Understanding sprint velocity fluctuations for improved project plans with Scrum: a case study

Filipe Albero Pomar[1,*,†], Jose A. Calvo-Manzano[1], Edgar Caballero[1] and Magdalena Arcilla-Cobián[2]

[1]*Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Madrid, Spain*
[2]*Escuela Técnica Superior de Ingeniería Informática, Universidad Nacional de Educación a Distancia, Madrid, Spain*

ABSTRACT

Starting from the documentation of high sprint velocity fluctuations in a Scrum project, this paper presents a thorough approach to identify the sources of issues arising in the context of Scrum implementation. Given that Scrum provides guidance on identifying process issues but not their root causes, various approaches are explored. This is of great relevance because Scrum defines project schedules relying heavily on sprint velocity and because it is the most widely used agile methodology. The findings provide a new approach to evaluate such fluctuations and establish a more realistic project assessment than what is currently defined by Scrum. In this respect, this paper contributes to improve the understanding of the software development process using this agile framework.

1. INTRODUCTION

In today's competitive and fast-paced world, companies are under pressure to adapt to an ever changing environment [1]. The software development industry is not an exception. What started as a technique to develop industrial products faster has now become part of the IT sector under the name of Agile Software Development [2, 3]. Among many methodologies that promise increased agility, Scrum [4] is the most widely used globally [3].

Scrum has been successfully used in a wide variety of industries to create software projects [5]. As opposed to traditional development models, such as waterfall, it defines the creation and delivery of software in small increments with little upfront planning. Hence, it allows companies to deliver value to their clients earlier and to adapt to changes faster. Although it exploits the benefits of agile methodologies, it does present a key drawback. In fact, Scrum does not provide detailed guidance on how to identify the root cause of process issues when they occur [4, 6].

This paper provides a case study on the implementation of Scrum in a software development project of a non-governmental organization. Analysing the role of Scrum within the project, it investigates the root cause of high fluctuations in the amount of work done, known as the velocity, in each short development cycle, called sprint. This issue is of paramount importance because project scheduling is derived from the aforementioned metric [7, 8]. In this respect, this paper contributes to improve the understanding of the development process within this agile framework.

*Correspondence to: Filipe Albero Pomar, Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Madrid, Spain.
†E-mail: filipe.albero.pomar@alumnos.upm.es

The paper is structured as follows. Section 2 characterizes the software development settings where high sprint velocity fluctuations are observed. Section 3 provides a structured approach for evaluating the possible forces at play causing the observed fluctuations. Finally, Section 4 presents conclusions.

2. CONTEXT

2.1. The organization

THE COMPANY (fictitious name for confidentiality reason) is a small non-governmental organization based in the UK with a staff of 44 employees. One of the services it provides to the local community is a volunteering web-based search engine. Since it was first released in the year 2000, it has become the leading website for finding volunteering opportunities in the country. In 2012, the search engine had over 1 million registered users and a monthly average of 186K unique visitors and 3.2 million page views. The first version of the software was not designed to cope with this traffic; response times were slow, new functionalities were needed, but the old codebase was difficult to adapt. To solve these problems, a complete rewrite of the system was commissioned.

The new version of the website was written in-house and passed onto a newly formed team of developers to deliver it to production. The new team was formed in April 2011 and given full responsibility for bug fixing and creation of new minor features. In December of that year, the system went live, completely replacing the legacy codebase. For the following 6 months, the team focused on bug fixing and creating new administrative subsystems not present in the original software.

2.2. Scrum introduction

Scrum is an empirical process control model founded on three pillars: transparency, inspection and adaptation [6, 9]. It states that software must be created in small increments that deliver business value to the customer. Instead of big upfront designs and exhaustive bureaucratic plans, it fosters the creation of a slim project plan that is revised and augmented as the project develops. Each software increment is created in a time-boxed period called sprint that usually lasts 2 to 4 weeks [10].

