FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 43 (2018) No. 4
ISSN 0867-6356
DOI: 10.1515/fcds-2018-0015 e-ISSN 2300-3405

Measurement and Impact Factors of Speed of Reviews and Integration in Continuous Software Engineering

Miroslaw Staron [*], Wilhelm Meding [†], Ola Söder, Magnus Bäck [‡]

Abstract. Continuous integration and continuous software deployment depend on the mix of automated and manual activities. The automated build and test processes are often intertwined with manual reviews and bug-fixing activities. In this paper, we set off to study how these manual and automated activities influence the speed of reviews and integration. We conduct a case study of two companies developing embedded software, measure the time required for reviewing and integrating software code (alias speed), and conduct a workshop to identify factors which explain the quantitative results. Our results show that the measurement of speed is a good alias for calendar time and triggers improvements better than using measures for velocity. We have also found that the distribution of code repositories, frequent reminders and team proximity decrease the time needed to deploy the software. Our findings are that there is a difference in the structure of code repositories between the fast and slow integration cases, which contributes to the debate on the pros and cons of different repository structures in modern companies.

1. Introduction

Continuous software integration and deployment aims to improve the quality of software products and their availability to the market [5]. However, they require software development to progress at a higher speed than today and work in ecosystems of software development organizations [6]. In order to achieve this higher speed, companies focus on customer agility, customer analytics and optimization of software development towards faster deliveries. Which measure of speed gets chosen is important in this context as different types of measures can lead to different effects.

One of the challenges in achieving high development speed is the combination of automated and manual activities – certain activities need to be done manually, e.g.

[*]Chalmers | University of Gothenburg, Gothenburg, Sweden, miroslaw.staron@gu.se
[†]Ericsson, Sweden, wilhelm.meding@ericsson.com
[‡]Axis Communications, Sweden, ola.soder, magnus.back@axis.com

code reviews. In modern companies, code review is part of the process of delivering the code from the team branch to the product's main branch. The goal of these reviews is to keep the quality of the code (and the product) high, and it can lead to effects such as spreading the knowledge of the code within the team [2]. However, conducting this manual activity can slow down the process of delivery, as it is based on the availability of the reviewers and the possibilities to find and remove problems in the code (not only defects).

Typical measures of speed of activities, used in industry, are variations of the measure of velocity (e.g. story points per sprint, number of integrations per day, [1]). Although these measures help companies to increase their customer responsiveness, they are dependent on two factors: estimated size of the work done and period of time – velocity quantifies the size in terms of story points and the period of time in terms of sprint. However, the challenge with velocity is that it can be manipulated. An organization can adjust the way in which story points are counted, which affects the number of story points delivered during one sprint. In this way, the organization can become faster without actually increasing speed of their development. The difference between the velocity and speed is the same as the difference between the bandwidth and latency.

At the same time, the organizations cannot change estimates or add resources to projects, as they have limitations on the size of teams (e.g. the well-known "two-pizza" principle showing decrease in efficiency after the size reaches a certain limit, [7]). The increased resources in turn increase the cost, so in order to get more efficient, organizations need to find ways to increase speed without increasing the cost, i.e. without the need to increase resources.

In this paper, therefore, we address the research problem of *Which factors are important when optimizing speed of reviews and speed of integration in continuous software development?* In particular, we focus on the ability of modern companies to quantify, monitor and improve the speed of reviews and integrations software development in ways alternative to velocity. We address this by addressing three research questions:

1. What is the most relevant and practically usable definition and measure of speed of reviews and integration in the studied companies?
2. Given the definition of the measure of speed, what is the speed of reviews, and integration in continuous software development in modern companies?
3. Which factors are perceived by the studied organizations to affect the speed of reviews and integration?

We conducted a case study at two software development companies which use continuous integration as part of their development practices. Our results show that the measure of speed which helps the companies to become faster is based on time. Although the measure of time is usually recognized as the measure of duration, we have found that using it under the alias of speed focuses the company's improvements on the activities that increase speed. Since using velocity-related measures allow for unnecessary level of subjectivity in the measurement, using velocity risks to lower

the trust in measurement results. The measure of speed which is based on time is objective and requires the companies to make process improvements in order to become faster.

The rest of the paper is organized as follows. Section 2 outlines the most important related work to our study. Section 3 describes the design of the case study presented in this paper. Section 4 presents the results and discusses threats to validity. Section 5 presents the answers to our research questions and conclusions.

2. Related work

Before the introduction of continuous integration, a lot of research has been conducted on inspections and reviews. This included the research on speed or cycle time of inspections [22] and [24]. As inspections are a much more complex process [12], involving also defect fixes and meetings, the results often include these activities and presented remarkable improvements [22] when focusing on the entire process. However, with the introduction of continuous integration, the focus of software companies changes into more narrow field of conducting code reviews continuously, on code changes (rather than entire code base when using inspections). Therefore, our research focuses on the reviews and integrations in the context of continuous integration.

Thongtanunam et al. [31] studied the process of review of open source projects and have found that slow reviews are often caused by large amount of code and previously slow reviews. We expand on their work and add new factors that can make the reviews slower than usual.

