The Journal of Systems and Software 127 (2017) 150–167

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

The continuity of continuous integration: Correlations and consequences

Daniel Ståhl[a,*], Torvald Mårtensson[b], Jan Bosch[c]

[a] Ericsson AB, Linköping, Sweden
[b] Saab AB, Linköping, Sweden
[c] Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 9 August 2016
Revised 22 November 2016
Accepted 6 February 2017
Available online 7 February 2017

Keywords:
Continuous integration
Continuity
Size
Scale
Complexity
Agile
Modularity
Manufacturability
Developability

ABSTRACT

The practice of continuous integration has firmly established itself in the mainstream of the software engineering industry, yet many questions surrounding it remain unanswered. Prominent among these is the issue of scalability: continuous integration has been reported to be possible to scale, but with difficulties. Understanding of the underlying mechanisms causing these difficulties is shallow, however: what is it about size that is problematic, which kind of size, and what aspect of continuous integration does it impede? Based on quantitative data from six industry cases encompassing close to 2000 engineers, complemented by interviews with engineers from five companies, this paper investigates the correlation between the continuity of continuous integration and size. It is found that not only is there indeed a correlation between the size and composition of a development organization and its tendency to integrate continuously; there is evidence that the size of the organization influences ways of working, which in turn correlate with the degree of continuity, raising the question of software manufacturability. It is further observed that developer behavior in ostensibly continuously integrating cases does not necessarily match expectations, and that frequent integration of the product itself does not automatically imply that each individual developer commits frequently.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

Today the agile practice of continuous integration is widespread in the industry. Numerous projects — particularly large scale projects — still struggle to fully realize the practice, however. These difficulties have been studied in related work (Rogers, 2004; Roberts, 2004) and by us in previous work (Ståhl and Bosch, 2014a; Ståhl et al., 2016). As the size of a software development effort increases several things tend to happen. The size of the code base increases, implying longer build times. The number and the scope of tests tend to increase, implying longer test times. This in turn leads to a decreased time frame in which to integrate one's changes — assuming one does not want to leave the office before verifying that the change was successful. Furthermore, a great number of developers involved implies both an increased rate of changes and more people impacted by failed integrations.

Taken together, these factors represent a significant challenge to successfully implementing continuous integration. Beck (2000) quite simply states that "if integration took a couple of hours, it would not be possible to work in this style". Despite this, there are numerous examples of very large scale implementations of continuous integration in industry — many of them arguably highly successful — but a question worth investigating is this: are all continuous integrations implementations *equally continuous*?

We posit that the answer is no: in previous work (Ståhl and Bosch, 2014b) we have found differences in a number of factors, including build frequencies, integration frequencies and build durations, from case to case. We further posit that a reasonable assumption is that such differences may be related to the size of the software product being developed and/or the organization developing it. As Rogers (2004) points out, a "natural reaction" to the difficulties above is to "reduce the frequency of commits" but that "committing less frequently means that more changes are included in each integration, which, in turn, increases the likelihood of merge conflicts and, if integration problems do occur, increases

* Corresponding author.
E-mail addresses: daniel.stahl@ericsson.com (D. Ståhl), torvald.martensson@saabgroup.com (T. Mårtensson), jan@janbosch.com (J. Bosch).

http://dx.doi.org/10.1016/j.jss.2017.02.003
0164-1212/© 2017 Elsevier Inc. All rights reserved.

the difficulty of fixing those problems". In conclusion, this "undermines the benefits of continuous integration".

In line with this we often find — both as practitioners and researchers — that members of large projects often do not speak of continuous integration in terms of "developers committing", but rather "teams delivering", hinting at an underlying difference in both practice and mindset (Ståhl and Bosch, 2014a). That being said, those teams may integrate very frequently internally before "delivering" their work to the mainline, and this is certainly in line with part of the practice as it is described by Fowler (2006): "members of a team integrate their work frequently, usually each person integrates at least daily — leading to multiple integrations per day". This is problematic, however, because there may be hundreds of developers external to one's team, who are just as exposed to the consequences of one's changes as one's team members. Conversely, only a subset of one's organizational team may be actively involved in or affected by the integrated changes. In this sense, it is arguably more accurate to think of anyone operating in the same code base as one's "team". Indeed, such team internal integration is clearly incompatible with other parts of Fowler's definition, which goes on to state that "everyone commits to the mainline every day" — not just a team branch!

Consequently, *we hypothesize that the size of the development context directly impacted by any changes made by the developer — demarcated by explicitly versioned dependencies — correlates with the continuity of continuous integration within that context.* To exemplify, at one end of the spectrum we would find microservices with separate source repositories and continuous integration and delivery pipelines (Newman, 2015), partitioning the system into clearly separated parts within which developers are afforded a high degree of autonomy. At the other end, however, we would find large monoliths in which any change made by that same developer results in rebuilding and re-testing the entire product, because it has the potential to directly impact any other part of that product and consequently any other member of the organization developing it. Fig. 1 clarifies the boundaries of the area of direct change impact by providing an example of how modules within a system depend on one another and how direct impact is either contained or leaked by these dependencies.

Following this reasoning, we phrase the following research question: *What is the correlation between size of an area of direct change impact and the continuity of continuous integration in industry practice, and how does it affect developer behavior?*

The contribution of this paper is three-fold. First, it provides both researchers and practitioners an improved understanding of the underlying factors affecting the de facto outcome of continuous integration practice in industry — one we in previous work have found to vary widely (Ståhl and Bosch, 2013) — by investigating and documenting the correlation between multiple metrics of size and integration continuity. Second, it sheds light on how developer behavior is affected by these factors. Third, it discusses architectural decisions which may help practitioners in improving their continuous integration practice and by extension the manufacturability of their software.

This paper is organized as follows. The next section provides an overview of related work with regards to the challenges involved in scaling continuous integration. This is followed by the employed research method in Section 3, and presentation of and reasoning behind the selected metrics in Section 4. The results are then presented in Section 5 and subsequently analyzed in Section 6. The findings are validated in Section 7, whereupon threats to validity are discussed in Section 8. The paper is then concluded in Section 9. The full interviewee responses from the validation interviews are available in Appendix A.

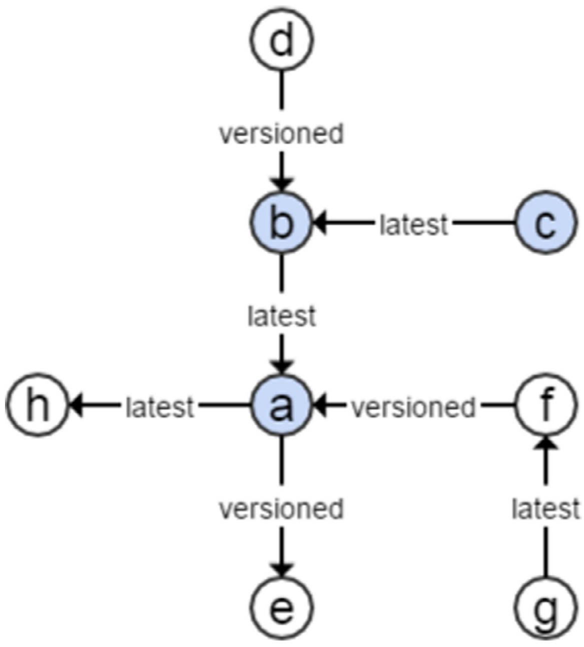

Fig. 1. Module dependencies and direct change impact. Circles represent modules of a system; edges represent dependencies, pointing from the dependent to the dependency. Shaded circles represent modules directly impacted by changes made in module **a** (and consequently with which such changes need to be synchronized). Modules **b** and **c** are thus shaded as they depend on the latest version of **a** — any change to **a** must therefore be synchronized with **b** and **c**. Modules **d, e** and **f**, however, are isolated by their versioned dependencies. From this point of view direct impact does not leak downstream, however: changes in **a** will not affect **h** and consequently must not be synchronized, even though the opposite is true. Similarly, module **f** would need to synchronize any changes with module **g**, but they are both insulated from **a**.

2. Related work

The difficulties of scaling the practice of continuous integration — and by extension, continuous delivery — have been highlighted in literature.

Continuous integration "build" (typically including compilation and certain types of tests) duration varies greatly from case to case, from "a few minutes" (Woskowski, 2012) to more than an hour (Roberts, 2004; Yüksel et al., 2009). As practitioners ourselves and as researchers in previous case studies we have ourselves found similar or even longer durations. If Beck (2000) is to be taken literally (see Section 1), such durations are close to invalidating the practice altogether.

Apart from large projects typically taking longer to build, Rogers (2004) points out that a larger number of developers "increases the rate of code production" and "means that more people are dependent on a working build". In other words, size not only increases the challenges through increased time and resources required to compile, test and analyze the software as well as an increased volume of changes to be compiled, tested and analyzed; size also magnifies the consequences of failure. This dynamic, Rogers finds, drives behaviors such as more infrequent integrations and rigorous pre-commit procedures — a finding confirmed by us in previous work (Ståhl and Bosch, 2014a).

To better understand the phenomenon, one might picture a single lane road with a checkpoint. The guard at the checkpoint is tasked with inspecting every vehicle passing the checkpoint. If the checks can be performed in a matter of minutes and traffic is light — say one or two vehicles an hour — this will work well. As traffic increases, however, the guard will have trouble to keep up and queues will begin to form, and if the checks no longer take minutes but an hour or more, the situation becomes untenable: major congestion with dozens or hundreds of vehicles stuck in traffic ensues. This way, size drives multiple factors which exacerbate one another.

There are multiple strategies for addressing this challenge, such as writing faster tests (Rogers, 2004) or reducing compilation times. Perhaps the most frequently proposed approach is modularity, however. To exemplify, Roberts (2004), calls for "enterprise continuous integration" based on modules integrated as binaries. Other sources support this view, stating that "modularizing features, lower coupling, and increased cohesion enable deployment and continuous delivery" (Bellomo et al., 2014) and "development needs to be modularized into smaller units, i.e. the build process needs to be shortened so that tests can be run more frequently" (Olsson et al., 2012). Other contemporary work confirm the importance of conducive architecture in order to enable parallel development (Bellomo et al., 2013), discuss architectural solutions to the challenges of continuous practices (Buchgeher et al., 2016). In addition, as demonstrated by Rodríguez et al. (2016), continuous practices and architectural concerns in general is an active research topic.

At the same time, while there are plenty of blog posts[1][2][3] and talks[4] on continuous integration and delivery at scale, there is a lack of attempts at objective and academic studies of quantitative data of actual behavior under the hood.

To continue the simile of the single lane road and the checkpoint, modularization has the effect of spreading traffic across multiple (but still single-lane) roads, each with its own checkpoint — with each checkpoint being given only a subset of the total responsibility, thereby speeding up the procedure.

