Moonlighting Scrum: An Agile Method for Distributed Teams with Part-Time Developers Working during Non-Overlapping Hours

Philipp Diebold, Constanza Lampasona
Fraunhofer Institute for Experimental Software Engineering
Kaiserslautern, Germany
{philipp.diebold, constanza.lampasona}@iese.fraunhofer.de

Davide Taibi
Software Engineering Research Group
University of Kaiserslautern
Kaiserslautern, Germany
taibi@cs.uni-kl.de

***Abstract*—Scrum and several agile development processes are becoming increasingly popular since they offer the ability to manage volatile requirements. This applies to many types of projects and teams. In case of development teams with moonlight developers working for at most ten non-overlapping hours per week, not all Scrum practices can be applied. In this paper, we introduce Moonlighting Scrum, an adaptation of Scrum aimed at optimizing effectiveness and efficiency by minimizing the amount of communication to the least necessary and maximizing the time invested in development. Our aim is to accomplish this by modifying Scrum practices to achieve a trade-off between development and communication effort to produce the best final results, given the available resources and time. An application of Moonlighting Scrum took place in a real cooperative project and provided interesting results.**

Keywords-agile software development; Scrum; distributed development

I. INTRODUCTION

The adoption of agile software development processes has increased over the years. Agile methodologies are known for being lightweight, which implies moving from heavyweight processes to methods allowing shorter development cycles and more intensive customer involvement. For this reason, many companies have been moving from plan-based development to agile development.

As mentioned by Boehm and Turner [3], these two approaches to software development are considered as opponents. Software development teams often need to stick to one specific process with a defined name and tailor it to their own needs [3]. We believe that the choice of a specific process should be based on tailoring the most advantageous practices from agile and plan-based approaches that best fit the respective team's and project's needs. To answer the issue with which we are confronted, we need such tailoring and a combination of approaches.

We encountered this issue during a software development project we are currently involved in and where requirements evolve over time, which suggested that we use an agile approach. Moreover, our developers are mainly students or researchers doing development in addition to their main activities (study or research), with part-time contracts for at most eleven hours per week without time constraints. Because they are developers in their second job, we call them moonlighters.

In addition to the short time they have available to invest in the project, there is the problem that they are working during different time slots, which makes daily meetings very difficult and pair programming impossible. This suggested the use of a less agile process.

Nonetheless, and although they work part-time, there is still the need for coordinating their work and monitoring project progress.

All these variables in the context of our development projects led us to research the following questions:

- RQ1: How much communication is needed to achieve a project's goals? (Effectiveness)
- RQ2: How much communication is needed before communication overhead becomes too large? (Efficiency)

Our goal is to find an adequate balance or combination of plan-based and agile approaches which best fits the context of our development projects: distributed moonlighters working during non-overlapping times. The proposed development approach is an adaptation of Scrum, which integrates existing development methods into an agile environment. It addresses a process "to produce best end results, given the current resources and time available" [9, pg. 25]. The approach should be helpful for teams in a similar context because such a constellation is very common in software development, e.g., for open source project or at German universities.

In Section II, we discuss different methodological approaches to software development and their advantages and disadvantages for our development context. In Section III, we introduce an adaptation of a distributed Scrum method that fits our needs, called Moonlighting Scrum. In Section IV, we show how we applied the process in a real project and the measurement plan we applied. Finally, in Section V, conclusions and future work are presented.

II. RELATED WORK

Today's software development processes range from heavy weight plan-based development, such as the waterfall model [15], to incremental and lightweight agile methodologies, such as Extreme Programming [1]. The spiral model combines some aspects of the waterfall model and introduces risk management as a regular step during the process. Unlike the waterfall model, the spiral model iterates through several steps during the entire product development. On the opposite side, agile methodologies

 ISBN: 978-1-61208-304-9

include some aspects of iterative models allowing for fast reaction to changes in requirements (Figure 1).

Figure 1. Spectrum of software development processes

Nevertheless, no process fits well to every project context and therefore several other processes have appeared in the literature. In this work, we will introduce the most important approaches we took into account during the design of our model: plan-based development in general and agile development processes such as Scrum, Distributed Scrum, and Extreme Programming.