Velocity is one of the most prominent ways of measuring speed in software development [9], [1]. The concept of velocity is based on the concept of a story point [10], [8]. First, a team defines the story point in reference to their user stories and then they use this measure to quantify size of each of their stories. As a team progresses, each story is developed and therefore the team's velocity is the number of story points for the finished user stories, divided by the calendar time for the development of these user stories. The measurement of velocity is similar to the measurement of velocity in the automotive sector, e.g. km/h.

Meding has shown how velocity can be visualized at Ericsson [19]. The study showed that software development teams see burn-up and burn-down charts as good indicators of their progress, at the same time admitting the limitations of velocity as the measurement of speed.

Speed, on the other hand, has been recognized as an important factor when developing software in modern companies [4]. In particular, when the distributed software development requires fast feedback loops [6].

Current studies, like Hütterman's [13], use the measures of turnaround times for defects or features as measures of speed. They also focus on DevOps as a concept which contributes to the optimization of speed through optimization of knowledge and competence exchange.

A recent study of advantages and disadvantages of having a single repository, conducted at Google [14], has shown that distribution of code repositories has significant

advantages in the flexibility while the monolithic repository has significant advantage in terms of reuse of APIs. Our study contributes with a new angle in this discussion – having multiple repositories seems to have a positive impact on the integration speed.

3. Case study design

Our work has been designed and structured according to the guidelines for conducting and reporting of empirical studies by Runeson et al. [26], [25].

3.1. Research questions

In order to address the main research problem in our study on how to optimize speed of software development, we needed to address a number of research questions:

1. What is the most relevant and practically usable definition and measure of speed of reviews and integration in the studied companies?
2. Given the definition of the measure of speed, what is the speed of reviews, and integration in continuous software development in modern companies?
3. Which factors are perceived by the studied organizations to affect the speed of reviews and integration?

We identified the first research question as important, because we noticed the difficulties with the existing measurements of speed – the aforementioned problem with adjusting velocity by changing the way story points are estimated. Instead, the sought measure of speed needed to be objective and independent of the measurer. In order to study the aspect of the measurement of speed, we focused on a selection of phases of software development – code reviews, and integration including the integration testing. We chose these phases as we could identify that these phases are often supported by similar processes, as many of continuous integration companies use the same tools – Git for code repositories, Gerrit for code reviews, Jenkins for code integration/build and unit/integration/component testing.

Once we identified the measure of speed, which fulfilled our requirements of being objective and free from the factor of size, we applied it to two cases. This application provided us with the possibility to quantify the speed at two companies. First, this provided us with the possibility to observe how fast modern companies are. Second, it provided us with the ability to study factors affecting the speed.

The study of the factors, i.e. addressing our third research question, required both quantitative analyses and workshops. The quantitative analyses were enabled by the objective measure and provided the input to the workshops. The workshops provided us with the possibility to understand which non-measurable development practices contribute to high/low speed.

3.2. Case and subject selection

In this work, we had a unique opportunity to study two different organizations in two different companies.

Company A: The company is a large infrastructure provider from Sweden. The studied organization within this company has over 100 developers who work in a combination of Agile and Lean principles. They develop an embedded software product which has been on the market for over ten years and has a stable, mature code base. It is based on the proprietary operating system. The product is sold to the infrastructure operators, who in turn provide the services to their customers. The organization has adopted practices of continuous integration for over five years.

Company B: The company is a medium-size consumer product provider from Sweden. The studied organization within this company has over 100 developers who work according to Agile principles. They develop an embedded software product, based on Linux and they adopted continuous integration for over five years. Compared to Company A, Company B's product is much smaller (at least an order of magnitude) and has much larger variability as it is sold to consumers.

Similarities: Both companies use a similar set-up of modern software development tools, including Jenkins for continuous integration and Gerrit for code reviews. The process of code reviews, code integration and testing is the same for both companies. The size of the teams is similar too.

Differences: Testing tools and equipment are different, as the products are different. The roles in the companies, which are involved in the processes, are also different. At Company A, the reviews are done within the same team, whereas at Company B, the reviews are often done by external roles to the team. The set-up of the code repositories is also different, which we describe in the Results section (4.4).

3.3. Data collection procedure

First, we collected quantitative data from the code review and integration tools – Gerrit and Jenkins. The data was in form of measures of time (seconds) between:

- submission of the code to Gerrit and having two reviewers accepting the submission,
- code being submitted to Jenkins for integration to end of build,
- start of unit test until the end of unit test,
- start of function test until the end of function test, and

- queue time between the end of review and beginning of compilation in the main branch.

Although we collected data from both finished and unfinished revisions/integrations, we used only the data from the finished ones.

Both Company A and Company B provided us with a detailed description of their state-machine of code review and integration. These state-machines detailed the progress of each code commit. Although the details of the state-machines are not important for this study, it is important to report that the state-machines states were similar for the periods measured in our study.

In order to identify the factors affecting the speed, we reviewed the literature on code reviews, and identified 58 factors (e.g. length of comments, number of comments). During a workshop with Company B, we presented these 58 factors and the company pointed to seven factors, which they deemed as relevant. Company B has also added two factors which were not in our list. These factors were then used in correlation analyses with the integration speed.

