Collocated Twenty-Hour Week

Yu Chin Cheng[1], Chin-Yun Hsieh[1], and Kai H. Chang[2]

[1]Department of Computer Science and Information Engineering, Taipei Tech, Taiwan

[2]Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama

{yccheng, hsieh}@csie.ntut.edu.tw, changka@auburn.edu

Abstract. *This paper presents a pattern for allocating and organizing work hours for Scrum team consisting of fully-time graduate students pursuing Master's degree in computer science. Degree requirements require Scrum team members to take a substantial number of courses which could lead to schedule conflicts that prevent the Scrum team from collocating. The solution is to coordinate electing courses so that common blocks of at least twenty hours are allocated for collocated work and Scrum meetings. The solution has been applied in several long running software development projects in Software System Lab of Taipei Tech.*

Categories and Subject Descriptors

•Software and its engineering → Agile software development

•Software and its engineering → Programming teams

•Software and its engineering → Pair programming

•Software and its engineering → Maintaining software

•Software and its engineering → Open source model

General Terms

Software development, student developers

Keywords

Scrum, collocation, scheduling, focus factor

1. Introduction

This paper shares our experience in running Scrum development projects where developers are graduate students pursuing Master's degree in computer science. As educators of software engineering, we take the position that students learn the most when they work on software that is actually used. In so doing, developers touch on all phases of development in requirements, design, coding, testing and maintenance by fixing bugs, adding new features, and so on. The question "*How can we make sure the development proceed as desired?*" is not easy to answer. In fact, we have experienced projects that failed or were abandoned after the principal developers left the team. After experimenting for a few years in the early 2000's, we settled with Scrum because it was the only method we tried that seemed to be working. Thus, in the past few years, we created a number of long running Scrum projects that have continuously released new products to users. Some of these projects are open-source, such as ezScrum [4], which have found some academic and industrial users that use it as a tool support for their Scrum projects. (It is interesting to see that the top five countries of download are Taiwan, China, the US, Brazil, and France [7].) Other projects develop tool support for our research work, including PS4Mobile [3] for modeling requirements for mobile applications based on the concept of pseudo software [2] and Robusta for detecting code smells in exception handling code in Java [5]. Both of them are mainly used by people in our own group.

Many proven patterns are available from the ScrumPLoP website [8]. This paper discusses a somewhat peculiar issue of allocating and organizing work hours for Scrum team consisting of graduate students. Popularized by the XP practice of *sustainable pace* in development [1], the forty-hour week is an accepted practiced in agile development. On the one hand, it is not possible for members to dedicate the forty-hour week for Scrum in the present case; individual members need to spend time to take courses and satisfy thesis requirements. On the other hand, if the time allowed for development is too scant, e.g., ten hours or less, time spent on various meetings of Scrum become an overburden and little time is left for development. Furthermore, having enough time for development by individuals is necessary but not sufficient in forming a Scrum team. Even if each developer sets aside twenty hours for development, it could still break a Scrum team when individuals' curricular work leaves little time for the whole team to be together. Symptoms of this happening include absences of members in various Scrum meetings, no pair programming between members, and so on. Gradually, the Scrum team becomes dysfunctional.

This paper documents a pattern of our experience in allocating and organizing work hours for a Scrum team consisting of graduate students pursuing Master's degree in computer science.

2. A collocated twenty-hour week

Problem

A Scrum team works best if team members are collocated and consistently work in the same hours. In a Scrum team where members are graduate students, some members could be away from the team at different times due to schedule conflicts created by individuals electing different courses. This creates situations where fundamental events of Scrum are difficult to become stable: daily scrum not happening regularly; people skipping sprint planning; members rarely pairing up for tasks; and so on. ***How can we keep the events more stable?***

Forces

- Taking courses is part of degree requirements for graduate students. In the present case, Master's degree students are required to complete 24 credit hours from eight 3 credit-hour courses. Some of the courses require a term project that involves building software.

- Team members are students who come from different backgrounds and different technical interests [9]. If uncoordinated, they tend to elect different courses that lead to conflicting schedules.
- Software under development by the Scrum team requires members to have some technical skills, e.g., coding, unit testing, design patterns, acceptance testing, and so on. Members need to learn the skills if they have not already known them.
- The students should be encouraged to take courses outside of their core competency to increase diversity.

Solution

Secure at least twenty hours in a week during which time all team members are collocated. Remind the team that the focus factor is 20 hrs/40 hrs or 50% and that this is lower than the recommended 70% for new teams [6] to make room for their coursework. Consolidate individual schedules by having the team members discuss which courses to take together. Encourage them to identify a number of courses which could help strengthen members' technical skills. For first-year students, the core courses could be recommended by the second-year team members and the advisor based on the required software development skills and domain knowledge. Allow people who couldn't fit into the collocated 20-hours week to seek out other development teams to join.