This focus on the importance of modularity supports us in our hypothesis that the size of the context in which developers operate is relevant for successful adoption of the practice, and it is an improved understanding of that relationship and its mechanisms that this paper targets.

3. Research method

In order to answer the research question, this study searches for correlations between metrics of direct change impact area size and the continuity of its continuous integration. This work consists of three distinct parts. First, the identification of conducive metrics for size and continuity, respectively. Second, the collection and analysis of primarily quantitative data from six primary industry cases, in order to confirm or reject the hypothesis that continuity correlates with size — and if confirmed, investigate the nature of such a correlation. Third, the validation of any resulting conclusions through interviews with senior engineers in additional cases.

3.1. Metrics identification

To search for correlations, two types of metrics must be identified: size metrics and continuity metrics. Furthermore, for each type multiple metrics are needed, as there is some room for interpretation of either. To exemplify, while we based on experience hypothesize that size impacts continuity, which kind of size is actually relevant? Size of the organization, or size the source code, or perhaps its complexity?

Consequently, with support from literature we searched for relevant metrics fulfilling several criteria: *general* in the sense of being applicable to not only a single programming language or paradigm, *pragmatic* in the sense of being repeatedly and reliably applicable to multiple cases with good tool support and *widely used and recognized*, not least by industry practitioners. This last part is particularly important, as we believe that by using metrics frequently used by practitioners the results become more accessible. The identified metrics and the reasoning behind selecting them is presented in Section 4.

3.2. Data collection and analysis

As far as possible, data was collected via querying of source code management systems (e.g. using *git*[5] *rev-list* and analogous commands to count commits, committers et cetera), performing static code analysis (using Lizard[6]) or continuous integration servers (e.g. counting Jenkins[7] job executions). During collection from source code management systems, any commits made by automated agents (e.g. build scripts) were filtered out.

During the collection process it soon became evident that several of the identified metrics were not feasible to collect as planned (see Section 5.2) and consequently eliminated. The remaining metrics were collated into a table, and Pearson's test for linear correlation (Agresti and Kateri, 2011) was applied to each pair of size and continuity metrics, respectively in search of statistically significant correlations. Conclusions were then drawn from the results of this analysis as well as from exploratory analysis of the data set. These conclusions were then used to inform the design of the interview guide used to validate the findings (see Section 3.3).

3.3. Validation

To validate the findings and to secure generalizability, an interview guide addressing the findings was created. Using this guide, ten separate individual interviews were conducted with ten engineers in five independent companies, including the company of the six primary cases. These individuals were purposively sampled for seniority and experience from multiple projects, as well as variation through multiple roles (e.g. system architects versus project managers) and industries, in line with the guidelines for qualitative data appropriateness given by Robson (2002): "interviewing 'good informants' who have experienced the phenomenon and know the necessary information".

The interviews were conducted face-to-face or by phone, depending on circumstances, by one interviewer transcribing the interviewee responses during the interview. Each response was read back to the interviewee to ensure accuracy. The interview questions were sent to the interviewees at least one day in advance to give them time to reflect before the interview, and each interviewee was encouraged to provide qualitative statements and individual reflections in addition to their direct answers.

The questions were designed to rate the extent to which the interviewees agree with a number of statements related to the original hypothesis (see Section 1) and the subsequent findings. The responses were then collated and analyzed in comparison with the findings, as presented in Section 7.

4. Metrics identification

This section presents the selected metrics used in the study, and the reasoning behind their selection.

[1] https://www.thoughtworks.com/insights/blog/case-continuous-delivery.
[2] http://techblog.netflix.com/2016/03/how-we-build-code-at-netflix.html.
[3] https://testing.googleblog.com/2011/06/testing-at-speed-and-scale-of-google.html.
[4] https://www.youtube.com/watch?v=Nffzkkdq7GM.
[5] https://git-scm.com.
[6] http://terryyin.github.io/lizard.
[7] https://jenkins.io.

4.1. Size metrics

When discussing the size of an area of direct change impact, there are two distinct categories of size measurements to consider. One is size of the software itself, e.g. the source code. The other is the size of the organization developing that software.

4.1.1. Software size

Since the first dedicated book on software metrics (Gilb, 1976) was published forty years ago, a number of approaches to measuring software size along with related concepts, such as complexity, have been proposed. Among these, lines of code (LOC) is a ubiquitous measurement of software size. As pointed out by Fenton and Neil (2000), its use for measuring software effort and productivity actually predates software engineering as a recognized discipline, and is still widely used today — arguably in no small part because it is very straight forward to measure.

Lines of code as a metric is not without its critics, however. It has been claimed to be “a crude measure” used “as a surrogate measure of different notions of software size (including effort, functionality or complexity)” (Fenton and Neil, 2000). Nevertheless, it is applicable to any programming language, with the caveat that a line of code written in one language may be more expressive — and the code consequently denser — than a line of code written in another language. It is also widely used and easy to reliably and automatically measure. Furthermore, according to the mathematical framework for software engineering measurements proposed by Briand et al. (1996) it is a true measurement of software size.

As for the claim that lines of code is actually used as a surrogate for other notions, we do not dispute this; yet we argue that as the purpose of this work is not to investigate causality, but rather to search for correlations, this is of secondary importance.

Apart from lines of code, another frequently used metric is cyclomatic complexity, also known as the McCabe metric (McCabe, 1976). Admittedly, it is not a software size metric, but instead measures the related concept of software *complexity*. Despite this, cyclomatic complexity is of interest for two reasons. First, it makes intuitive sense that software complexity might impact the effort of integrating parallel work and thereby the continuity of that integration. Second, like lines of code, cyclomatic complexity is ubiquitous in the software industry with a wide array of popular code analysis tools supporting it (Lizard, SonarQube[8], ndepend[9], Checkstyle[10] or the Eclipse Metrics plugin[11]).

In addition to these, numerous other metrics have been proposed in literature over the years. To exemplify, Halstead (1977) lists a number of “basic properties and their relations”, namely program length, program volume and program purity, programming effort and language level, with methods for calculating each of these. Further, Briand et al. (1996) discuss and classify a large number of metrics in the context of their software measurements framework. For size alone they list “#Statements, #Modules, #Procedures, Halstead’s Length, #Occurrences of Operators, #Occurrences of Operands, #Unique Operators, #Unique Operands”. That being said, none of these have achieved the popularity and tool support of lines of code and cyclomatic complexity.

While the size metrics discussed above capture the state of the software at some particular point in time, another approach is to document the accumulated effort expended in reaching that point. The reasoning is that software that has been under development for a long time and by a large number of people may be very complicated and difficult to integrate continuously, disproportionately to its apparent size. One popular method of measuring such an effort — particularly in agile projects — is to use Story Points (Cohn, 2005) or similar estimations of effort and/or time. The problem is that such estimations are only internally consistent within a context with agreed upon rules or guidelines for performing that estimation. It is rare to find estimations that are consistent and comparable across projects or organizations, not to mention across companies. In other words, we do not find estimations of expended effort to be a feasible size metric for the purposes of our study.

Consequently, in this work, we have selected the following two software size metrics[12]:

- **Si$_1$. Lines of code:** The number of lines of code in the area of direct change impact.
- **Si$_2$. Cyclomatic complexity:** The total cyclomatic complexity of the source code in the area of direct change impact.

4.1.2. Organization size

There are multiple ways in which one might measure the size of an organization, and similarly several reasons to believe it may be an important factor. In the words of Perry et al. (1994), “even in the most tool-intensive parts of the process [...] the crucial job of tracking down sources of inconsistency and negotiating their resolution is performed by people”. This and the fact that a substantial portion of the time of software engineers is taken up by activities other than actual programming — such as meetings and discussions — speaks to the importance of an effective organization. Furthermore, Brooks (1975) famously asserts that product quality is affected by organizational structure, while the often cited Conway’s Law states that “organizations that design systems are constrained to produce systems which are copies of the communication structures of these organizations” (Conway, 1968).

The original hypothesis driving this study is that the *size* of an area of direct change impact may be correlated to the *continuity* of its continuous integration. There are several straight forward metrics for this size: the total headcount the organization(s) developing the software of that area and the number of actual developers. Following our experiences described in Section 1, that in some cases developers do not so much “commit” to the mainline as teams “deliver” to it, we argue that the actual number of people performing mainline commits is also an interesting data point.

However, following the reasoning above regarding organizational and communication structures, analyzing the complexity of the developing organization is also of interest — analogous to the inclusion of cyclomatic complexity in software size metrics (see Section 4.1.1).

Nagappan et al. (2008) propose a set of eight organizational metrics for the purpose of predicting failure-prone software, some of which are strictly size related and similar to the above (e.g. number of engineers and the percentage contributing to development), while others address what we refer to as continuity, discussed in Section 4.2 (e.g. edit frequency). Several are, on the other hand, related to organization complexity. While all of potential interest, due to scope and time restraints of our study we selected what we consider the most promising: depth of master ownership. In the words of Nagappan et al. (2008), “this metric determines the level of ownership of the binary depending on the number of edits done” and relates to “less diffusion of activities, a single point of approval/control which should improve intellectual control”, which we argue may be related to the ability to make rapid, frequent changes to the software.

Consequently, we have selected the following metrics of organization size:

[8] http://www.sonarqube.org.
[9] http://www.ndepend.com.
[10] http://checkstyle.sourceforge.net.
[11] https://sourceforge.net/projects/metrics.

[12] With the admission that cyclomatic complexity is, strictly speaking, not a size metric, yet conducive to our purposes as discussed in Section 4.1.1.

- **Si_3. Total headcount:** The number of members in the organization developing the software in the area of direct change impact, regardless of role. As opposed to **Si_4**, this includes not only developers, but e.g. line managers, testers, configuration managers, product owners, project managers et cetera.
- **Si_4. Number of developers:** The number of software developers in the organization developing the software in the area of direct change impact.
- **Si_5. Number of mainline committers:** The number of committers to the software mainline of the area of direct change impact.
- **Si_6. Depth of master ownership:** The hierarchical depth of the organization developing the software in the area of direct change impact. Influenced by Nagappan et al. (2008), but adapted to represent the maximum vertical distance between the members of the smallest organizational unit performing at least 75% of the changes. To exemplify, if more than 75% of all changes would be performed by developers all sharing the same manager, the depth would be 1. If instead their closest shared manager would be their "grandparent", the depth would be 2.

4.2. Continuity metrics

Looking at the continuous integration definition provided by Fowler (2006), it says that "usually each person integrates at least daily" and that it allows one to "detect integration errors as quickly as possible". To measure the *continuity* of the practice, then, one might study the number of integrations (or commits) to the mainline per developer in a given time interval as well as the lead time from commit to feedback on its build and test results. The latter is less than trivial to measure, however: we know that the scope of activities in continuous integration varies greatly (Ståhl and Bosch, 2014b); thus long lead times might simply be a consequence of an ambitious test regime. Additionally, as test activities may be separated into stages (Sturdevant, 2007; Sunindyo et al., 2010; Yüksel et al., 2009; Brooks, 2008), i.e. providing incremental feedback, it is not obvious where to place such measurement points. To exemplify, one case might appear to provide much faster feedback (and thus appear to be more continuous) simply by having a smaller scope, while another case providing equivalent feedback in the same amount of time, but in subsequent stages performs additional tests, would appear to be slower.