The steps for developing software with Scrum start with the creation of a prioritized list of requirements called product backlog [5]. Just before each sprint, the team gathers in a sprint planning meeting in which it estimates requirements from the product backlog and decides what can be implemented. Throughout the sprint, the daily Scrum meeting is carried out to identify issues and to communicate what each individual is working on. High visibility tools (i.e. sprint backlog, sprint burndown) are used to communicate progress to all parties. To close the short development cycle, the team carries out two meetings: sprint review, where developers showcase work that has just been completed, and a sprint retrospective, to identify process improvements. The final outcome of the sprint is a potentially shippable software increment [11].

Scrum identifies three main roles that work together daily throughout the project [4]. The product owner represents the business, provides a project vision, and generates requirements and their priorities. The Scrum master, often a developer, ensures the correct implementation of Scrum. Lastly, the development team is composed by a cross-functional group of software developers.

2.3. Scrum implementation

The project implemented Scrum as prescribed. All team members had prior experience with agile methodologies, but only one knew Scrum in depth and had used it professionally. The framework was taught by the most experienced developer and achieved high buy-in from both developers and managers. The project data set available refers to the usage of Scrum from June 2011 to June 2012.

Throughout the 20 sprints analyzed in this paper, the team composition remains mostly unchanged. There are three senior developers, one front-end specialist, and at the 20th iteration, a junior developer is added to the team. With the exception of the front-end specialist, all members develop all application layers and create tests for each functionality. As defined by Schwaber and Beedle [9], the development

team and the product owner are collocated and work together daily throughout the project. Requirements are written in the form of user stories [12]. Estimations are done using a consensus-based technique called planning poker [11] to quantify size (in 'points') of either new features or change requests. At the end of each 2-week sprints, all sizes of work items done are added together to what is known as velocity, a widely used agile metric [3]. To the team, its definition of done [4] means that the functionality is coded, verified by automated tests, approved by the product owner and stored under version control.

2.4. The project

The project comprises one main application and eight web-based subsystems that support its operation. When considered as a whole, the codebase sizes up to over 50K lines of code, which is equivalent to a mid-sized system.

Scrum identifies some high visibility tools to communicate progress from the development team to the rest of the organization [7, 8]. Throughout the project, both the sprint backlog and the sprint burndown are maintained. The sprint backlog is a whiteboard where requirements are placed at the leftmost part of the board and are physically moved to the right to indicate the progress towards completion. The aforementioned whiteboard is divided into four sections, starting from the far left: not started, started, signoff pending, and done. Next to the sprint backlog, a sprint burndown chart shows the amount of work still left to be done in the current sprint. The team works from one sprint to another without planning a wider horizon of a release planning. In fact, this business decision complies with Scrum as the sprint planning meeting is a mandatory activity whereas release planning is not [4]. For this project, the lack of release planning is never a real issue, as new functionalities are regularly delivered and deadlines met.

Nevertheless, going forward with the project, it becomes evident that better scheduling allows the creation of more reliable plans and setting stakeholders' expectations. In order to predict what can be accomplished, Scrum identifies historical sprint velocity as the most reliable forecast of future outcomes [11]. In that direction, there are two techniques for predicting future performance, and both are based on past accrued velocities (which is the amount of work completed per sprint). First, the velocity of each sprint can be plotted to identify any trends (i.e. a downward trend could indicate problems [11]). Second, it is possible to calculate a confidence interval to understand the probability of future velocities and employ it in the creation of a reliable project schedule [8].

Figure 1 shows the development team's velocity per sprint. In the first three sprints, the team is getting used to the planning poker and Scrum; therefore, over optimism leads to somewhat inflated estimations. Turning to the analysis of abnormal sprints, sprint 1 marks the use of Scrum. In this case, the team is already acquainted with the codebase through *ad hoc* tasks given to them. Sprints 15, 18 and 19 have zero velocity as a result of work on a new functionality based on new technologies that turn out to be problematic for production use. During sprint 16, the team is engaged in the recruitment of a replacement for a senior developer. Finally, a previous employee familiar with the system is contracted.