3.4. Cross-company Workshop

In the cross-company workshop, we showed the results of the measurement to both companies (the same figures as we have in the paper) and we showed them the average times, min-max and outliers. Then the companies were asked to describe their integration process. Company B provided a diagram in advance for Company A to examine and identify differences. Once everyone provided their questions to the descriptions of the process, each company asked the other about how they perform certain activities, e.g. "How many repositories do you have?" and follow up questions, e.g. "So, how often do you integrate all the code from all repositories?"

3.5. Analysis procedures

In the analysis of quantitative data, we used descriptive statistics, visual analysis and correlation analysis. For workshops, we used brainwriting techniques [11] and moderated discussions.

4. Results

The results from our study are structured per research question, beginning with the general description of the execution of the study.

4.1. Case and subjects description

For both companies, we collected the quantitative data as planned in Section 3.3. For Company A we collected 43,217 reviews and for Company B we collected 67,324 reviews from a period of 5 years. We also collected 623 integrations for Company A for approximately six months and for Company B, 155,273 integrations for the same period.

The data sets are not balanced, but the company representatives assessed the data sets as representative for both companies.

4.2. What is the most relevant and practically usable definition and measure of speed of reviews and integration in the studied companies?

Before starting to review measures for speed in general, we reviewed the measurable concepts in this area, as prescribed by the ISO/IEC 15939 standard [21]. These measurable concepts provided us with the ability to narrow the set of measures for speed and these were:

1. Velocity – the number of story points per unit of time,
2. Speed of delivery – the time for delivering the product to the customers,
3. Speed of development – the duration of development,
4. Speed of integration – the duration of integration of different product parts into a complete product, and
5. Speed of response to change – the duration between the submitted change request and the decision made for the request.

We used these measurable concepts as input to the discussion of "What do we actually want to measure?" We intended to find the concepts that are important for the stakeholders. Given different viewpoints of different stakeholders, we could measure any of the measurable concepts, but we decided that we start with the concepts that are related to software development, i.e. quantify the measurable concept of speed of development (item #3).

We started the search for the measures by reviewing the literature from the conferences in this area and from the internet sources citing these publications. Our search criteria included the following keywords: speed, review, integration, agile, lean, software development, devops, empowered teams and lead time. We combined these keywords in the way appropriate to the search engines in IEEE eXplore, ACM Digital Library, ScienceDirect, Wiley and SpringerLink. We reviewed the relevant publications and identified a list of measures.

We discussed the measures of speed of development with a number of stakeholders at both companies. In particular, we discussed the possibility to use the following measures, which were mentioned by the companies at the beginning of the study:

1. Time for the feature to be ready for delivery to customer (after main branch delivery) [20], which is the time that the organization needs to finish the product,
2. Time for the feature to be taken up by customers [20], which is the time between the availability of a feature and its "purchase" by customers,
3. Time to arrive at 'done' according to the definition of done [20], which is the time from the start of development until the delivery criteria for a feature are fulfilled,
4. Cycle and Lead times [20], which is the time needed to finish a work package (cycle time) and the time before the work on the work package starts (lead time),
5. Velocity [20], which is the number of story points delivered per unit of time,
6. Cumulative flow [23], which is the cumulative number of items (e.g features, defects) in the development project,
7. Process cycle efficiency [20], which is the number of work items delivered during one cycle,
8. Mean Time To Detect per defect (MTTD) [13], which is the time between the integration of the code and detection of the defect in the code,
9. Mean Time To Repair per defect (MTTR) [13], which is the time between the discovery of a defect and the removal of that defect,
10. Queue Time [28], which is the time that each item was present in the project backlog,
11. Throughput [28], which is the number of items per unit of time,
12. Execution Time, which is the time of the project, from the start to the end, and
13. Feedback loop time, which is the time from the delivery of the software item until it is commented in the review.

The last two items were identified by our industrial contacts at Company A and Company B as important to consider. After the discussions with the company representatives, we decided to pursue the measures that were related to two activities in software development – review of code and integration of code (including integration testing). We focused on these two activities as we could rely on production systems to obtain accurate quantitative data for selected measures. The researchers and the practitioners from the companies selected "Time to arrive at definition of done" for the reviews. We naturally defined the relevant beginning of the activity and the end of the activity for the measurement method. These start and end points are presented in Figure 1

The definition of the measure is presented in Table 1, using the format presented in ISO/IEC 15939 standard for specifying Quality Measure Elements (QME) [21], [29].

Table 1. Measures of speed of review

QME Attribute	**Definition**
Information need	How much time does it take to review a commit? / Alias: What is the speed of software reviews?
Measurable concept	Change of source code committed to Gerrit
Relevant entities	Change
Attribute 1 Attribute 2	Commit time stamp Merge time stamp
Base measures	Time between start and end of review / Alias: Review speed
Measurement method	1. Export all revisions for the change from Gerrit 2. Sort the revisions ascending by time stamp 3. Commit time stamp (CT) is the time stamp for the first revision 4. Find the revision where all reviewers change their status to +1, +2 or when the status of the change is set to "merged" 5. Merge time stamp (MT) is the time stamp of the revision from step 4 6. Review speed is the difference between the merge time stamp and the commit time stamp (MT – CT)
Type of measurement method	Objective
Type of scale	Ratio
Unit of Measurement	Second

