commits). Several of the other interviewees commented that their response referred to development of new functions, adding that corrections is delivered “instantly”. That is, during development phases that includes a lot of problem report and corrections, a larger part of the developers are probably also “mixed committers”.

The commit frequency for all interviewees are shown in Figure 21. The branch committers are shown as number 1-16, and mainline committers as number 14-20 (with number 14-16 as the mixed committers). The outliers are number 12 (committing much less frequently to branch) and number 13 (committing the branch much less frequently to the mainline).

A summary of the interviewees’ responses is presented in Table 13 (with an exclusion of the two outliers that seemed not to have established a way of working that could be related to as continuous integration). The interviewees’ (except the two outliers) commits to a branch with an average time of 0.84 days between commits, with a span from 0.33 days to 2 days. The branch was committed to the mainline on average time of 22.41 days between commits. The time between commits from branch to mainline differs more between the interviewees, spanning from 2.80 days to as much as 70 days. Mainline committers deliver directly to the mainline, with the most
frequently committing interviewee on only 0.33 days between commits and the least frequently on 7 days between commits (on average 2.52 days).

Figure 21: Commit frequency for each interviewee to branch and mainline.

Another observation is that 15 out of 20 of the interviewed developers commit to a branch or directly to the mainline daily, or even more frequently than daily. This can be interpreted as that as many as 75% of the interviewees integrate their software at a frequency that follows the guidelines in Martin Fowlers article (Fowler 2006) (shown in Figure 22). If the two outliers are removed, the number is as high as 83% (15 out of 18 developers). The question is then why so many of the developers commit to a branch and not directly to the mainline.

<table>
<thead>
<tr>
<th>Interviewees</th>
<th>Time between commits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>Commits to branch (except outliers)</td>
<td>14</td>
</tr>
<tr>
<td>Commits from branch to mainline</td>
<td>14</td>
</tr>
<tr>
<td>Commits directly to mainline</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 13: Time between commits to branch and mainline.

6.3.4 Commit to a Branch or Directly to the Mainline?

There is no correlation between current ways of working (commit to branch or main) and how the interviewees responded that they want to work. Instead there were very mixed responses, where some of the developers wanted to change from committing to branch to committing directly to the mainline (or vice versa) and some wanted to remain in their current way of working. The results are summarized as:
• Prefer to commit to branch: 9 developers
• Prefer to commit directly to mainline: 11 developers

It shall be noted that eight of the nine developers that prefer to commit to branch come from the same case study company. The commit frequency that each of the interviewees would have preferred is shown in Figure 23 (note that the interviewees are not listed in the same order in Figure 23 as in Figure 21).

Figure 22: Fifteen of twenty interviewees responded that they commit to a branch or directly to the mainline “daily”, or even more frequent (including the two outliers).

The major difference between the data shown in Figure 23 (preferred way of working) and Figure 21 (the present situation) is that only nine developers prefer to deliver to a branch, compared to 16 developers which currently deliver to branch (of which three are mixed committers). There are also no longer any outliers similar to what we found in the data that is related to the present situation. A summary of the interviewees’ responses is presented in Table 14. The interviewees who want to commit to a branch responded that they want to commit with on average 1.24 days between commits, with a span from 0.33 days to 3.5 days. They want the branch to be committed to the mainline on with on average 21.78 days between commits, spanning from 7 days to 63 days. The interviewees who want to commit directly to the mainline want to commit on average every 2.08 days, spanning from 0.06 days (every 30 minutes) to 7 days. The values (min, average and max) are very similar to the corresponding values in Table 13, sometimes a bit higher and sometimes a bit lower. This is rather surprising as only two of the twenty interviews responded with the same values on the questions how they work now and how they want to work.

The responses on how often other developers’ software should be committed to the mainline were generally the same values as how often the interviewees want their own software to be committed. There is one exception though, where a platform developer responded that updates of the platform should be committed every 4-8 week, but a “functional branch should commit daily”. 
The interviewees were quite generous with comments on pros and cons regarding committing to branch or directly to mainline. The comments can be sorted into four groups, each group supported by statements from 6-9 of the interviewees:

- In a branch you can test your functionality before committing to mainline
- Sometimes it’s complicated to break down your work into small pieces that can be integrated directly to the mainline
- If you commit directly to the mainline, you won’t have so much merge problems as if you work on a branch
- Working on a branch is more stable than working directly to the mainline
- If you commit directly to the mainline you get feedback faster

Developers from both case study companies described that they had test activities running on the branch, which were run either regularly or before committing from the branch to the mainline. Testing before commit builds confidence in that the software will not brake anything when it’s on the mainline. As it seems, in both case study companies the test activities that are run as part of the integration process are not considered to be sufficient by the developers. The developers then chooses to run tests on the branch.