From the 20-hour schedule, give priority to securing a fixed slot for the 15-minute daily standups. Make sure that a long enough stretch of time (at least two hours) is allocated to sprint planning, and to have at least one hour each for sprint review and sprint retrospective, respectively. Have the team agree not to undertake coursework during the twenty hours as much as possible. Occasionally, when it happens that a particular (usually first-year) member is entirely occupied by coursework, encourage him/her not to check out any task from the sprint backlog. This gives more freedom to other members on checking out tasks and prevents a task from being locked out by an occupied member. One the other hand, require the coursework-occupied members to join all other Scrum meetings to ensure that they know what's going on.

Consequences

- The team has at least twenty hours for collocated meetings and development.
- Course elections are better aligned with the required development skills and domain knowledge.
- Individual members may not be able to elect courses according to their own wills.
- Coursework may still devour into the twenty-hour blocks.
- First-year graduate students may not feel that they have accomplished much in the first semester.

Pattern in action

Collocated twenty-hour week has been practiced in a number of software development projects running Scrum, at the Software Systems Lab, Taipei Tech, including PS4Mobile [3], ezScrum [4], and Robusta [5]. We describe how it is done in the fall semester of 2015 in the case of PS4Mobile.

PS4Mobile is a requirement development tool for mobile applications. Based on the concept of pseudo software [2], a requirement artifact created using PS4Mobile has not just the look and feel of a real mobile application, but also brings operability to requirement artifact concretely as a way of mediation, thereby closes the gap of communication between development and requirement. Specification of requirements in PS4Mobile is end-to-end, and

test scripts can be generated for automatic execution. Since August 2015, the PS4Mobile Scrum team consists of three second-year graduate students and three first-year graduate students. Two of the second-year members have been with the team for one year, while the other four newly joined the team. The newly joined second-year student previously worked on the development of a mobile testing platform, and was regarded as technically competent for the development tasks. The three first-year students were fresh out of college. Even though they were not as technically prepared, they showed good motivation for the development work. The product owner was a PhD degree candidate who works on test automation. The team uses a sprint length of two weeks.

Table 1. Courses elected by the team members.

member / course	A(2)	B(2)	C(2)	D(1)	E(1)	F(1)
SRS	x	x	x	x		
POSD				x	x	x
SWE					x	x
ALSD				x	x	x
Seminars				x	x	x
Total hours	3	3	3	11	11	11

M(i): Member M in his/her i-th year of graduate program, where M = A, B, …, F and i = 1 or 2.

SRS: Software Requirements and Specifications

POSD: Pattern-Oriented Software Design

SWE: Software Engineering

ALSD: Agile and Lean Software Development

Before the fall semester began, the team got together to discuss which courses to take. Table 1 lists the courses elected by the team. The courses elected by members are highly coherent and are aligned for acquiring the skills required by the development work. For example, the course POSD covers design patterns, unit testing, and architecture design. All second-year students elect software requirements and specifications since the knowledge could be useful for the development of PS4Mobile, a requirement modeling tool.

The consolidated course elections make it possible to have continuous slots for running Scrum on every weekday. Table 2 shows the collocated twenty-hour week. Daily standup takes place at 4:45 pm just before the work day closes. When members come to work the next day, they already know the day's development assignments, which allow them to proceed even without all members present. This is particular helpful to the second-year members since they have a much lighter coursework. The bi-weekly sprint planning takes place right after sprint review and retrospective, intervened by a lunch break to avoid over-stretched meetings.

Table 2. The twenty-hour week (shaded) with Scrum ceremonies in red.

hour \ day	Mon	Tue	Wed	Thu	Fri
9	SRS	POSD	SWE	ALSD	
10	SRS	POSD	SWE	ALSD	Demo(1h)
11	SRS	POSD		ALSD	Retro(1h)
13					Planning(1h)
14					Planning(1h)
15				Seminar	
16		SWE		Seminar	
17	Daily(15m)	Daily(15m)	Daily(15m)	Daily(15m)	Daily(15m)