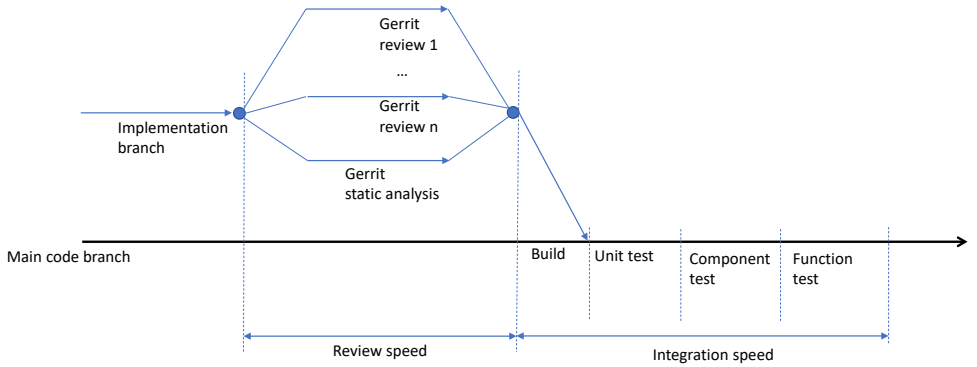


Figure 1. Start and end points of the measures

For the integration (and integration testing), we selected "Time to arrive at definition of done" and "Queue time". Table 2 presents the measures.

These definitions of measures are based on the calendar time, and therefore free from the manual estimations.

4.3. Given the definition of the measure of speed, what is the speed of reviews, and integration in continuous software development in modern companies?

The review time for Company A was 0.022 days on average (approx. 30 minutes), whereas for Company B, this was 8.5 days on average. This difference between these two companies helped us to show the relevance of further discussion between the companies; a discussion about the differences between ways of working, which can contribute to this difference in review speed. We report on these differences in Section 4.4.3.

Company A's and B's box-plots for speed of integration are presented in Figure 2.

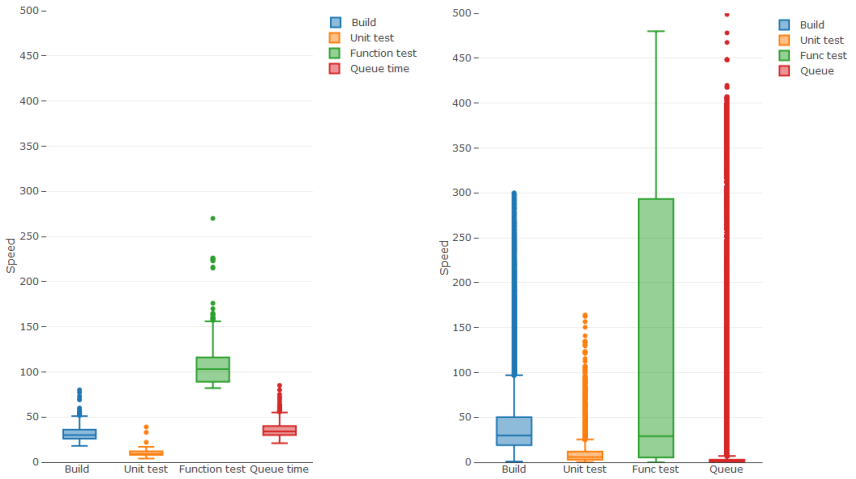
We scaled these plots to the same values on the Y-axis, in order to be able to compare the companies. The figures show that the integration speed is not significantly different between the companies. The main difference is the length of the queue – it is longer for Company B.

4.4. Which factors are perceived by the studied organizations to affect the speed of reviews and integration for the studied companies?

These results are organized into three parts. The first is the list of factors, which we identified in the literature review. The second is the analysis of correlation between

Table 2. Measures of speed of integration

QME Attribute	**Definition**
Information need	How much time does it take to integrate software into the main branch? / Alias: What is the speed of integration?
Measurable concept	Source code commit checked-in to the main branch
Relevant entities	Source code commit
Attribute 1 Attribute 2 Attribute 3 Attribute 4 Attribute 5 Attribute 6	Time stamp when merged to the main branch Time stamp for the start of build Time stamp for the start of unit test Time stamp for the start of function test Time stamp for the start of component test Time stamp for the end of function test
Base measure 1 Base measure 2 Base measure 3 Base measure 4	Duration of build / Alias: Build speed Duration of unit test / Alias: Unit test speed Duration of component test / Alias: Component test speed Duration of function test / Alias: Function test speed
Measurement method	1. Export the commit data from Jenkins 2. Sort the events by time stamp 3. Find the time stamp for the start of build event (BD) 4. Find the time stamp for the start of the unit test event (UT) 5. Build speed is the difference between UT and BT (UT-BT) 6. Find the time stamp for the start of component test event (CT) 7. Unit test speed is the difference between CT and UT (CT-UT) 8. Find the time stamp for the start of function test (FT) 9. Function test speed is the difference between CT and FT (FT-CT) 10. Find the time stamp for the event after the function test (ET) 11. Component test speed is the difference between ET and FT (ET-FT)
Type of measurement method	Objective



(a) Company A (b) Company B

Figure 2. Box-plots for integration speed

speed and a selection of these factors. The third, and the last, is the model which is based on the workshop with Company A and Company B.

4.4.1. Initial list of factors

Initially, we reviewed the literature in the area of code review. We performed a search in ScienceDirect, IEEE Explorer and ACM digital library for these papers. The list of the identified factors is shown in Table 3.