<table>
<thead>
<tr>
<th>Interviewees</th>
<th>Time between commits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>Commits to branch</td>
<td>9</td>
</tr>
<tr>
<td>Commits from branch to mainline</td>
<td>9</td>
</tr>
<tr>
<td>Commits directly to mainline</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 14: How often do the interviewees want to commit to a branch or to the mainline?
Another problem described by the interviewees from both case study companies is that it is sometimes hard to break down the work into small pieces that can be committed separately to the mainline, e.g. as one developer stated: “The important thing is that you are done with your work. You cannot always chop up things in small pieces and deliver half a function.” A difference between the two case study companies is that in one of the companies the developers talk about committing complete functions, in the other case study company the developers separate the terms working software and complete software.

Problems with merge (or rebase) when working with a branch were brought up mainly by developers from one of the companies, e.g. “All the merge conflicts you accumulate. You build up a technical debt. […] But if you make small deliveries to the mainline you know what you have and what to expect.” However, another developer from the same company commented that merge and rebase problems also exists when working directly on the mainline: “They batched commits and they failed, and you ended up rebasing for multiple days, so it took a long time to get even small commits in.” We can only speculate around the factors behind that this problem was brought up by the developers from only one of the companies. One explanation could be that in an organization working with functional teams, many developers make changes in the same software components (causing merge conflicts), to be compared with an organization that works with component-based teams. Another explanation to that this problem was not described by the developers from one of the companies is that other issues is more in focus right now.

The opinion that a branch is more stable than the mainline was brought up by developers from mainly one of the companies. One aspect was about changing interfaces: “[I want] stable functions around me when I develop my software. I don’t want other functions to change which makes me update my interfaces often”. The other aspect was to prevent that new commits break existing functions on the mainline, e.g. “People must deliver working functionality that does not destroy other people’s work”. A developer from the other case study company stated in a similar way: “There have been situations where all others have been banned from integrating, because we had to keep a stable mainline when developing our [critical] feature”. The opinion that a branch is more stable than mainline (supported mainly by the developers from one of the case study companies) is mirrored by the opinion that an advantage from committing directly to mainline is faster feedback, which was commented by mainly developers from the other company. As said before, we can only speculate on the factors behind this differences between the companies. One explanation is that one of the companies has invested more in test activities that both provides faster feedback to the developers and keeps the mainline more stable.

Finally, there were also some interesting comments by single interviewees. One interviewee (platform developer) commented on that major changes in the platform should be done in major steps, in order to provide stability for other teams. This corresponds well with the comments on that a branch is more stable than the mainline (above). One interviewee commented on that it’s better to work in a branch when you are working with prototyping: “If you work with prototyping where the system changes a lot [on the mainline], you must update your automated tests a lot.”
In a strict continuous integration context as described in literature (Duvall 2007, Fowler 2006), developers should commit to the mainline at least once a day. Instead of dismissing the developers that commit to a branch as not practicing continuous integration, we argue that the definition of continuous integration must be seen in a wider perspective. Our previous work (Ståhl et al. 2017b) as well as related work (Humble 2014) shows that even though a large-scale organization is integrating and delivering in a very rapid pace, this does not mean that every developer is “continuously integrating” as described in the practice. As we see it, the developers who commit to a branch look at the branch as “their mainline”, where they implement and test their function step by step. As eight of the nine developers that prefer to commit to branch come from the same case study company, it is plausible that the preference to commit to branch or mainline for the most part is related to company culture and tradition. However, we believe that product architecture can also be a plausible explanation. The developers in projects within Company B that commit directly to the mainline are often working on a product that is integrated as a binary with other products in a wider context. The overall system (including thousands of developers) is integrated at a frequency far from “every day”. On the other hand, Company A develops a large product that consists of several subsystems with different purposes and characteristics, but share the same mainline due to architectural limitations. We argue that continuous integration could not be simplified to discussing how often the developers commit to the same mainline, but also include integration of a subsystem on a branch or integration of binaries built from several different mainlines.

6.4 Threats to Validity

This section discusses threats to construct validity, internal validity and external validity.