Another relevant metric is the number of "builds" in an activity. Again, this has been found to vary from case to case (Ståhl and Bosch, 2014b), but going back to Fowler (2006) it is clear that "each integration is verified by an automated build (including test)". However, apart from anecdotal evidence and findings in related work (see Section 2) simple reasoning shows that this may be problematic in large scale: if a thousand developers integrate once a day then, assuming a ten hour working day, this results in a hundred integrations per hour, or one to two integrations per minute. If each one is to be verified individually, that would require a very fast build indeed, not to mention the time required for humans to analyze and act on the results of those builds.

On a related note the size of integrated changes is also of interest, given that a frequently stated purpose of continuous integration is to avoid "big bangs" and instead make more frequent but smaller changes, thus achieving continuous growth of verified functionality.

Finally, our research question not only concerns correlations, but also asks for any effects on developer behavior (see Section 1). If it is the case that continuous integration is less continuous on the mainline in larger areas of direct change impact, is it possible that developers instead integrate frequently outside of the mainline, e.g. on team branches (following our reasoning in Section 1)? Consequently, we include four continuity metrics:

- **Co_1. Number of mainline commits per developer:** The number of commits on the mainline branch during the studied time period, divided by the number of developers in the area of direct change impact.
- **Co_2. Number of non-mainline commits per developer:** The number of commits to branches other than the mainline during the studied time period, divided by the number of developers in the area of direct change impact.
- **Co_3. Average mainline commit size:** The average number of lines of code changed per mainline commit during the studied time period.
- **Co_4. Number of mainline builds:** The number of integration builds per commit on the mainline during the studied time period. In case of multiple subsequent activities forming a pipeline, count the first (or "root") activity.

4.3. Context

In addition to the above metrics, the following factors were documented for each studied case to provide context:

- **Cn_1. Age:** Age of the source code in the area of direct change impact.
- **Cn_2. Product type:** The type of product, e.g. embedded software system or web application.
- **Cn_3. Industry:** The industry context, e.g. telecommunications or defense.
- **Cn_4. Source code management system:** The source code management system used, e.g. Git or Subversion.[13]
- **Cn_5. Continuous integration server:** The continuous integration server used, e.g. Jenkins or Bamboo.[14]

5. Results

This section first presents the studied cases, and then proceeds to present the collected data.

5.1. Primary cases

The primary cases of the study is a set of six modules in a telecommunications system developed by a large software company. The system has been commercially available for multiple years and is deployed around the world in a business-to-business-to-consumer market; as a whole, the overarching development project is very large, involving thousands of engineers, but the system is also highly modularized. New versions of each module are continuously integrated as binaries (similarly to what is described by Roberts (2004)), with all inter-module dependencies handled by separately versioned interfaces and developed at multiple sites and by separate units within the company. Consequently we regard these modules to be separate areas of direct change impact (as defined in Section 1), and particularly suited for this study as they vary greatly in size but are still within the same company and even the same overall development project — implying that other and potentially confounding factors (such as culture or management style) are somewhat similar. Please see Table 1 for an overview of the primary cases.

5.2. Metrics feasibility

In the data collection phase it soon became evident that not all of the identified size and continuity metrics (see Section 4) were

[13] https://subversion.apache.org.

[14] https://www.atlassian.com/software/bamboo.

Table 1
Overview of the primary cases. Mainline commits refers to the number of commits during the two month study period.

Case	Headcount	Developers	Mainline commits	Age
C_1	53	41	283	3
C_2	250	200	1570	3
C_3	1200	920	1883	8
C_4	280	200	1109	8
C_5	64	45	168	3
C_6	5	5	58	3

feasible, from a strictly pragmatic point of view. These are discussed in detail below. All other metrics could be collected without significant difficulty, and are presented in Section 5.3.

5.2.1. *Software size feasibility*

The software size metrics $\mathbf{Si_1}$ and $\mathbf{Si_2}$ (see Section 4.1.1) proved impossible to collect with any satisfactory level of quality. There were several reasons for this:

- **Divergence in languages and styles:** Each of the studied cases uses its own mix of languages (predominantly C/C++, Java and Python). While a complicating factor adding considerable noise to the resulting data, this was anticipated and would, by itself, arguably have been manageable.
- **Divergence in tools and tests handling:** All of the studied cases handle their tools and tests differently — storing them in separate repositories, in more or less stringent directory structures, or closely intertwined with the production code. Consequently, in some cases it was impossible to reliably separate non-production code from production code. To cope with this we measured the software size of each case both with and without non-production code, but for both data sets we ended up with a significant degree of uncertainty and estimations.
- **Model based software development:** Some of the cases employed model based software development techniques to a greater or lesser extent. This poses an additional challenge to comparison of product size: does one measure the size of the models, or of the generated code? Both are problematic. Models are as a rule significantly more expressive than code, while the code they generate is arguably no more relevant for the purposes of this study than the size of the binaries compiled from it.

Based on the difficulties above, the software size metrics $\mathbf{Si_1}$ and $\mathbf{Si_2}$ were eliminated from our study. We still believe that software size may be a relevant factor worthy of further exploration, but that it requires a larger population, more sophisticated metrics and/or a more homogeneous population to be feasible. While regrettable, we argue that this finding is not without value in itself: while *measuring* product size in a single case is easy, performing meaningful comparison based on such measurements is not — at least not for a small set of heterogeneous cases.

5.2.2. *Non-Mainline commits feasibility*

Metric $\mathbf{Co_2}$ (see Section 4.2) was designed to capture developer behavior outside of the software mainline by measuring how often developers commit to team or feature branches, rather than integrating with the “master” branch. In most cases this proved impossible, however: when used, such localized branches existed in private, unmanaged repositories with no central tracking mechanism. Instead we collected qualitative data on de facto and recommended ways of working with regards to non-mainline commits and branches by asking the engineers in each respective case. This qualitative data is analyzed and discussed in comparison with the quantitative data in Section 6.3.

The difficulties encountered with regards to Si_1 and Si_2 (see Section 5.2.1) prompted us to also question the feasibility of measuring the size of commits (Co_3). Analysis of the nature of the commits, however, revealed that even if e.g. the code base included non-production code such as various tools, these were not updated by the typical commit the same way that the production code was: in a sense, they constituted “ballast”. Consequently, this metric is much less problematic.

5.3. *Collected data*

The collected metrics are based on 5071 commits made by 1049 unique committers (in organizations of in total 1852 engineers) over a two month period. This low number — approximately five commits in two months per developer in organizations ostensibly practicing continuous integrations — is interesting in itself. We again reflect on differences in mindset between “developers committing” and “teams delivering” (see Section 1) and further analyze the significance of the collected data in Section 6.

As described in Section 3.2, Pearson’s test of linear correlation was applied to each pair of size and continuity metrics, excluding $\mathbf{Si_1}$, $\mathbf{Si_2}$ and $\mathbf{Co_2}$ (see Section 5.2). The result of the correlations analysis is shown in Table 2, with *p* representing the two-tailed p value and *R* representing Pearson’s R statistic, all calculated using the SOFA Statistics[15] open source statistical package.

We further note that even though we have in this paper chosen to focus on a subset of the intersections between size and continuity metrics presented in Table 2, the same tendency is evident across the board: projects of larger size tend to be less continuous, regardless of which factor of continuous integration (speed, size of “bangs” or build frequency) we consider. Though the p value of some of the individual correlations is higher than others, taken together we find them to paint a coherent picture.

6. Analysis

This section presents the analysis of the collected quantitative and qualitative data from the primary cases. As described in Section 3.2, the primary method of analysis was to verify the hypothesis by testing the correlation between the size of the developing organization and the continuity of continuous integration. Following this direct investigation of the original hypothesis, however, the collected data was also analyzed in a more exploratory fashion in search of unanticipated patterns and tendencies.

The following subsections discuss a number of perspectives gained from these analyses.

6.1. *Size of commits and size of organization*

As shown in Table 2, the data collected from the six primary cases shows a clear correlation between the number of people involved and the average size of commits. This is true for the total headcount, number of developers as well as actual committers of code during the studied two month interval — a fact which is not entirely surprising, as these size metrics themselves are strongly linked. Fig. 2 exemplifies this by plotting $\mathbf{Si_5}$ versus $\mathbf{Co_3}$. Interestingly, this is analogous to recent findings in related work that productivity correlates negatively with organizational size (Scholtes et al., 2016), which in turn builds upon a large and long-established body of work on the economics of scale (Stigler, 1958; Boehm, 1984; Boehm et al., 2000).

The data does not show any correlation at all with regards to $\mathbf{Si_6}$. The depth of master ownership varies between 1 and 6, and is

[15] http://www.sofastatistics.com.

Table 2
Overview of Pearson's test of linear correlation results for size and continuity metrics. Results of particular interest are highlighted in **bold font**.

Metric	Co_1: Mainline commits	Co_4: Mainline builds per commit	Co_4: Mainline builds per commit
Si_3: Total headcount	p: 0.1523 R: −0.662	**p: 0.004796 R: 0.943**	p: 0.03508 R: −0.843
Si_4: Number of developers	p: 0.1588 R: −0.654	**p: 0.005981 R: 0.936**	p: 0.03845 R: −0.835
Si_5: Number of committers	p: 0.1757 R: −0.635	**p: 0.01039 R: 0.916**	**p: 0.01384 R: -0.902**
Si_6: Depth of master ownership	p: 0.1417 R: −0.674	p: 0.9294 R: 0.047	p: 0.8185 R: −0.122

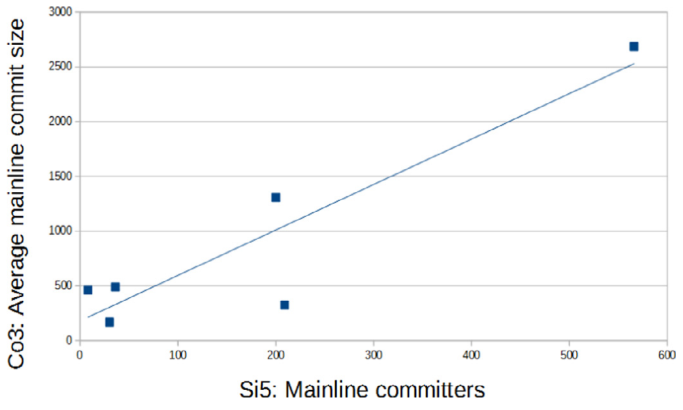

Fig. 2. Plot of number of mainline committers (Si_5) versus average mainline commit size (Co_3).