Table 3. Factors identified as relevant in literature review

Factor name	**Ref.**	**Selected by Company**
Size (LOC)	[15]	x
Code quality after the review (defects/LOC)	[15]	
Software quality in test (defects / LOC) measured during the testing	[15]	
Initial code quality (defects/LOC) measured before code review (e.g. unit testing)	[15]	
Code review coverage (% of code that is reviewed)	[18]	
Code complexity (McCabe)	[18]	
Churn (added, removed, changed code)	[18]	
Change entropy (changes spread over many files)	[18]	

Number of previous patches	[17]	
Module location	[16]	x
Number of resubmits	[16]	
Number of comments per bug	[17]	
In-house (Ratio of internal contribution in the entire history of the current branch. Calculated in the same manner as Reviewed Commit)	[27]	
Code review rate (LOC/hour)	[15]	x
Task type (e.g. adaptive and corrective)	[3]	
Total number of authors	[18]	
Minor authors (contributed to less than 5% of the changes)	[18]	
Major authors (contributed to more than 5% of the changes)	[18]	
Author ownership (proportion of changes attributed to the author who made the largest % of changes)	[18]	
Number of developers	[3]	
Number of reviewers	[3]	x
Number of review tasks	[3]	
Number of reviews / task	[3]	
Number of review rounds	[3]	
Number of changed files	[3]	
Number of reviewed files	[17]	
Proportion of reviewed changes	[18]	
Proportion of reviewed churn	[18]	
Proportion of self-approved changes	[18]	
Proportion of hastily reviewed changes	[18]	
Proportion of changes without discussion	[18]	
Developer ability (defects/LOC) measured as average of previous assignments	[15]	
Number of writer's previous patches	[17]	
Review queue (number of pending reviews per reviewer)	[17]	
Reviewer/writer experience (number of completed reviews)	[17]	
Reviewer/writer experience for module (number of completed reviews for the module in question)	[17]	
Defect removal effectiveness of code review (review defects of all defects)	[15]	
Priority (priority of the found bug)	[17]	
Severity (severity of the found bug)	[17]	
Super review of core-code which usually is more defect-prone	[17]	

Number of developers in CC	[17]	
Number of commenting developers	[17]	
Average number of comments per developer	[17]	
Number of reviewer comments	[17]	x
Number of writer comments	[17]	
Number of iterations	[30]	
Discussion Length (same as review comments)	[30]	
Proportion of Revisions without Feedback	[30]	
Number of Non-Author Voters	[30]	
Proportion of Review Disagreement (A proportion of reviewers that vote for a disagreement to accept the patch, i.e. assigning a negative review score)	[30]	
Response Delay (Time in days from the first patch submission to the posting of the first reviewer message)	[30]	x
Average Review Rate (Average review rate (KLOC/Hour) for each revision)	[30]	

The factors in boldface were identified during the workshop with Company B as important to investigate further. Company B also complemented this list with two more factors, as they identified them to be relevant for Company B's context: (i) Acceptance of comment (number of accepts for comments) and (ii) Indication that a reviewer looked at the file when reviewing the patch (from Gerrit).

After the workshop, the results were sent to the reference group at Company A, who confirmed that these factors were indeed relevant for the comparison.

4.4.2. Correlation analysis

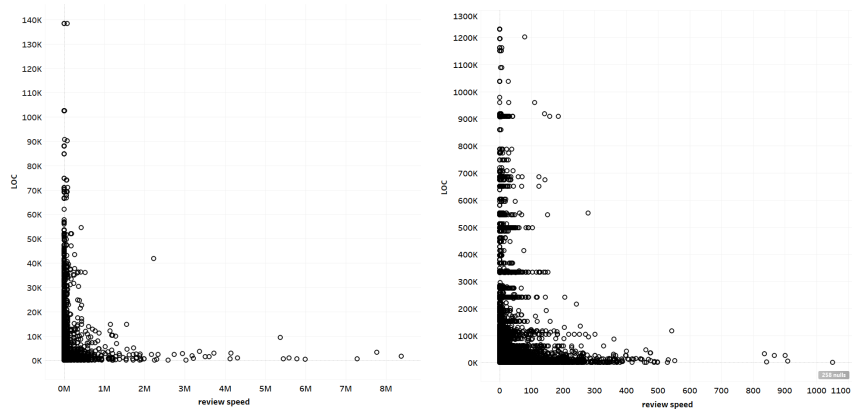
In the correlation analysis, the first step was the visual analysis using scatter plots. The Spearman correlation coefficient between these measures are presented in Table 4.

For Company A, the scatter plot for the relation between the review speed and the size of the modules is presented in Figure 4(a). For Company B, the scatter plot for the relation between the review speed and the size of the modules is presented in Figure 4(b).

Visually, we can observe that there is no correlation between the size of the module and the review speed. The same trend was observed for all other measured factors. As we can also observe, the trend is the same for both Company A and Company B.

However, an important difference between Company A and Company B is the number of reviewers assigned to review a module, which we present in Figure 4.