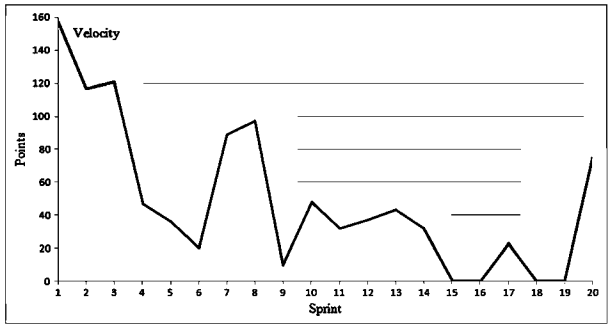

Figure 1. Historical sprint velocity.

The descriptive statistics of the historical velocities show a median velocity of 36.5, standard deviation of 45.7 and mean of 49.2 points. Furthermore, there is 90% likelihood that in future sprints, the actual accrued velocity will fall between 20 and 75 points. The plot of the historical sprint velocity presents a strong downward trend that could signal problems. These prevent the creation of a reliable schedule for the project and, in fact, suggest that there is a process issue in play [11].

The team follows Scrum as prescribed and uses the tools it provides to gauge the project's progress. However, the preliminary analysis signals a potential problem related to the implementation of the methodology. In this respect, the Scrum framework does not describe a procedure to identify the root cause of the problem. Based on the analysis of the project data, the next sections give a description of the troubleshooting efforts carried out to uncover the causes of the velocity fluctuations. This allows deriving some general considerations on problem detection, which may lead to the creation of better project schedules in Scrum.

3. SPRINT VELOCITY FLUCTUATION STUDY

The absence of clear guidelines on identifying root-causes of sprint velocity fluctuations compels Scrum practitioners to employ exploratory troubleshooting efforts at their discretion. This section describes a structured approach for evaluating the possible forces at play causing the observed fluctuations.

3.1. Commitments are not fulfilled

The fulfillment of commitments by the team solidifies trust between developers and business people [6]. As suggested by Cohn [11], the development team decides what can be accomplished in a sprint following the consensus-driven approach. According to this technique, the team takes into consideration their availability and task complexities to decide what they can commit to. Even though the development team has free choice on deciding what they can commit to, they consistently fail to fulfill those commitments.

In order to gauge the amount of commitments not fulfilled, it is possible to resort to a traditional (non-agile) project management metric called Schedule Performance Indicator (SPI) [8]. The SPI is calculated as the ratio of earned value on planned value. In the case of Scrum, it can be described as the total points completed at the end of a sprint over the total points the team committed to in the sprint planning meeting. The descriptive statistics of the SPI series present a median of 64.17% of commitments fulfilled and 90% likelihood that commitments will be honoured by 37 to 83% in a future sprint. When evaluating SPI, a useful complementary metric is team availability (TA), which is the percentage of developers available in the sprint. Figure 2 shows the two aforementioned metrics in use. On sprint 1, the team delivers 82% of the work it committed to while having 92% of the team present for the entire sprint. It is important to note that either metric could potentially go beyond 100%, even if neither case ever occurred.

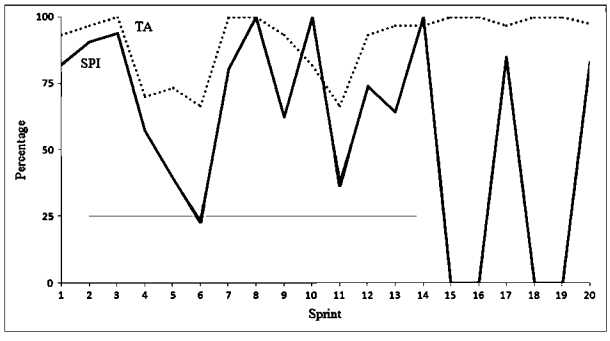

Figure 2. Schedule Performance Indicator (SPI) and team availability (TA).