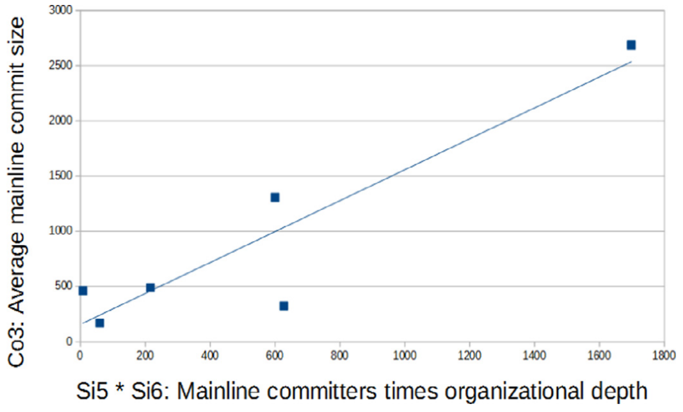

Fig. 3. Plot of number of mainline committers (Si_5) times organizational depth (Si_6) versus average mainline commit size (Co_3).

not strictly related to the headcount of the organization: the deepest case has far from the largest headcount. Our conclusion from this is that to the extent that depth is relevant to the continuity it is not strong enough a factor to bear out by itself in such a small sample size, where it is overshadowed by the headcount metrics.

Rather than considering **Si_6** in isolation, however, one may reason about it in combination with the other size metrics: where headcount represents the *width* of an organization, **Si_6** represents its *depth* or *height*, depending on perspective. We argue that such a combination of the two metrics is not arbitrary, but that it makes intuitive sense: if the developer needs to coordinate not only with a large number of colleagues, but colleagues far removed physically and/or organizationally, that is reasonably an aggravating factor. Following this line of reasoning, when plotting the total organizational *area* of each primary case against its continuity, the result is a somewhat increased correlation. In the example of number of mainline committers versus commit size (see Fig. 2), p decreases from 0.01039 to 0.009672 (see Fig. 3). To summarize, we find that the collected data shows that the headcount (or *width*) of the developing organization affects the average size of commits, and consequently the continuity of continuous integration; on a

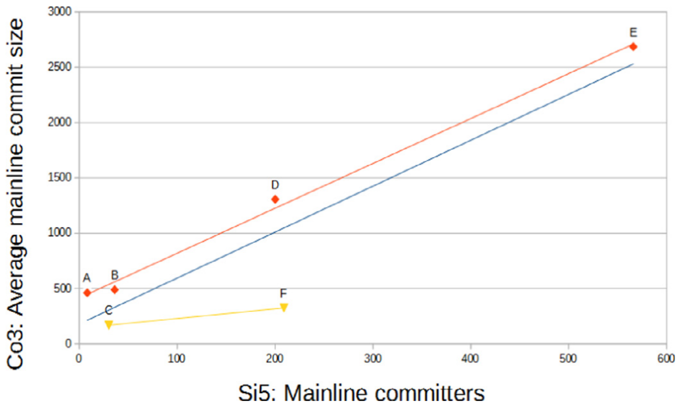

Fig. 4. Split of cases according to way of working with regards to integrating directly with the mainline or via local branches. The former category is shown as yellow triangles, and the latter as red rhombuses. The overall trend line of both categories (blue) is preserved for reference. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

more speculative note we also argue that there is also some reason to believe that its *depth* may be a relevant factor, albeit a weaker one.

6.2. Frequency of commits and size of organization

From the collected data we find that there is a tendency for developers of larger organizations to commit less frequently, but with p values ranging from 0.14 to 0.18 for all size metrics (see Table 2), the correlation is not strong enough to draw any conclusions.

6.3. Size of commits and ways of working

Looking at the graphs shown in Section 6.1, it is clear that — unsurprisingly — the size of the organization is not the only factor in determining continuity: some of the cases are well below or above of the trend line. This poses the question whether the cases above and below, respectively, have anything in common. An interesting data point in this context is the qualitative information received from the practitioners with regards to ways of working in each case. Labeling the cases as either prescribing integration directly with the mainline or integration via local branches shows the two cases of the former category below the trend line, and the four cases of the latter category all above the trend line. Fig. 4 demonstrates this by splitting the plot shown in Fig. 2 into the two categories, with separate trend lines.

Another item of interest which presents itself when exploring the data is that the behavior of in-house developers is, in some cases, very different from that of external consultants. Of the six cases, three make heavy use of external consultants (more than 25% of commits): cases B, D and F shown in Fig. 4. In cases B and D — located above the trend line and integrating via local branches rather than directly with the mainline — external consultants make significantly larger commits than internal developers: the average external commit is approximately twice as large as the average internal commit (across a total of 1277 commits by 236 committers).

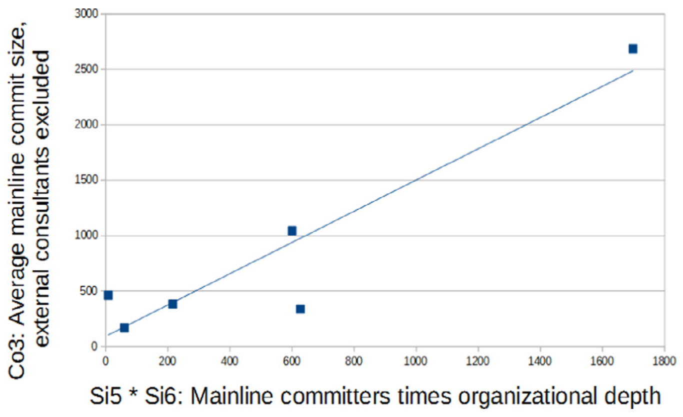

Fig. 5. Plot of number of mainline committers (Si_5) times organizational depth (Si_6) versus average mainline commit size (Co_3), excluding commits by external consultants.

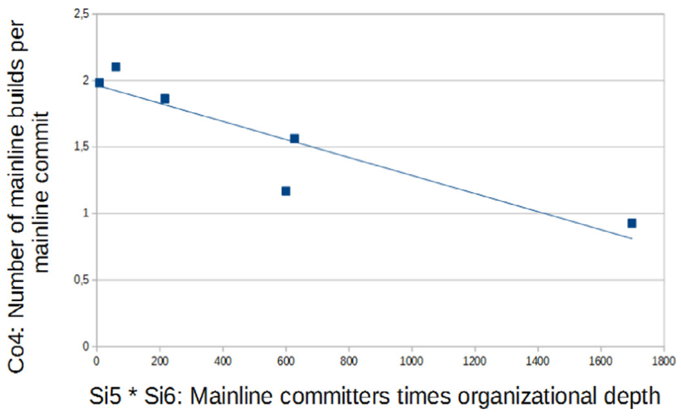

Fig. 6. Plot of number of mainline committers (Si_5) times organizational depth (Si_6) versus number of mainline builds per mainline commit (Co_4).

In case F, however — well below the trend line and emphasizing direct mainline integration — the average size of commits is approximately the same across both populations.

While these findings are far from conclusive, we argue that they make intuitive sense and that they do point at a very interesting phenomenon. It is entirely conceivable that external consultants, not physically present where most of the development is conducted and therefore presumably missing out on much of the day-to-day communication and social interaction, tend to work more closely with other external consultants on local branches and staying there longer before integrating with the development mainline. That is, unless directly integrating with the mainline as individual developers — rather than delivering as a team from a local branch — is the mode of operations, whereupon such differences are mitigated or even eradicated.

The data collected from the six primary cases is unfortunately not sufficient to conclusively answer this question. Based on the above line of reasoning, however, regarding the behavior of external consultants as noise in the overall data set (as it is presumably unrelated to the size of the area of direct change impact) and consequently removing all commits by external consultants from the data set produces a much stronger correlation between organizational size and continuity. To exemplify, the two-tailed p value of $\mathbf{Si_3/Co_3}$ correlation decreases from 0.004796 to 0.001683. Correlation with number of mainline committers ($\mathbf{Si_5/Co_3}$) decreases similarly, and when combined with the depth of the organization (as shown in Fig. 3) decreases further in Pearson's test of linear correlation to a p value of 0.006686 and an R value of 0.932 (see Table 2 for reference). This correlation is also shown in Fig. 5.

6.4. Number of builds and size of organization

As shown in Table 2, the size of the organizations correlates not only with the average size of commits, but also the frequency of builds relative to the number of commits. This correlation — with depth of the organization taken into account, as in Section 6.1 — is plotted in Fig. 6. Its Pearson's p and R values are 0.01293 and −0.906, respectively.

The reason that this number is interesting is that it provides an indication of the capability to build and test every change to the source code independently, rather than batching multiple changes into one build (see Section 4.2 for a more in-depth discussion).

Looking at the values of the vertical axis the reader may reflect that even at the low end they are all above or close to one build per commit, which intuitively should still be fine. It shall be noted, however, that the data includes a large number of tqextraneous builds: manually triggered builds, scheduled off-hours builds, retries et cetera. As a consequence, the number of builds acting on newly introduced changes is much smaller.

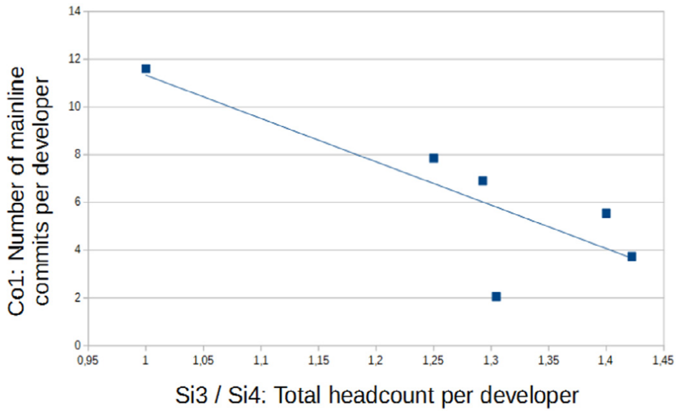

Fig. 7. Plot of total headcount (Si_3) over number of developers (Si_4) versus number of commits per developer during the studied two month period (Co_1).

6.5. Frequency of commits and composition of organization

Generally speaking, all three metrics related to the "width" of the organization (Si_3, Si_4 and Si_5) correlate with the same continuity metrics (see Table 2). This is not surprising, as they are themselves correlated: an organization with larger headcount tends to have more developers, which implies more people committing code. They do not correlate perfectly, however, and there is some variance in proportion of developers to total headcount — that is, people whose main responsibility it is to actually develop software as opposed to working on the software product in any capacity, including developers, testers, line managers, product owners et cetera (see Section 4.1.2). Exploring this variance reveals an interesting link to developer behavior, shown in Fig. 7. Incidentally, the percentage of the organization contributing to development has also been identified as a relevant metric for software quality in related work Nagappan et al. (2008).