The numbers at Axis Y show that the number of reviewers assigned to review a module is much larger for Company B than for Company A. We conduct the workshop with the companies, in order to find what can contribute to the difference in the review



(a) Company A (b) Company B

Figure 3. Scatter plot of review speed vs. size of module (LOC)

Table 4. Correlations

Company A / Company B	LOC	number of reviewers	review speed
LOC	1.00/1.00	-0.01/0.01	-0.01/-0.02
number of reviewers	-0.01/0.01	1.00/1.00	0.22/0.10
review speed	-0.01/-0.02	0.22/0.10	1.00/1.00

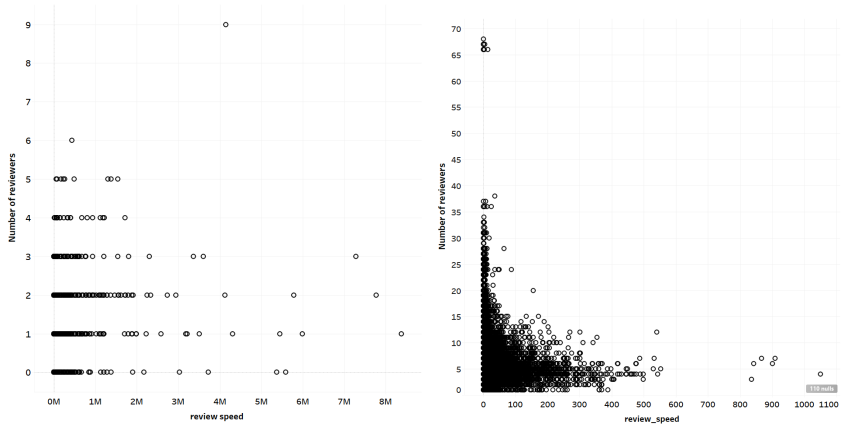
speed.

4.4.3. Factors affecting the review and integration speed – a model

After the workshop, we summarized the good practices in form of a contingency matrix, presented in Figure 5. The figure is organized into two dimensions – phase [review and integration] and speed [slow and fast].

We found that these two companies focus on two different activities and therefore one of the companies reviews faster while the other integrates faster. When focusing too much on the speed of reviews, Company A risked integration problems and slower integration. Whereas focusing on the review quality, Company B risked spending too much time for reviewing. In order to balance the speed of reviews with the speed of integration, the software development organization needs to find which practices contribute to either increasing or decreasing of the speed. Below, we summarize the practices found in our study.

Fast reviews are achieved when the team does the reviews within the team, as it is shown by Figure 4. Teammates can agree on the needed time for a review,



(a) Company A (b) Company B

Figure 4. Scatter plot of review speed vs. number of reviewers

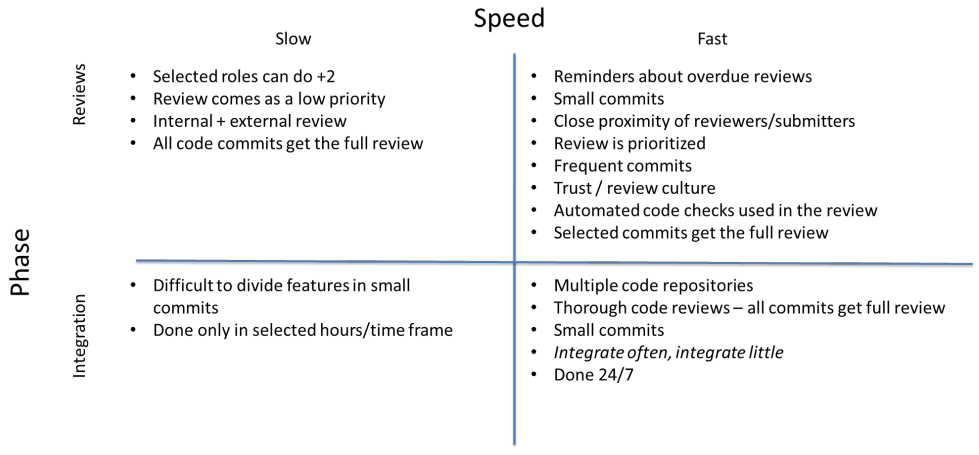


Figure 5. Factors affecting speed.

they can remind each other, and they have the joint responsibility for their product. It is also important to set reminders for the overdue reviews. In Company A, the reminders were set to two hours, which provided an implicit deadline; even if this was not a requirement to have the review ready within two hours, teammates recognized that the review is expected to be done in a matter of minutes, hours and not days or weeks. Having the reviews within one team leads to high quality review culture, increased trust and knowledge of the code being committed. However, in order to reach there, the team needs to prioritize the reviews – select which code commits have the full review and which ones get the quick one; the team also uses automated static code analysis checks to decrease the risk of missing obvious problems in a quick review. Finally, the team has identify the practice of "commit often, commit little" as important – they commit small code increments (e.g. 10-20 lines of code) and do that very often.

Fast integration is achieved when the company organizes their code base in a distributed manner. Company B has their code base distributed over more than 100 repositories. These repositories are integrated at different speeds and one repository's integration problems do not propagate to other repositories. Company B also makes a full review of all code commits and assures that the commits get external review; although it makes the review process slower, it increases the quality of the code and therefore increases the integration speed (a.k.a. fewer test case failures). The commits are also small (as for Company A and for fast reviews), but not as small as for Company A; Company B can have reviews of code of over 100 lines, spread over multiple files. However, their recipe for fast integration is "Integrate often, integrate little". During the workshop, both companies agreed that integration in a global way, with the ability to integrate and test round-the-clock (in different time zones) speeds up the development.