The graphical analysis shows that commitments are only met three times out of 20 iterations. It could be argued that the development team does not finish its work because its focus is deviated from the sprint into other tasks that either have no assigned points or are not value-adding activities (i.e. unscheduled meetings). Although these scenarios happen sporadically, they are not the norm; time for meetings is always taken in consideration when committing to work during the sprint planning meeting. For these reasons, SPI alone does not explain the high sprint velocity fluctuations but provides input for further investigation.

3.2. Correlation of team availability and commitment fulfilment

While unaccounted leave (i.e. because of sickness) may lead to the systemic missing of commitments, SPI and TA present a correlation coefficient of 0.12. In this respect, availability does not provide a comprehensive explanation of the team poor commitment fulfilment.

3.3. High work in progress (WIP)

WIP may be an important factor in the analysis. Features that are too big to be finished in an iteration can lead to high WIP. High WIP can cause at least two big problems [8, 11]. First, it leads to high context switching that is known to decrease developers' performance. Second, it means that many tasks are started but not completed at the end of the sprint. By not meeting the team's definition of done, those tasks are not added to the sprint's velocity [11, 13]. To curb WIP, each developer avoids working in more than one story at the time.

In addition, larger tasks (those longer than a day's work) can slip from one sprint to another, which results in high WIP and low delivery. To tackle this problem, larger tasks are broken down into smaller ones that can be completed in one work day and only some uncommon tasks are allowed to be at most 3 days. Furthermore, to create good quality estimations, the development team follows [11] and ensures that estimations are made relative to each other and created against a baseline of sample user stories. Hence, WIP is kept to a minimum and is not the culprit of high sprint velocity fluctuations.

3.4. Team dynamics and rework

Communication between developers and product owner is frequent and honest. This and the clear definition of done leads to small amount of rework and allows the team to solve any doubts during estimation sessions; in other words, the team is confident that they know all that is needed when estimating. Team coherence is never affected by individual's different cultural backgrounds, as warns Cohn [7], because of candid face to face communication. Although there could be politics in play pushing the team to over-commit, that is never the case.

3.5. Hidden complexity

Ken Schwaber noted in his books that complexity in software projects is influenced by requirements, technology and people [6, 9]. When these factors interact, complexity rises and project control becomes increasingly challenging. In order to acknowledge complexity, projects can be categorized as simple, complicated, complex, or chaos depending on technology certainty and requirements' agreement. The rationale behind these categories helps identify the software development process that better adapts to each project. Simple projects can be controlled through any methodology, including waterfall; complicated or complex projects benefit most from empirical processes such as agile; chaos projects are highly unstable and cannot be properly controlled. Figure 3 [14] (original from [15] and presented in the context of software development by [9]) presents a visual representation of project complexities' interrelations.

The project analyzed in this paper can be classified as complex when taken as a whole because not all technologies involved are mastered by developers, and some requirements are far from certain. If instead each subsystem is considered as a different project, different complexity classifications emerge. There is one simple project, three complicated projects, and five complex projects. By aggregating subsystems in the aforementioned categories, it is possible to calculate confidence intervals at 90% for their SPIs. The picture that forms is as follows (some activities are included in

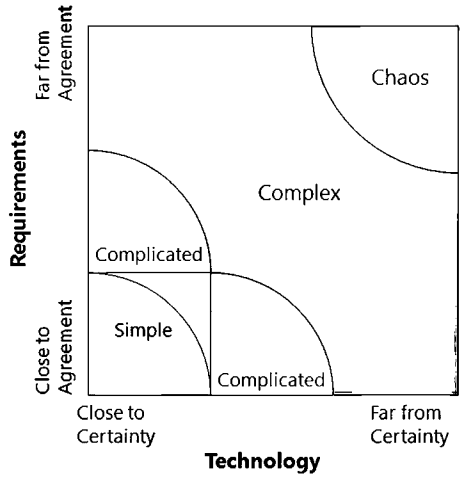

Figure 3. Project complexity.