Pearson's test of linear correlation yields a two-tailed p value of 0.04519 and an R value of −0.821. In other words, the correlation is not as strong as the ones presented in Section 6.1, but highly interesting nonetheless. It is worth pointing out that what this plot shows is not that fewer commits are made per engineer if a lower proportion of them are developers (which would be wholly expected, and indeed confirmed by the data collected in this study). Instead, it suggests that the behavior of those who actually are designated developers changes the smaller the fraction of the workforce they constitute.

One could argue that this finding is counter intuitive: the more personnel that is available in "supporting" roles, one might reason, the more developers should be able to focus on writing and committing code. What this data implies is that the opposite is happening with regards to committing frequently, with the caveat that it says little about *productivity* per se. On the other hand, one might also speculate that an organization with a low proportion of developers is less focused on actual software development as a craft than it is on other aspects of product development, such as project planning, requirements management, documentation and reviews, and that development practices such as continuous integration suffer as a consequence.

On a side note, it is worth noting that even in the most continuous case the number of commits per developer is only 11.6 over a two month period — a far cry from the prescribed minimum of one per day. One might be tempted to interpret this as evidence that the studied cases are indeed not practicing continuous integration at all. Both as researchers and practitioners, however, we consider this finding to be extremely interesting, for several reasons.

First, regardless of the pace of integration, these projects are undeniably practicing continuous delivery: new fully evaluated and ready-to-deploy system revisions are produced many times per day. Moreover, the infrastructure, the automation and the essential practices (Humble and Farley, 2010) are there, yet the integration frequency is surprisingly low. The implication of this is that the popular claim that continuous integration is a prerequisite or enabler of continuous deliver and/or deployment Rodríguez et al. (2016); Olsson et al. (2012) may need to be reconsidered, or at least qualified.

Second, this finding is at odds with the popular image of continuous practices. As discussion in Section 2, there are numerous blog posts and conference talks on successful continuous integration implementations, even at large scale. It deserves to be pointed out, that this is very similar to the image the studied company projects as well, not just externally, but also internally: in our experience from conducting the study, the general perception within the company is that of highly successful and high frequency implementation of continuous integration. Again, this is understandable as from a product point of view as new product revisions indeed are produced both rapidly and frequently, but as it turns out that does not necessarily mean that this matches the behavior of the individual developer. We argue that it is quite probably that this mismatch of perception and practice is not unique to the studied cases, but may indeed manifest in other ostensibly continuous large scale projects in the industry. Indeed, if we look closer at some of the cases held up as leading examples, we see evidence of this. In one such case[16] 100–150 changes are integrated every day by 400 developers. This is a seemingly high number, but still only adds up to an average of one commit per developer every three or four days, which also falls way short of the prescribed minimum of one per day.

Third, it supports the reasoning of Duvall (2007) that the word "continuous integration" is actually something of a misnomer: arguably, a better word would be "continual integration". We observe that there is not so much a black or white dichotomy where either one practices continuous integration or one does not. Instead, there is an interplay of factors, each of which may be more or less developed in the individual case. Particularly, given enough developers, it is entirely possible to "continuously" integrate, build, test, package and deliver hundreds of source code changes every single day, even though the individual developer checks in once a week or once a month.

6.6. Context metrics

The gathered context metrics (Cn_{1-5}) show that each case was embedded software in the telecommunications industry and each used Git and Jenkins for SCM and CI server, respectively. They had been in development for three to eight years, but no correlation with continuity could be established.

6.7. The case for design for manufacturability

As discussed in Section 1, the question driving this work is not *whether* large scale software development can adhere to continuous integration — we know from experience that this is possible. The six primary cases are a case in point: they are all integrated into a very large system, continuously integrating and delivering new versions at a very rapid rate. We also know, however, that it can be problematic, and related work suggests that the solution has to do with modularity (Roberts, 2004; Bellomo et al., 2014; Olsson et al., 2012). It is important to understand, however, that there are different types of modularity: a runtime modular system may well be developed, built and tested as a monolith, and consequently qualify as but a single area of direct change impact. Hence, the *continuity* of continuous integration would seem dependent on the architectural design of the system.

The architectural style of microservices focuses on methods of splitting a software system into small, relatively autonomous fragments, which can then be deployed, scaled and upgraded in their own independent life cycles. Its proponents stress the importance of considering the structure of the organization developing the system so that teams can take end-to-end ownership of "their" services — in effect, making Conway's law work for them (Newman, 2015). In this regard, the microservices style may be considered to be at one extreme of the modularity spectrum. That being said, a microservices style of system architecture is not without its problems. Particularly, given its heavy use of modern cloud oriented infrastructure, it can be difficult to achieve in an embedded software context — a context already posing significant challenges to the adoption of continuous integration (Mårtensson et al., 2016) — often due to resource constraints and strict real-time requirements leading to a higher level of integration and connectedness between components that, in turn, violates the principles of a microservices architecture.

We believe that not only do the results presented in this paper strongly support the concepts underpinning the microservices architectural style, but also in a broader sense point towards the importance of considering the *manufacturability* (or perhaps more accurately the *developability*) of software in architectural decisions: how can the software be designed and partitioned not just to achieve desired runtime characteristics, but to optimize its development and integration? We argue that while the concept of design for manufacturability has always been a core concern for most engineering disciplines, it has largely been neglected in software engineering. This is not entirely surprising, as the concept of manufacturing does not translate cleanly to the realm of software. Nevertheless we believe that it is applicable in the slightly modified sense of developability and that data presented in this work suggests that the size of direct change impact — and consequently modularity — is a crucial factor.

7. Validation

As described in Section 3.3, ten individual interviews with senior software engineers in five independent companies (one of

[16] https://www.thoughtworks.com/insights/blog/case-continuous-delivery.

Table 3
Validation interview questions.

Interview Questions	
IQ_2	Does each developer push his/her commits to the main track, or do the developers commit to a team or feature branch, which is then integrated into the main track?
IQ_3	Would you agree that there is a correlation between the size of the software product and the continuity of continuous integration?
IQ_4	Would you agree that there is a correlation between the size of the organization and the continuity of continuous integration?
IQ_5	Which is most important for the correlation between scale and the continuity of continuous integration – size of the organization or size of the software product?
IQ_6	Would you agree that there is a correlation between hierarchical depth of the organization and the continuity of continuous integration?
IQ_7	Would you agree that there is a correlation between the proportion of software developers (to the overall headcount of the developing organization) and the continuity of continuous integration?
IQ_8	Would you agree that there is a correlation between the "integration time" modularity of the software architecture and the continuity of continuous integration?

them the company of the primary cases, see Section 5.1) were conducted in order to validate the findings presented in Section 6 and to investigate their generalizability. All five companies operate in separate industry segments: military aeronautics, telecommunications, road vehicles, video surveillance and military electronics and radar, respectively. All five companies are members of Software Center.[17]

7.1. Interview guide

The interview guide first queried the extent of the interviewees' experience: how many years of industry software development experience did they have, and what was the largest development project they had been involved in, in terms of headcount? This contextual information was included because — assuming that the findings in Section 6 are valid — the extent to which engineers have experienced their effects may depend on the situations they have been exposed to.

Having explained the meaning of size and "the continuity of continuous integration", the remaining interview questions (see Table 3) were used to query the interviewees' opinions based on their experiences from their reference projects. For each question, the interviewees were given the option to decline to answer in case they did not feel that they had the required experience or insight.

The interviewees were asked to rate their answers to questions $IQ_{3-4,6-8}$ on a scale of 1 to 5, with 1 representing "Do not agree" and 5 representing "Fully agree". Similarly, they were asked to rate their answers to IQ_5 on a scale of 1 to 5, with 1 representing "Size of the organization" and 5 representing "Size of the software product". The full interviewee responses are available in Appendix A while an overview of their responses to questions IQ_{3-8} is shown in Fig. 8.

Questions addressing the behavior of external consultants (discussed in Section 6.3) were deliberately omitted from the interview guide. This was because while the differences in behavior exhibited in the primary cases are interesting, we believe that further data is required before drawing any clear conclusions suitable for validation in this format. The questions were also kept at a relatively high level of abstraction, e.g. not going into details with

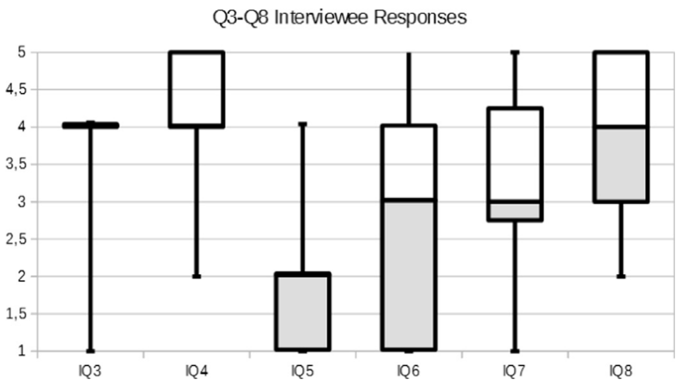

Fig. 8. Candlestick chart of interviewee responses to questions IQ_{3-8}, displaying quartiles zero through four for each question.

regards to types of continuity, as we felt that the introduction of such specific concepts in an interview situation would only serve to muddy the waters and risk confusing the interviewees.

7.2. Interview responses

The responses clearly show that engineers are widely experiencing a correlation between size and continuity, confirming the quantitative data of our study (IQ_{3-4}). In particular, the responses suggest a correlation in the case of organizational size. Indeed, when asked which might be the more important factor (IQ_5), the majority of interviewees considered organizational size to be the dominant factor. It is also interesting to note that a number of the interviewees spontaneously commented that size does not *prevent* continuity, but rather that it comes down to how the architecture of the product (and the organization) deals with that size: "this works if you don't have dependencies between the teams", "this depends on the architecture", "with decoupled components and by allowing a certain margin of error you can reduce the impact" and "there are ways you can architecturally mitigate [the problems]". That being said, there were dissenting voices, e.g. stating that size is not the important factor but rather "the goal of the organization is what is essential". Meanwhile, others agreed that there is a strong correlation, but provided the additional perspective that not only does size affect the ability to work continuously, but also drives the need, saying that particularly a "large project needs continuous integration but it is hard to work that way".

Interviewees gave very mixed responses with regards to correlation with organizational hierarchies (IQ_6). Some stated that it is completely irrelevant, while others believed it to be an important factor. One take on this issue is that hierarchies can have an adverse effect on the ability to unite on a shared vision, which in turn affects the ability to adopt continuous integration. As one interviewee put it, "it's about communication, culture and common understanding". In summary, we find that with regards to hierarchical depth in the development organization, the interviewee responses are as inconclusive as the quantitative data. That being said, we see significant potential for further work in this area to better understand the mechanisms at play and attempting to answer the question of why senior engineers in the industry display such divergence in their experiences (see Section 9.1).