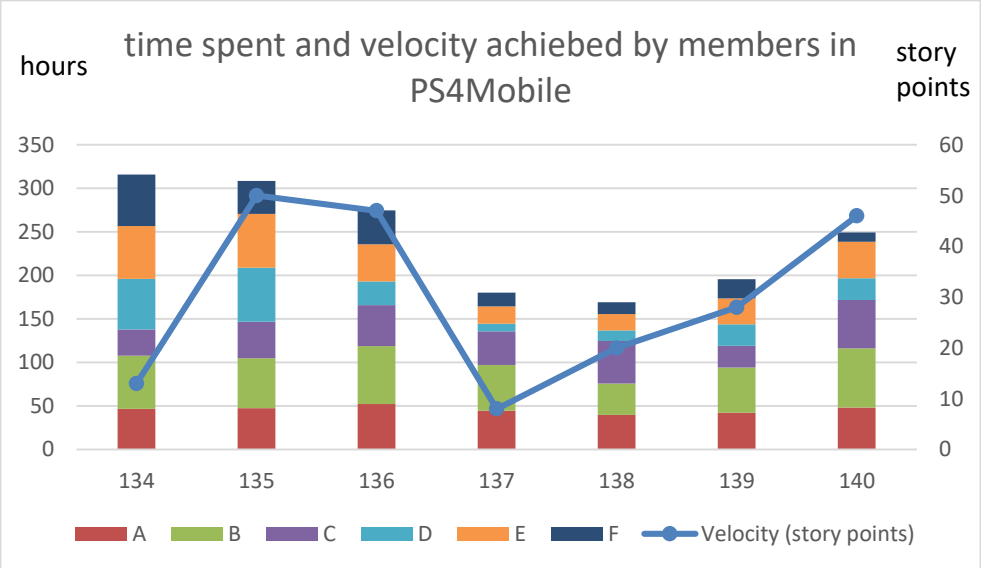

Figure 1. Actual time spent and velocities achieved by members of PS4Mobile from sprint 135 to sprint 140. Fall semester began during sprint 137.

Figure 1 shows the time spent by members on collocated Scrum development and the achieved velocities from sprint 134 to sprint 140. Note that the time spent by members was not part of the Scrum requirements but rather a common practice of the Software System Labs. Each week, there is a brief review of how individual members spent time during the previous week. In so doing, it has been emphatically declared by the Lab co-directors that the purpose of time recording is to help the members of the lab understand how they spend time; no reward or penalty is ever associated with the recorded amount of time. Under this context, team members are able to provide their actual time spent on the project.

In Figure 1, it is interesting to note that members spent more than twenty hours per week during the summer sprints (sprints 134, 135, and 136). Once fall semester began, the second-year students A, B, and C were able to keep to the twenty-hour week. In contrast, the first-year students D, E and F were visibly unable to keep up due to coursework.

Figure 1 also shows the velocity achieved by individual sprints. Recall that all first-year members and one second-year student joined at the beginning of sprint 134. Their initial contribution to the project was relatively limited as they were still getting familiar with PS4Mobile. Interestingly, although velocity plunged in sprint 137 when fall semester began, it kept increasing steadily and reached the 60% level of the summer sprints in sprint 139. Note that sprint 140 covers three weeks to account for the midterm exam. However, since the second-year students had only one midterm exam, they were able to contribute more time. As a result, both the time spent on project and the velocity reached the level of summer. This probably suggests that the second-year students are still the main contributors of the project. Reviewing historical data from the previous year for the second-year students, A and B, we have found a similar pattern. As first-year students, their work hour contributions were at about the same level as the present first-year students (D, E, and F).

3. Conclusion

We have presented a pattern for allocating and organizing work hours for a Scrum team consisting of graduate students in our Master's degree program. The pattern has been applied to a number of long running projects of the Software Systems Lab at Taipei Tech. All projects have been able to release new versions of the product continuously. As a side note, although not directly studied in this paper, a large proportion of students of the Software Systems Lab continued to practice agile development in industry, which probably suggests their experiences gained in the Scrum are helpful.

Acknowledgements

We would like to thank our shepherd Yasunobu Kawaguchi and the writer's workshop participants Morris Chuang, Helena Shih, Paulina Silva, and Allen Wirfs-Brock for helping us to improve this pattern. This research is supported in part by grants from Ministry of Science and Technology, Taiwan, under contracts 104-2221-E-027-010 (Y. C. Cheng) and 104-2221-E-027-007 (C.-Y. Hsieh).

References

[1] Beck, Kent. Extreme programming explained: embrace change. Addison-Wesley Professional, 2000.

[2] Jwo, Jung-Sing and Yu Chin Cheng. "Pseudo software: A mediating instrument for modeling software requirements." Journal of Systems and Software 83.4 (2010): 599-608.

[3] PS4Mobile. Available from http://ps4mobile.blogspot.tw/. Accessed Jan 4, 2016.

[4] ezScrum: tool support for Scrum. Available from http://sourceforge.net/projects/ezscrum/files/Release/. Accessed Jan 4, 2016.

[5] Robusta – exception handling smell detection tool. Available from Eclipse marketplace, http://marketplace.eclipse.org/ and Github, http://github.com/. Accessed Jan 4, 2016.

[6] Kniberg, Henrik. Scrum and XP from the Trenches: How we do Scrum. Lulu.com, 2007.

[7] ezScrum download statistics. Available from http://sourceforge.net/projects/ezscrum/files/Release/stats/map?dates=2010-03-10%20to%202016-01-04. Accessed Jan 4, 2016.

[8] Scrum Patterns. Available from http://www.scrumplop.org/. Accessed Jan 4, 2016.

[9] “Unity of Purpose” pattern in Organizational Patterns. Available from http://orgpatterns.wikispaces.com/UnityOfPurpose. Accessed Jan 4, 2016.