the whole sample but excluded from the detailed project samples as a result of their specific nature that does not allow an objective association to a specific subsystem):

- Simple: the team will meet their commitments by 81 to 100%
- Complicated: the team will meet their commitments by 74 to 100%
- Complex: the team will meet their commitments by 20 to 100%

As it can be seen from the confidence intervals, the complexity of the modules directly influences how the development team meets their commitments. Simple subsystems contain much more stable requirements and defined set of technologies, which allows for greater predictability of work. Whereas complicated or complex subsystems present both technological and requirement novelties that surface after commitments are made and code is being developed.

This consideration allows explaining the root cause for the high sprint velocity fluctuations on a cause–effect basis. Each project's subsystem has different levels of complexity with respect to requirements and technologies. The development team does not account for these important differences in the sprint planning meeting because it follows Scrum as prescribed that is creating estimations for the project as a single unit. Commitments are made but are often missed because requirements change and some technologies used are new to the development team. Hence, the accrued sprint velocities present great variations over time. This is of relevance for both the specific project and the agile community because Scrum uses a single measurement of historical sprint velocity to derive a project schedule.

Generally, the interpretation of the results of Scrum implementation greatly benefit from a more thorough analysis than that prescribed by the framework itself. In this sense, by detecting the specific impact of complexity in the components of one project, we managed to clarify the root cause of what is observed to be large sprint velocity fluctuations.

4. CONCLUSIONS

This paper investigates possible reasons for high sprint velocity fluctuations in a Scrum project. Given that Scrum provides guidance on identifying process issues but not their root causes, different approaches are explored. It is shown that sprint velocity fluctuations are caused by the team missing their commitments that, in turn, depends on the unaccounted complexities of different subsystems. This distortion occurs because Scrum is implemented as prescribed, that is, considering sprint velocity for the project as a single unit. Future research could formalize a Scrum extension to account for projects that have heterogeneous levels of requirements' agreement and technologies' certainty.

These findings can aid Scrum practitioners in creating more reliable schedules based on historical data. The use of historical sprint velocity along with the acknowledgement of the project's SPI and confidence intervals for different system's modules provide a more realistic project analysis than

what currently is defined in the Scrum framework. In this respect, this paper contributes to improve the understanding of the software development process using this agile framework.

REFERENCES

1. Kotter JP. Accelerate! *Harvard Business Review*, 2012; November.
2. Takeuchi H, Nonaka I. The New Development Game. *Harvard Business Review*, 1986; January.
3. State of Agile Survey. VersionOne, 2011. http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf [28 March 2014].
4. Schwaber K, Sutherland J. http://www.scrum.org/Scrum-Guides [28 March 2014].
5. Pichler R. Agile Product Management with Scrum. Addison-Wesley: Boston, 2010.
6. Schwaber K. Agile Project Management With Scrum. Microsoft Press: Redmond, 2004.
7. Cohn M. Succeeding with Agile Software Development Using Scrum. Addison-Wesley: Boston, 2009.
8. Griffiths M. PMI-ACP Exam Prep. RMC Publications: Minnesota, 2012.
9. Schwaber K, Beedle M. Agile Software Development with Scrum. Prentice Hall: Upper Saddle River, 2002.
10. Kenneth S. Essential Scrum. Addison-Wesley: Boston, 2012.
11. Cohn M. Agile Estimating and Planning. Prentice Hall: Upper Saddle River, 2006.
12. Cohn M. User Stories Applied. Addison-Wesley: Boston, 2004.
13. Kniberg H. Scrum and XP from the Trenches. C4Media: United States of America, 2007.
14. Malik N. All Effective Enterprise Architects are Agile. http://blogs.msdn.com/b/nickmalik/archive/2013/01/15/all-effective-enterprise-architects-are-agile.aspx [28 March 2014].
15. Stacey RD. Strategic Management and Oganisational Dynamics: The Challenge of Complexity. Prentice Hall: Harlow, 1999.