The question concerning the composition of the organization (IQ_7) yielded largely positive responses. The interviewees reflected that, counter to the notion that more supporting roles would enable developers, it is harmful in that it "slows down the process [because it] removes the responsibility from developers" and "a lot of boards and forums slows down the development process". One went so far as to say that "this is the most important issue", since

[17] http://software-center.se.

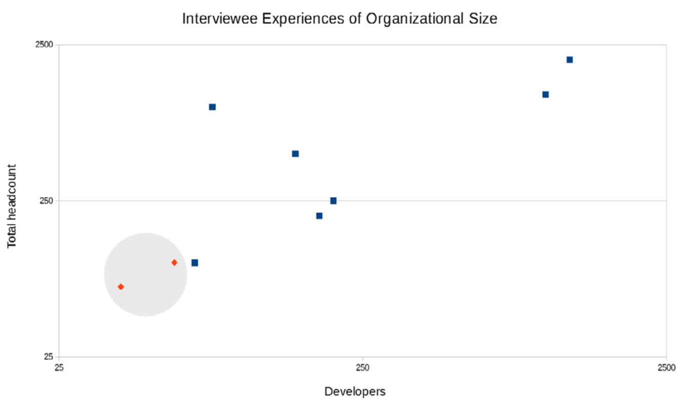

Fig. 9. Overview of the organizational size of the projects the interviewees reported experiences from. Cases where commits were made via team or feature branches are represented by blue squares, while cases where individuals integrate directly with the mainline are represented by red rhombuses and highlighted by the shaded area. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

"there are so many people who don't understand software". Interestingly enough, another interviewee from the same company and indeed the same project took the opposite view: "there is no relation between software work and other tasks". One may speculate as to the cause of such disparate views — we find it plausible that it is related to the individual's own background and role in the organization.

That being said, we feel compelled to comment on the undertone of frustration and urgency we experienced from several of the interviewees in response to this question. It was clear to us that what they perceived as the inability of an organization dominated by "people who work with papers" to understand software in general and continuous integration in particular was a subject they felt very strongly about. This cultural aspect and its impact on continuous integration practice is also a subject we have discussed in previous work (Mårtensson et al., 2016).

The final question concerning the existence of a correlation between continuity and modular architecture (IQ_8) received generally strong agreement with the majority of interviewees scoring 4 or 5. Their comments offer further support of our interpretation of the quantitative data, e.g. stating that "if the system is large and you don't have modularity you have a constant war between changes and conflicts between teams". Interestingly, one interviewee held what might be considered an opposite view: yes, there is a correlation, but "it is better to have a fully integrated product" as this "drives a common vision" which they felt was the single most important factor in achieving continuity.

One interesting observation can be made from studying IQ_2, querying the size and ways of working of the projects the interviewees have experienced. Comparing their stated ways of working — committing directly to the mainline or via team and/or feature branches — to the size of their respective organizations one finds that it is only the smallest projects which actually utilize the former strategy (see Fig. 9). This way of working, in turn, appears from the quantitative data to be linked to the continuity of continuous integration (see Section 6.3).

A full report of interviewee comments is presented in Appendix A.

7.3. Validation summary

In summary, we find that the interviewee statements of ten senior engineers in five independent companies confirm our analysis of the data collected from the primary cases:

- Engineers in industry do see a correlation between size and the continuity of continuous integration, particularly when it comes to organizational size.
- There is no consensus among engineers in the industry regarding any correlation between hierarchical depth in the organization and the continuity of continuous integration.
- Engineers in industry do see a correlation between organizational composition and the continuity of continuous integration.
- Engineers in industry do see a correlation between modular architectural styles and the continuity of continuous integration.
- There is a correlation between organizational size and ways of working with regards to integration and branching.

8. Threats to validity

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

8.1. Threats to construct validity

In this work we have presented the hypothesis that it is the size of the area directly impacted by changes that matters. This is what we have studied, and the statements by interviewed practitioners lend some support to this hypothesis, but it can not be ruled out that other definitions of size — equally applicable to the studied cases — may be as relevant or even more so. This can be interpreted as a threat to construct validity.

Our position is that the study clearly shows that size matters. The exact details of the underlying mechanisms are unclear, however. Consequently any attempt at defining the precise boundaries of the entity the size of which matters are going to include a certain element of speculation, as one is forced to reason about which connections matter and which do not. To exemplify, if one is overly generous one might argue that the entire Internet is a single "system". If one is overly conservative, on the other hand, one might argue that a single source file is its own isolated entity. The answer, if indeed there is a single answer, clearly lies somewhere in between, and we argue for what we consider to be a reasonable definition based on experiences and observations (see Section 1). As long as this caveat is kept in mind, we do not believe this fact poses any significant threat to construct validity.

8.2. Threats to internal validity

Of the 12 threats to internal validity listed by Cook et al. (1979), we consider Selection, Ambiguity about causal direction and Compensatory rivalry relevant to this work.

Selection The studied cases were not randomly selected, but rather purposively sampled: they were selected to create a population of accessible cases as similar as possible in all respects but their size. Similarly, interviewees were purposively sampled to represent as wide a segment of the industry as possible. Considering the rationale of these samplings and the fact that the resulting data sets are mutually supportive, however, 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. In other words, there is no threat to the validity of the study per se. That being said, we do argue that causation is a plausible explanation for our findings and that e.g. larger organizations lead to reduced continuity, but the precise relationships deserve more in-depth attention. Consequently, we consider this to be an important topic for further work.

Compensatory rivalry When performing interviews and comparing scores or performance, the threat of compensatory rivalry must always be considered. In our validation interviews, however, the questions (see Table 3) were deliberately designed to be value neutral by assessing correlation, rather than judging own capability, performance or skill. That being said — in line with our experiences from previous work — we found the interviewed engineers more prone to self-criticism than to self-praise.

8.3. Threats to external validity

The quantitative data is gathered from six cases within a single company. While representing a total body of 1852 engineers, it is conceivable that the findings from these cases are only valid for the one company, or the one industry segment. For this reason the validation was designed to bring in perspectives from a larger number of companies developing other types of products. By confirming the findings from quantitative data in one case with statements from engineers in five companies operating in separate industry segments, we argue that this threat has been mitigated.

9. Conclusion

In this paper we have studied the correlation between the size of software development efforts and the ability to practice continuous integration. Based on findings in related work and our own observations as both researchers and practitioners, we have hypothesized that the size of the development context directly impacted by any changes made by the developer — demarcated by explicitly versioned dependencies — correlates with the *continuity* of continuous integration within that context.

To investigate this hypothesis we have identified six size metrics and four continuity metrics and collected data from six industry cases, analyzing 5071 commits made by 1049 unique committers in organizations encompassing a total of 1852 engineers over a two month period. During this process we found that several of the attempted metrics — lines of code, cyclomatic complexity and developer behavior on team and/or feature branches — were, for reasons discussed in Section 5.2, not feasible. While not the focus of the study, we consider this to be an important finding in itself: measuring product size may be relatively straight forward, but for comparison of a small set of heterogeneous cases it is arguably not a conducive approach.

In addition, we note that while the absolute number of commits may appear to be great, per individual developer it is actually surprisingly low: approximately 2.4 commits per committer and month. As discussed in Section 6.5, this is highly interesting for three reasons. First, it demonstrates that a large enough project can continuously *deliver*, without each individual engineer necessarily *integrating* very frequently (although the total number of integrations in the system may be very high — the numbers in terms of organizational size and number of integrated changes in the product are in fact similar to those reported for cases held up as leading examples in the industry). Second, the company of the primary cases has a very ambitious continuous integration and delivery agenda, and prides itself on a number of achievements in this area. The collected data shows that even so, developer behavior may not necessarily match those expectations. Unless this mismatch is unique to the studied company, this implies that other ostensibly continuously integrating products in the industry may be facing similar challenges. Third, we observe that continuous integration is not so much a black or white dichotomy where either one practices it or not. Instead, given enough developers, it is entirely possible to “continuously” integrate, build, test, package and deliver hundreds of source code changes every single day, even though the individual developer checks in once a week or once a month.

As discussed in detail in Section 6, however, analysis of the remaining metrics reveals that organizational size clearly correlates with continuity — a finding which is validated by subsequent interviews with ten senior engineers in five independent companies operating in separate segments of the industry: military aeronautics, telecommunications, road vehicles, video surveillance and military electronics and radar. A summary of assertions, support from quantitative data and validating interviews and conclusions, respectively, resulting from this study is shown below.

Assertion: A larger software size correlates with lower continuity.

Support from primary cases: Product size metrics could not be gathered.

Support from validating interviews: Industry engineers report experiencing such a correlation, albeit to a lesser degree than organizational size. Several interviewees qualified their responses by saying that size matters, but it also depends on the software architecture.

Conclusion: We believe that it is plausible that software size negatively affects the continuity of continuous integration.

Assertion: A larger organizational size correlates with lower continuity.

Support from primary cases: A clear negative correlation with regards to size of commits and number of builds is evident in the data. There is also a tendency for larger organizations to commit less frequently, but the p value is not small enough to draw any definitive conclusions.

Support from validating interviews: Industry engineers strongly support the correlation.

Conclusion: There is a clear negative correlation between organizational size and continuity — particularly with regards to size of commits and number of builds.

Assertion: A larger proportion of developers in the organization correlates with higher commit frequency.

Support from primary cases: The collected data shows a clear tendency of developers in organizations with a higher proportion of non-developers to commit less frequently.

Support from validating interviews: Industry engineers largely support this assertion. While some disagree, others were adamant that a software development organization with a large proportion of non-developers will struggle to “understand software” and consequently experience difficulties in adopting continuous integration.

Conclusion: We believe it is highly plausible that continuity correlates positively with the proportion of developers in the organization. That being said, even in the most continuous of the studied cases falls far short of the often cited goal that “each person integrates at least daily” (Fowler, 2006).

Assertion: A modular architectural style supports continuous integration.

Support from primary cases: The studied cases are all modules in a larger system — a system which is itself continuously integrated, rapidly and frequently producing new release candidates. The larger the organization of each respective module, however, the less continuously it is developed.

Support from validating interviews: Industry engineers strongly support the notion that architecture plays a key role in enabling continuous integration, stressing a number of factors including competition and conflicts between teams, coordination between parts of the software and the time required for building and testing.

Conclusion: In line with claims in related work and the correlations found in the studied data, we argue that breaking down large systems — software systems but perhaps even more importantly organizations — into smaller pieces is a key enabler for continuous integration at scale.

Assertion: A deeper hierarchy of the organization correlates with lower continuity.

Support from primary cases: The data does not show any clear correlation.

Support from validating interviews: There is no consensus among the interviewed industry engineers.

Conclusion: This study offers no support for the notion that hierarchical depth is an important factor in general, but does not rule out that it may be relevant in particular circumstances.

Assertion: External consultants exhibit a less continuous behavior than internal developers when integrating via team or feature branches.