6.4.1 Threats to Construct Validity

One must always consider that a different set of questions and a different context for the interviews can lead to a different focus in the interviewees’ responses. In order to handle threats against construct validity, the interview guide was designed with a terminology that we see as most commonly used (see Section 6.2.2). Open questions were purposely designed to fit both developers who commit to a branch and those who commit directly to the mainline. We also describe the interview guide used in the study and the background of the interviewees and the case study companies.

6.4.2 Threats to Internal Validity

Of the 12 threats to internal validity listed by Cook, Campbell and Day (1979), we consider Selection, Ambiguity about causal direction and Compensatory rivalry relevant to this work:
Selection: The interviewees from both case study companies were purposively sampled in line with the guidelines for qualitative data appropriateness given by Robson and McCartan (2016) (see Section 6.2.1). Due to this, the interviewees represent a wide range of sub-systems (with different characteristics) and project types (size and development context). Considering the rationale of these samplings and the fact that Robson considers this type of sampling superior for this type of study in order to secure appropriateness, we consider this threat to be mitigated.

Ambiguity about causal direction: While we in this study discuss correlation, we are very careful about making statements regarding causation. Statements that include cause and effect are collected from the interview results, and not introduced in the interpretation of the data. Due to this, we consider this threat to be mitigated.

Compensatory rivalry: When performing interviews and comparing scores or performance, the threat of compensatory rivalry must always be considered. Therefore, the questions in our interviews (see Section 6.2.2) were deliberately designed to be as value neutral as possible, rather than judging the interviewee’s performance or skills. However, our experiences from previous work is that we found the interviewed engineers more prone to self-criticism than to self-praise.

6.4.3 Threats to External Validity

The study is based on interviews from two case study companies. It is conceivable that the findings from this study are only valid for these companies, or only for companies that develop large-scale and complex software products which also include a significant amount of mechanical and electronic parts (software-intensive embedded systems). However, related work (Humble 2014) also describes that large-scale continuous integration is challenging, which shows that the problems described in this paper is evidently not isolated to our two case study companies.

6.5 Conclusion

In this paper we have presented interview results from 20 developers with an experience of 4-32 years (on average more than 12 years) working in large-scale software development projects. The interviewees come from two case study companies, which both develop large-scale and complex software systems for products which also include a significant amount of mechanical and electronic parts. We describe the developers’ integration behaviors, as well as how they look at pros and cons regarding committing to branch or directly to the mainline.

The interviewed developers currently commit either to a branch or directly to the mainline, or in some cases both (described in Section 6.3.3). The interviewees commit to a branch on a frequency spanning from multiple times a day to every second day. Commits directly to the mainline span from multiple times a day to once a week. Fifteen of 20 developers commit to a branch or directly to the mainline on daily basis, or even more frequently than daily. We find that the developers who commit to a branch look at the branch as “their mainline”, where they implement and test their function step by step.
The interviewed developers’ attitudes to committing to a branch or directly to the mainline are mixed (as presented in Section 6.3.4). Nine of the twenty interviewees prefer to commit to a branch, and the others prefer to commit directly to the mainline. Instead of dismissing the developers that commit to a branch as not practicing continuous integration, we argue that the definition of continuous integration must be seen in a wider perspective. We argue that continuous integration could not be simplified to discussing how the developers commit to the same mainline, but also includes integration of a subsystem on a branch or integration of binaries built from several different mainlines. Continuous integration is not necessarily black or white, but there is a grey zone where one must consider what the system which one integrates is, and that arguably there are levels of continuous integration. Integration within your component or sub-system, integration with the larger system and integration with any system-of-systems layer on top of that – and there may be different integration frequencies at different levels. In other words, one might be practicing continuous integration at one level, but not at another, and this may have different pros and cons depending on the context.

6.5.1 Further Work

In addition to the results presented in the analysis and the conclusions, we believe that this study also opens up several interesting areas of further work.

In Section 6.3.3 we found that during development phases that include a lot of problem reports and corrections, a larger part of the developers are probably also “mixed committers”. An area of further work is to examine how the developers’ continuous integration behaviors change during different development phases.

In Section 6.3.4 we discuss explanations to why developers see their branch as “their mainline”. A large area of further work is to examine how this is related to company culture and tradition, the product architecture or other factors. We also believe that further work should examine how the practice of continuous integration should not only discuss how often the developers commit to the same mainline, but also include integration of a subsystem on a branch or integration of binaries built from several different mainlines.

Acknowledgement

The authors would like to thank all the participating engineers for their insights, patience and willingness to share their experiences and data with us.