A. Plan-based Development

Plan-driven development approaches (also known as document-centered approaches) such as the waterfall [15], V-Modell XT [7], iterative, or spiral process models [4] are mainly document-centered approaches differing in their execution of the different Software Engineering (SE) phases and have several common requirements on assuring good software quality in their performance. Normally, they are performed for larger projects with larger teams, but with smaller teams the amount of project management stays almost the same [5]. Additionally, plan-based projects try to avoid refactoring because it is very expensive [2], even in a large project, as changes can influence many parts of the product. In contrast to other approaches, plan-based development covers current and future requirements in the architecture. However, this also implies early stable requirements. The developers using such an approach need to work in a plan-oriented manner and have adequate skills or access to external knowledge. The customers of products developed with such plan-based development need to be collaborative, representative, and empowered, since they are mainly involved at the beginning when it comes to making decisions about the requirements.

Plan-based approaches have several disadvantages for the project we want to perform because we only have a small team where all team members are distributed and work during different time slots. In addition, most of the requirements are not stable – and might even not be finished at the beginning of the project. This might lead to a considerable number of refactoring steps, which are expensive in plan-based development.

B. Scrum

The vast majority of Scrum practices are not new to SE. Scrum was developed at Easel Corporation in 1993 [1], basically with the same idea behind Barry Boehm's Spiral Model [4].

Scrum speeds up the requirements adaptability of the spiral model with some agile practices from Extreme Programming [13], such as pair programming and daily meetings.

Scrum is a lightweight, iterative, and incremental development model based on three principles: transparency, inspection, and adaptation.

Moreover, Scrum prescribes formal practices for inspection and adaptation:

- Sprint Planning Meeting
- Daily Scrum: daily meeting where each member answers three questions:
    - What did I do yesterday that helped the team meeting the sprint goal?
    - What will I do today to help the team meet the sprint goal?
    - Do I see any impediment that prevents me or the team from meeting the sprint goal?
- Sprint Review
- Sprint Retrospective

Because of the practical requirements, we cannot apply Scrum directly in our team but need to adapt it in a distributed way.

C. Distributed Scrum

Distributed teams always face different issues when applying development models. If we increase team distribution, we need to introduce a classification in cooperative SE using globally distributed teams [11]:

- Collocated: Team members are all in the same location.
- Collocated Part-Time: Team members are usually all in the same location but some of them occasionally work off-site. They face similar issues as distributed teams even if they have the opportunity to meet face to face.
- Distributed with Overlapping Work Hours: Team members have a few hours during the workday in which they interact with each other. Scrum meetings can be held during the overlapping time. Sprint planning meetings are more difficult and tend to be less efficient.
- Distributed with No Overlapping Work Hours: Teams have no interaction during their working hours.

In addition to the different levels of distributed teams, we also have to take into account different models that can be considered when using Scrum with distributed teams [17] (Figure 2):

- Isolated Scrums: Teams are isolated across geographies.
- Distributed Scrum of Scrums: Scrum teams are isolated across geographies and integrated by a Scrum of Scrums that meets regularly across geographies.
- Totally integrated Scrums: Scrum teams are cross-functional with members distributed across geographies. Additionally, each team has members in several locations and has its own Scrum Master.

 ISBN: 978-1-61208-304-9

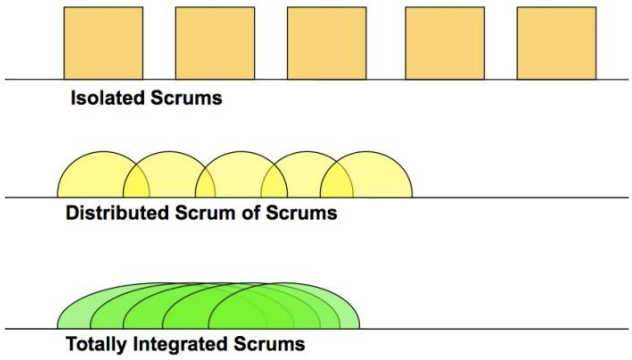

Figure 2. Strategies for distributed Scrum teams [10]

Several works report on the application of Scrum with one of these three categories [8][10][12][13][16].

Sutherland reports on two examples of project management with distributed Scrums of Scrums and fully distributed Scrums [17][18]. These works led to the conclusion that distributed teams can be as productive as small collocated teams if the entire set of teams works as a single team with a global development infrastructure (repository, tracking and reporting tool, and daily meetings). Unlike our work, all teams were composed of several developers working full time and focused on team interaction with daily meetings.

However, not much work has been done to date regarding how to reduce the effort for each team member in teams working during non-overlapping hours.

D. Extreme Programming

Extreme Programming [1] is another lightweight software development methodology, which also arose from the need for agility in the development process. Its main idea consists of taking best development practices to the extreme by eliminating anything that might interfere with productivity. The methodology emphasizes incremental development as a response to changing customer needs. Its creator Beck claims that it is especially suitable for small to medium-sized teams. The main practices include pair programming, refactoring, and simple design.