Support from primary cases: Half of the studied cases rely heavily on external consultants (25% of more of commits). Of these, in the two using team or feature branches the external consultants made approximately twice as large changes, on average. In the third case, integrating directly with mainline, there was no significant difference between in-house developers and external consultants.

Support from validating interviews: Not addressed in interviews.

Conclusion: We do not believe that the data set allows for any definitive conclusions, but it does suggest an interesting phenomenon.

Assertion: Integrating directly with the mainline, as opposed to via team or feature branches, leads to increased continuity. Furthermore, larger organizations tend to use team or feature branches.

Support from primary cases: Among the studied cases there is a clear difference between cases where developers integrate directly with the mainline, and cases where they do not.

Support from validating interviews: Analysis of the background of the interviewed engineers reveals that only the smallest of the organizations they represent practice direct integration with the mainline.

Conclusion: Our study suggests that direct integration with the mainline is superior from a continuity point of view, but that larger organizations are unable or unwilling to work in such a way.

9.1. Further work

Apart from questions raised in Section 8, we believe that this study also opens up other promising and important avenues for further work.

The argument that a key enabler for continuous integration is the successful communication of a shared vision in the organization — touched upon from several points of view in the interviews — deserves further attention. It begs the question of how such a successful vision can be formulated and how it can be communicated. Any interplay with organizational size and/or depth is particularly interesting, as our study strongly suggests that larger organizations struggle with the adoption of continuous integration. We hypothesize that one explanation for this could be the difficulty of uniting on a single goal in a large organization. Another explanation could be a tendency for engineers to focus on and optimize for their small part of the whole, as the complete system is much too large for them to overview.

A second avenue is that of manufacturability. This work suggests that as a consequence of architectural design decisions, software can be more or less easy to integrate, and by extension to develop. It is far from clear what those architectural decisions are, however. Modularity — in particular “integration time” modularity — is clearly relevant, but there are many ways to achieve modularity and many ways to handle dependencies between modules and mapping of organization to modules. The statement by one of the interviewees that their earlier “transfer to cross functional teams was not good” and that their “previous component-based organization was faster” prompts a number of questions for the research community to address.

Third, the indication that external consultants may exhibit less continuous behavior than their in-house counterparts is highly interesting and worth of further study.

Fourth, we believe that the question of whether the observed mismatch between internal perception of continuous integration and evident developer behavior is unique to the studied case is important to address, as well as investigating the reasons for this mismatch: what precisely is keeping the developers from integration more frequently?

Last but not least, while this work reports on the size of areas of direct change impact and correlations with the continuity of continuous integration, we can also identify another dimension: the time dimension, or more accurately the life cycle dimension, of such an area. In other words, how far in the life cycle of the product is that context decoupled (and actually not directly impacting) from that of the larger product or system? Here, as in Section 6.7, we find microservices at one end of the spectrum, where the parts of the system are first integrated in deployment. At the other end we find a monolithic system with a single source, where the point of integration between every developer of the product is immediately upon code commit. We consider the question of whether and how this time dimension affects development practices in general, and continuity in particular, to be a highly relevant area of further work.

Acknowledgment

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

Appendix A

This appendix includes the complete validation interview response material, presented per interviewee. See Section 7 for further detail and a list of interview questions.

Company A, Interviewee 1

Years of industry software development experience: **16**. Total headcount in project: **500**. Developers in project: **150**. Hierarchical depth: **5**.

IQ$_2$:
Response: Committing to a team or feature branch (one developer commits for the whole team).

IQ$_3$:
Response: 4
Comments: “Continuous integration is more difficult with a large, complex software product. Complexity is more important than lines of code. Complexity means dependencies.”

IQ$_4$:
Response: 2
Comments: "The number of people is not interesting. Every team should deliver independently. This works if you don't have dependencies between the teams. It does not work if you have such dependencies. However, if the organization scales, continuous integration infrastructure must scale, too. Otherwise queues will [reduce] continuity."

IQ$_5$:
Response: No response
Comments: None

IQ$_6$:
Response: 1
Comments: "Not at all interesting. This affects information flow, but not at all continuous integration."

IQ$_7$:
Response: 1
Comments: "No, there is no relation between software work and other tasks."

IQ$_8$:
Response: 5
Comments:"Continuous integration is much easier with modularity."

Company A, Interviewee 2

Years of industry software development experience: **10**. Total headcount in project: **500**. Developers in project: **150**. Hierarchical depth: **5**.

IQ$_2$:
Response: Committing to a team or feature branch (one developer commits for the whole team).

IQ$_3$:
Response: 4
Comments: "A small project doesn't need [continuous integration]. A large project needs continuous integration but it is hard to work that way."

IQ$_4$:
Response: 5
Comments: "The organization! [sic]"

IQ$_5$:
Response: 1
Comments: None

IQ$_6$:
Response: 5
Comments: "Much harder to work the same way with many hierarchy levels, at least harder to communicate a unified way of working."

IQ$_7$:
Response: 5
Comments: "In this project it is a problem that there are so many people that are not developers. There are so many people who don't understand software. There is not enough focus on following the processes for software development. This is the most important issue!"

IQ$_8$:
Response: 3
Comments: "Integration of binaries might cause lots of problems if you have problems with interfaces then it is better to integrate source code."

Company B, Interviewee 1

Years of industry software development experience: **10**. Total headcount in project: **70**. Developers in project: **40**. Hierarchical depth: **2**.

IQ$_2$:
Response: Committing to a the main track.

IQ$_3$:
Response: 1
Comments: "A lot of people causes a need for continuous integration. A large product does not imply that it is more difficult to work with continuous integration."

IQ$_4$:
Response: 5
Comments: "You want to deliver more often if a lot of people work with the same components as you do. Extra work when big changes must split up into small deliveries instead of be delivered as a monolith."

IQ$_5$:
Response: 1
Comments: None

IQ$_6$:
Response: No response
Comments: None

IQ$_7$:
Response: 3
Comments: None

IQ$_8$:
Response: 5
Comments: "Better modular architecture gives more freedom and development is more continuous."

Company B, Interviewee 2

Years of industry software development experience: **20**. Total headcount in project: **100**. Developers in project: **60**. Hierarchical depth: **3**.

IQ$_2$:
Response: Committing to a the main track.

IQ$_3$:
Response: 1
Comments: "No, not as our architecture is of today. Scale does not affect the way of working. Our code base is divided into [hundreds of] modules."

IQ$_4$:
Response: 4
Comments: "Not regarding headcount, but there is a correlation to the structure of the organization. Our previous component-based organization was faster. The transfer to cross functional teams was not good. Integration tests are component-based and there is no support for them in the new organization."

IQ$_5$:
Response: 2
Comments: None
IQ$_6$:
Response: 4
Comments: None

IQ_7:
Response: 4
Comments: “Requirements on for example test coverage affects software development. New activities are added to what the developers must do.”

IQ_8:
Response: 2
Comments: “The developer has about 700 of 900 modules in their development environment. The system is built as a monolith. I don’t think integration should be easier with integration time modularity.”

Company C, Interviewee 1

Years of industry software development experience: **25**. Total headcount in project: **100**. Developers in project: **70**. Hierarchical depth: **7**.

IQ_2:
Response: Committing to a team or feature branch (each team works on its own mainline, with each component running on its own CPU).

IQ_3:
Response: 4
Comments: “It depends on the architecture and debugging. Modularity and self-containment affects this.”

IQ_4:
Response: 4
Comments: “We are too many people. Much too little code is produced. We are trying to introduce a continuous integration way of working and this is hard as the right information is never on the right place.”

IQ_5:
Response: 1
Comments: None

IQ_6:
Response: 2
Comments: “It is more related to the size of the organization. By the way an organization in many levels is a bad organization. It might [result in the] organization becoming political and creates ownership for components in a bad way.”

IQ_7:
Response: 2
Comments: “This contradicts the agile way of working. A lot of boards and forums slows down the development process. Too many people are involved in an issue. Or is it. Maybe this does not affect this at all?”

IQ_8:
Response: 5
Comments: “Yes! If the system is large and you don’t have modularity you have a constant war between changes and conflicts between teams.”

Company C, Interviewee 2

Years of industry software development experience: **20**. Total headcount in project: **1000**. Developers in project: **80**. Hierarchical depth: **8**.
IQ_2:
Response: Committing to a team or feature branch (each team works on its own mainline, with each component running on its own CPU).

IQ_3:
Response: No response
Comments: “The largest [product component] is the one that [is most] continuous. Larger products work better with continuous integration.”

IQ_4:
Response: No response
Comments: “The goal of the organization is what is essential. We see that a large organization can work continuously, some other small organization can not. The interesting thing is that you formulate a common vision — we shall build a [product].”

IQ_5:
Response: No response
Comments: “Business goals and vision are interesting — to focus on speed!”

IQ_6:
Response: 1
Comments: “No. The [product component X] project has a lot of hierarchies and is working continuously. They have focused on overall vision and speed. Others have focused on specifications.”

IQ_7:
Response: No response
Comments: “920 of 1000 people in the project are braking the process. If they could focus on test models, it would help. Now when they run around with specifications and powerpoints they are just a brake.”

IQ_8:
Response: 3
Comments: “Affects a lot. Every part of the organization think their [part] is the product. They don’t share the same vision of a common product. It is better to have a fully integrated product – this drives a common vision.”

Company D, Interviewee 1

Years of industry software development experience: **16**. Total headcount in project: **1200**. Developers in project: **1000**. Hierarchical depth: **3**.

IQ_2:
Response: Committing to a team or feature branch (with occasional exceptions for small items and fixes).

IQ_3:
Response: 4
Comments: “With caveats. A larger product size requires more levels of integration, so in that way it has an effect. In a perfect software architecture it wouldn’t have as large an effect as it does today. [...] With decoupled components and by allowing a certain margin of error you can reduce the impact, but the way it is now [in our product] there is an impact. It’s what we struggle with on a daily basis.”

IQ_4:
Response: 4
Comments: “It feels as though the more developers you add the more the amount of work on branch increases. You get more parallel work going on. At the same time there are so many tests going on you can’t integrate all the time, but end up on branch, which leads to larger commits. And all this grows; if you add another team the queue grows. This leads to adding on more work [required to commit], leading to larger commits. Our way of working contributes to this — we have cross functional teams work-

ing across multiple [requirement] areas. With more decoupling I'm sure we could increase the pace."

IQ$_5$:
Response: 2
Comments: "A larger number of people creates a larger number of jobs on branch which need to be integrated. As it is right now we can squeeze in 30 commits per day. When I look at [a smaller product] they have about the same pace of integration as we do, but with a much smaller number of people. And we get all these knock-on effects, like setting up continuous integration on branches. But that's not all, it's also about the modularity of the product. You can choose not to have people working across the entire product. The difficulty here is how to stop guessing, and obtaining evidence to support architectural decisions and changes to our continuous integration machinery. It's hard to looking into one's crystal ball and see what the effects will be from e.g. testing more or less."