Slow reviews are often caused by being over-protective of the code. A development team might ask for an external review of the code of each commit, which leads to leaving the responsibility to a third party (Company B does it in another department). This means that the external party needs to put aside the time for the review and to be able to "switch context" to the code developed by the submitting department. Asking for all code commits to get the full review also decreases the speed. We have also observed that slow reviews can start a negative spiral, when the submitters wait for a larger part of the code to be finished before submitting, thus asking for more time from the reviewers.

Slow integration is often caused by the fact that the code is complex and inter-dependent. The interdependency can require more code to be committed, more to be reviewed, taking longer time. This leads to risks that code contains defects and triggers failures of tests. Having the integration during specific hours of the day also decreases the speed.

4.5. Evaluation of validity

In reporting the validity of our study, we follow the guidelines from Runeson et al. [26]. We discuss them per research question.

4.5.1. Construct validity

When addressing the first research question (What is the most relevant and practically usable definition and measure of speed), the main threat to validity is the process of conducting the literature review. Since we did not conduct a systematic review, we used snowballing to minimize the risk that the review was done without a rigour. We used the original paper by Fagan, 1976, to start the snowballing. Another threat to construct validity is the lack of review of literature of the factors that can theoretically influence the speed of integration. Since we have experienced professionals in our workshop and given the experience of the researchers in this particular field, we believe that this is not a major threat to the construct validity.

When addressing RQ2, we collected them from source systems, i.e. systems which are used for software development, not for reporting. Using these kind of measures reduce the researcher bias in data collection procedures; in particular, it reduces the probability of obtaining data points that are incorrect. We also used calendar time as the main measure, which is objective as the clocks of the source systems are computer clocks which are updated in real-time. One important aspect, however, is the fact that we only use the measures from finished reviews/integrations; this means that the reviews which are very long, never finished, are not part of our analysis. However, it was difficult to include the relevant ones (e.g. the ones that should be included because they should have been finished vs. the ones that are still in progress and are not expected to be finished yet).

When addressing RQ3 (Which factors are perceived by the studied organizations to affect the speed), we identified the threat that we missed important factors as we conducted a workshop, but not a survey. The participants of the workshop were expert integration managers, senior designers and architects, working with the process of code review. We see that their experience minimizes this risk.

4.6. Internal validity

For RQ1, we discussed the measures with the companies and therefore we perceive the risk of missing important measures as minimized.

For RQ2, we collected the data using scripts, which reduces the risk of errors in measurement. The scripts were tested before they were used.

For RQ3, we used workshops and close collaboration with industrial partners to validate the measures taken. In the workshops we also discussed whether we should expect certain causality relations in the data. When we examined the correlations visually, we noticed that there is no correlation, and therefore we did not report the

numbers.

4.7. External validity

For all research questions, we used two companies in our case study to increase the external validity. We believe that our observations at two case companies are an evidence that the results are not specific to one set-up, but they are more related to the way of working – Agile and Lean software development, continuous integration and deployment.

4.8. Reliability

For all research questions, since we reported our method and used standard tools to collect the data, we believe that other researchers could replicate the study with the same results. The company representatives in the workshops represented larger groups and they did not provide their own opinion, but stated facts that can be verified by others (e.g. number of repositories, size of commits).

5. Summary and Conclusions

In this section, let us summarize the main findings and recommendations from our study.

5.1. Summary

What is the most relevant and practically usable definition and measure of speed of reviews and integration in the studied companies? We have found that the definition of speed is a very good alias for the measurement of duration and is practically the most usable one. Since time is the only parameter in this measure, this measure of speed cannot be manipulated by changing the prerequisites. The objectiveness means that this measure forces companies to implement changes (improvements) in their ways-of-working in order to change (improve) their speed. The definitions of these measures are provided in Table 1 and Table 2. Although the measure of speed using time seems like common sense, our results show that in practice, as indicated by the companies, it has very valuable benefits.

Given the definition of the measure of speed, what is the speed of reviews and integration in continuous software development in modern companies? We have found that despite being described as continuous, the average speed of review and integration can be measured in days. The speed is determined by low priorities

of the review tasks or problems with quality during the integration. Our contribution here is the presentation of how fast modern organization are. Although we only provide the data from two companies, this kind of data comes from proprietary software development, i.e. not open source, voluntary based software development.

Which factors are perceived by the studied organizations to affect the speed of reviews and integration for the studied companies? The studied organizations have identified 19 factors which affect the speed of review and integration. These factors varied from organizational (e.g. only selected persons were allowed to approve the code reviews) and product-oriented (e.g. difficult to divide features in small commits). Although these factors were not measured, they were identified as different – one company was able to divide features in small commits and could integrate fast while the other did not and therefore integrated much slower.

We believe that the findings from our study can help companies to structure their code reviews and integration in such a way that they can achieve high speed without the need to decrease quality.