IQ$_6$:
Response: 1
Comments: "I don't see anything of that. The teams have a free reign, they can do any type of changes. The organization has a rather limited impact in that regard."

IQ$_7$:
Response: No response
Comments: "If you have an organization of 1200 where 5 are developers, compared to an organization of 1200 where everybody is a developer, of course there's going to be a difference. You could also argue that if we don't have team leaders, project leaders, coordinators et cetera, we wouldn't have any control. Then we would step all on one another's toes all the time. You would like to think these people [leaders, managers] serve a purpose."

IQ$_8$:
Response: 5
Comments: "I think if we achieve increased modularity we'll also see a higher pace of integration. And I think that's what [managers] often look at. We see that clearly in the KPIs we measure, managers look a lot at number of commits per day and such. But then you forget that in our way of working those commits are on branch, but that's not what's measured."

Company D, Interviewee 2

Years of industry software development experience: **12**. Total headcount in project: **2000**. Developers in project: **1200**. Hierarchical depth: **4**.

IQ$_2$:
Response: Committing to a team or feature branch (both variants exist, but working on branch is the common case).

IQ$_3$:
Response: 4
Comments: "The reason I believe that is because in order to maintain high continuity you need to keep track of your dependencies. The larger the product, the harder it is to understand how your contribution affects the product. Then there are ways you can architecturally mitigate that, but the way we work I experience [this effect]."
IQ$_4$:
Response: 4
Comments: "A large organization makes it harder to communicate between teams."

IQ$_5$:
Response: 4
Comments: "It's harder to stimulate collaboration [through organization] than it is to choose a product architecture that stimulates continuity."

IQ$_6$:
Response: 3
Comments: "You haven't included any question regarding geographic distribution. If you have a deep hierarchy that can imply that you are also geographically distributed, because we have a tendency to organize like that. But apart from that I don't think [there is any correlation]."

IQ$_7$:
Response: 3
Comments: "I'm thinking that as a developer, as opposed to [other roles], you probably see a greater benefit from maintaining high continuity. You get better feedback from being continuous."

IQ$_8$:
Response: 4
Comments: None

Company E, Interviewee 1

Years of industry software development experience: **20**. Total headcount in project: **200**. Developers in project: **180**. Hierarchical depth: **3**.

IQ$_2$:
Response: Committing to a team or feature branch.

IQ$_3$:
Response: 4
Comments: "Testing: A larger system must be tested – which takes longer time. Modularity: You have never a fully modular system – you affect other modules, you must wait for implementation in other modules and you have to wait. If you have a mature system, with only a few people involved, it might work well. There is no natural connection. It depends on the architecture."

IQ$_4$:
Response: 5
Comments: "This is more relevant [than product size]! The problem is that there are too many rebases. It is also harder to keep a good coding standard and architecture. But if the system is modular both architecturally and organizationally [it can work]."

IQ$_5$:
Response: 2
Comments: None

IQ$_6$:
Response: 5
Comments: "We have not succeeded to handle development between different parts of the organization in a good way. It's about communication, culture and common understanding."

IQ$_7$:
Response: 5
Comments: "To build internal structures that don't do anything (people working with requirements etc) slows down the process. It also removes the responsibility from developers. It is always better to implement and then re-factor than to ask for permission."

IQ$_8$:
Response: 4
Comments: None

Company E, Interviewee 2

Years of industry software development experience: **10**. Total headcount in project: **250**. Developers in project: **200**. Hierarchical depth: **3**.

IQ$_2$:
Response: Committing to a team or feature branch.

IQ$_3$:
Response: 4
Comments: "This depends on the architecture. In the best of worlds you have no correlation, but in the real world there is a correlation."

IQ$_4$:
Response: 5
Comments: "This is also an architectural problem (architecture of the organization), but more difficult to solve than the software problem. There is no efficient way of communicating within our organization right now. That is a problem as everyone must understand the status of the software."

IQ$_5$:
Response: 2
Comments: None

IQ$_6$:
Response: 4
Comments: "A deep hierarchy is harmful to continuity. It is harder to spread information through many levels in an organization."

IQ$_7$:
Response: 3
Comments: "People coming from other disciplines than software are used to that it takes longer time to transform an idea to implementation. People who work with papers will use longer time spans than people who work with software. [Software developers] work faster."

IQ$_8$:
Response: 5
Comments: "We want to deliver binaries, and are working with this. Most people shall deliver add-ons to the platform. This will make integration easier."

References

Agresti, A., Kateri, M., 2011. Categorical Data Analysis. Springer.
Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional.
Bellomo, S., Ernst, N., Nord, R., Kazman, R., 2014. Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail. In: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, pp. 702–707.
Bellomo, S., Nord, R.L., Ozkaya, I., 2013. A study of enabling factors for rapid fielding combined practices to balance speed and stability. In: 2013 35th International Conference on Software Engineering (ICSE). IEEE, pp. 982–991.
Boehm, B., 1984. Software engineering economics. IEEE Trans. Software Eng. 10 (1), 4–21.
Boehm, B.W., Madachy, R., Steece, B., 2000. Software Cost Estimation with Cocomo II with Cdrom. Prentice Hall PTR.
Briand, L.C., Morasca, S., Basili, V.R., 1996. Property-based software engineering measurement. IEEE Trans. Software Eng. 22 (1), 68–86.
Brooks, F.P., 1975. The Mythical Man-Month, 1995. Addison-Wesley Reading, MA.
Brooks, G., 2008. Team pace keeping build times down. In: Agile Conference. IEEE, pp. 294–297.
Buchgeher, G., Klammer, C., Heider, W., Huber, H., et al., 2016. Improving testing in an enterprise soa with an architecture-based approach. In: Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on. IEEE, pp. 231–240.
Cohn, M., 2005. Agile Estimation and Planning. Addison-Wesley.
Conway, M.E., 1968. How do committees invent. Datamation 14 (4), 28–31.
Cook, T.D., Campbell, D.T., Day, A., 1979. Quasi-Experimentation: Design & Analysis Issues for Field Settings, 351. Houghton Mifflin Boston.
Duvall, P.M., 2007. Continuous Integration. Pearson Education India.
Fenton, N.E., Neil, M., 2000. Software metrics: roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ACM, pp. 357–370.
Fowler, M., 2006. Continuous Integration. http://www.martinfowler.com/articles/continuousIntegration.html, [Online; accessed 12-February-2016].
Gilb, T., 1976. Software Metrics (winthrop computer systems series). Englewood, NJ: Winthrop.
Halstead, M.H., 1977. Elements of Software Science, 7. Elsevier New York.
Humble, J., Farley, D., 2010. Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Pearson Education.
Mårtensson, T., Ståhl, D., Bosch, J., 2016. Continuous integration applied to software-intensive embedded systems – problems and experiences. In Review.
McCabe, T.J., 1976. A complexity measure. IEEE Trans. Software Eng. 4, 308–320.
Nagappan, N., Murphy, B., Basili, V., 2008. The influence of organizational structure on software quality: an empirical case study. In: Proceedings of the 30th international conference on Software engineering. ACM, pp. 521–530.
Newman, S., 2015. Building Microservices. O'Reilly Media, Inc..
Olsson, H.H., Alahyari, H., Bosch, J., 2012. Climbing the "stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on. IEEE, pp. 392–399.
Perry, D.E., Staudenmayer, N.A., Votta, L.G., 1994. People, organizations, and process improvement. Software, IEEE 11 (4), 36–45.
Roberts, M., 2004. Enterprise continuous integration using binary dependencies. In: Extreme Programming and Agile Processes in Software Engineering. Springer, pp. 194–201.
Robson, C., 2002. Real world research, 2. Blackwell publishers Oxford.
Rodríguez, P., Haghighatkhah, A., Lwakatare, L.E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J.M., Oivo, M., 2016. Continuous deployment of software intensive products and services: a systematic mapping study. J. Syst. Software.
Rogers, R.O., 2004. Scaling continuous integration. In: Extreme Programming and Agile Processes in Software Engineering. Springer, pp. 68–76.
Scholtes, I., Mavrodiev, P., Schweitzer, F., 2016. From aristotle to ringelmann: a large-scale analysis of team productivity and coordination in open source software projects. Empirical Software Eng. 21 (2), 642–683.
Ståhl, D., Bosch, J., 2013. Experienced benefits of continuous integration in industry software product development: A case study. In: The 12th IASTED International Conference on Software Engineering, pp. 736–743.
Ståhl, D., Bosch, J., 2014a. Automated software integration flows in industry: a multiple-case study. In: Companion Proceedings of the 36th International Conference on Software Engineering. ACM, pp. 54–63.
Ståhl, D., Bosch, J., 2014b. Modeling continuous integration practice differences in industry software development. J. Syst. Software 87, 48–59.
Ståhl, D., Hallén, K., Bosch, J., 2016. Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework. Empirical Software Eng. 1–29.
Stigler, G.J., 1958. Economies of scale. the. JL & Econ. 1, 54.
Sturdevant, K.F., 2007. Cruisin'and chillin': Testing the java-based distributed ground data system" chill" with cruisecontrol system" chill" with cruisecontrol. In: IEEE Aerospace Conference. IEEE, pp. 1–8.
Sunindyo, W.D., Moser, T., Winkler, D., Biffl, S., 2010. Foundations for event-based process analysis in heterogeneous software engineering environments. In: 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). IEEE, pp. 313–322.
Woskowski, C., 2012. Applying industrial-strength testing techniques to critical care medical equipment. In: Computer Safety, Reliability, and Security. Springer, pp. 62–73.
Yüksel, H.M., Tüzün, E., Gelirli, E., Bıyıklı, E., Baykal, B., 2009. Using continuous integration and automated test techniques for a robust c4isr system. In: 24th International Symposium on Computer and Information Sciences. IEEE, pp. 743–748.

Daniel Ståhl is continuous integration subject matter expert and architect at Ericsson AB. He has a background of nine years of software development, integration and architecting in the telecom industry, where his work primarily revolves the application of continuous integration and delivery practices to multinational enterprise scale organizations. He received a MSc degree from Linköping University, Sweden, in 2007.

Torvald Mårtensson is principal engineer in systems integration at Saab AB. He has a background of eleven years in the aeronautics industry and another eight years in the telecom industry. His work has primarily revolved around systems integration and system testing of large-scale software systems. He received a MSc degree from Linköping University, Sweden, in 1997.

Jan Bosch is professor of software engineering and director of the software research center at Chalmers University Technology in Gothenburg, Sweden. Earlier, he has worked as Vice President Engineering Process at Intuit Inc and as head of the Software and Application Technologies Laboratory at Nokia Research Center, Finland. Before joining Nokia, he headed the software engineering research group at the University of Groningen, The Netherlands, where he holds a professorship in software engineering. He received a MSc degree from the University of Twente, The Netherlands, and a PhD degree from Lund University, Sweden.