5.2. Conclusions

In this paper, we set off to study how manual and automated activities influence the speed of software development's review, integration and integration testing. Our results show that the measurement of speed can be done using calendar time rather than velocity, which contributes to the discussion of how to measure speed. The distribution of code repositories (and thus ability to divide large features into smaller commits), reminders and team proximity decrease the time needed to deploy the software.

Based on the discussions of which factors affect the speed and this measurement, we can conclude that when companies evolve in their agility, they realize that in order to be faster, they need to focus on developing their software products in shorter time. They realize that there is a limit to how much parallel development can be done and that no other means supersede their ability to shorten their development time. They realize that the shorter development cycles bring both obvious and hidden benefits.

Our future work is to expand the study by conducting another case study on importance of the identified factors on other measures (e.g. number of reviewers vs. quality of reviews).

References

[1] Alleman G. B., Henderson M., and Seggelke R. Making agile development work in a government contracting environment-measuring velocity with earned value. In *Agile Development Conference, 2003. ADC 2003. Proceedings of the*, pages 114–119. IEEE, 2003.

[2] Baum T., Liskin O., Niklas K., and Schneider K. Factors influencing code review processes in industry. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 85–96. ACM, 2016.

[3] Beller M., Bacchelli A., Zaidman A., and Juergens E. Modern code reviews in open-source projects: Which problems do they fix? In *Proceedings of the 11th working conference on mining software repositories*, pages 202–211. ACM, 2014.

[4] Blackburn J. D., Scudder G. D., and Van Wassenhove L. N. Improving speed and productivity of software development: a global survey of software developers. *IEEE Transactions on Software Engineering*, 22(12):875–885, 1996.

[5] Bosch J. *Continuous Software Engineering.* Springer, 2014.

[6] Bosch J. Speed, data, and ecosystems: The future of software engineering. *IEEE Software*, 33(1):82–88, 2016.

[7] Choi J. The science behind why jeff bezos's two-pizza team rule works, 2014.

[8] Coelho E. and Basu A. Effort estimation in agile software development using story points. *International Journal of Applied Information Systems (IJAIS)*, 3(7), 2012.

[9] Cohn M. *User stories applied: For agile software development.* Addison-Wesley Professional, 2004.

[10] Cohn M. *Agile estimating and planning.* Pearson Education, 2005.

[11] Coskun H. Cognitive stimulation with convergent and divergent thinking exercises in brainwriting: Incubation, sequence priming, and group context. *Small group research*, 36(4):466–498, 2005.

[12] Fagan M. Design and code inspections to reduce errors in program development, IBM Systems Journal, vol. 15, 1976.

[13] Hüttermann M. *DevOps for developers.* Apress, 2012.

[14] Jaspan C., Jorde M., Knight A., Sadowski C., Smith E. K., Winter C., and Murphy-Hill E. Advantages and disadvantages of a monolithic repository: a case study at Google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 225–234. ACM, 2018.

[15] Kemerer C. F. and Paulk M. C. The impact of design and code reviews on software quality: An empirical study based on psp data. *IEEE transactions on software engineering*, 35(4):534–550, 2009.

[16] Kononenko O., Baysal O., and Godfrey M. W. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering*, pages 1028–1038. ACM, 2016.

[17] Kononenko O., Baysal O., Guerrouj L., Cao Y., and Godfrey M. W. Investigating code review quality: Do people and participation matter? In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 111–120. IEEE, 2015.

[18] McIntosh S., Kamei Y., Adams B., and Hassan A. E. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201. ACM, 2014.

[19] Meding W. Effective monitoring of progress of agile software development teams in modern software companies: An industrial case study. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, IWSM Mensura '17, pages 23–32, New York, NY, USA, 2017. ACM.

[20] Nicolette D. *Software development metrics*. Manning, 2015.

[21] Organization I. S. and Commission I. E. Software and systems engineering, software measurement process. Technical report, ISO/IEC, 2007.

[22] Perry D. E., Porter A., Wade M. W., Votta L. G., and Perpich J. Reducing inspection interval in large-scale software development. *IEEE Transactions on Software Engineering*, (7):695–705, 2002.

[23] Petersen K. A palette of lean indicators to detect waste in software maintenance: A case study. In *Agile processes in software engineering and extreme programming*, pages 108–122. Springer, 2012.

[24] Porter A., Siy H., Mockus A., and Votta L. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(1):41–79, 1998.

[25] Runeson P. and Höst M. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131, 2009.

[26] Runeson P., Host M., Rainer A., and Regnell B. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[27] Shimagaki J., Kamei Y., McIntosh S., Hassan A. E., and Ubayashi N. A study of the quality-impacting practices of modern code review at sony mobile. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 212–221. ACM, 2016.

[28] Staron M. and Meding W. Monitoring bottlenecks in agile and lean software development projects–a method and its industrial use. *Product-Focused Software Process Improvement*, pages 3–16, 2011.

[29] Staron M. and Meding W. *Software Development Measurement Programs: Development, Management and Evolution*. Springer, 2018.

[30] Thongtanunam P., McIntosh S., Hassan A. E., and Iida H. Investigating code review practices in defective files: An empirical study of the qt system. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 168–179. IEEE Press, 2015.

[31] Thongtanunam P., McIntosh S., Hassan A. E., and Iida H. Review participation in modern code review. *Empirical Software Engineering*, 22(2):768–817, 2017.

Received 19.06.2018, Accepted 19.11.2018