Extreme Programming has been criticized because of its lack of emphasis on design and documentation, which would encourage hacking [9]. It also requires pair programming, which suggests that it might require more effort. People also criticize that it requires constant customer availability and very disciplined teams, which could make its adoption more difficult.

For our context, Extreme Programming is the least suitable methodology, as the team members work only part-time and during different time slots.

TABLE I. COMPARISON OF DEVELOPMENT PROCESSES

	Requirements and SE Practices			**Meetings and Communication**				**Roles**				**Location and Working Hours**				
	Requirements change during project	Refactoring	Pair programming	Daily meeting	Review and retrospective	Planning meeting	Initial meeting	Scrum master	Developers	Product owner/Customer	Project manager	Collocated developers	Collocated part-time developers	Distributed teams, overl. hours	Distributed teams, non-overl. hours	Distributed developers. non-overl. hours
Plan-Based							x		x	x	x	x	x	x	x	x
Moonlighting Scrum	x	x		(x)	x	x	x	x	x	x						x
Scrum	x	x		x	x	x	x	x	x	x		x	x	x	x	
XP	x	x	x	x		x			x	x		x				

III. MOONLIGHTING SCRUM

Distributed teams with part-time developers working during non-overlapping hours are used in several projects. Moreover, at the University of Kaiserslautern, development is often assigned to students with part-time contracts, which requires them to work for a small number of hours per week, in their spare time.

Applying the existing development processes to these teams is always challenging. Table I compares some of the most important development methodologies with Moonlighting Scrum. As we can see from Table I, plan-based development and XP cannot be applied at all, while Scrum has some points in common.

Moonlighting Scrum is a Scrum extension that helps developers to structure the development process with the

goal of releasing the best product possible with the available resources in time.

Just like Scrum, Moonlighting Scrum requires sprint planning meetings, sprint reviews, and retrospectives (Figure 3). During the meetings, the whole team and the product owner must meet in person or via video conference.

In Scrum, sprints last from two to three weeks, whereas in Moonlighting Scrum they last from one to two weeks.

Because of the physical distribution and the non-overlapping time for the developers, pair programming cannot be applied and the daily meetings prescribed by Scrum cannot be attended in person.

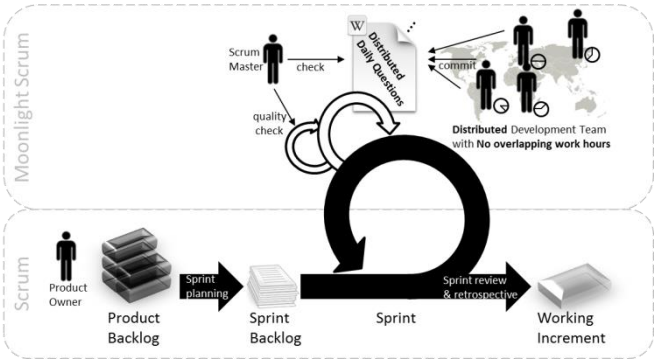

Figure 3. Moonlighting Scrum process schema

Code quality and inspection are the responsibility of the Scrum master, who is in charge of checking overall quality and help the developers preserve a minimum amount of code quality. Moonlighting Scrum is thought to deliver the highest quality possible if limited resources are available.

Therefore, as reported in [12], we substituted morning meetings with an online forum by creating a thread for every six working hours to which each developer posts his/her comments by replying to three questions:

- What have you completed, with respect to the sprint goal, since the last daily meeting?
- What specific tasks, with respect to the sprint goal, do you plan to accomplish until the next daily meeting?
- What obstacles got in the way of completing this work?

The Scrum Master also has to take care of communication efficiency by reducing or increasing the online reporting interval, and is in charge of increasing or decreasing the reporting time based on the team's efficiency.

For this reason, the team members must also answer two additional questions in their online report:

- When did you work (start-end)?
- How much time did you spend on writing this report?

The developers are working for at most ten hours per week and are requested to work for at least two hours continuously. Consequently, the time needed to write the report at the beginning and at the end of their work might take up an important percentage of their working time.

In classical Scrum, daily meetings take 15 minutes. Taking into account 40 working hours per week, daily meetings should take up approximately 3% of the working time.

In contrast, Moonlighting Scrum requires an online report, which usually takes from 5 to 8 minutes, every four to six working hours [12], with at least one report per week. If the developers work for more than six hours per week, they are requested to report twice. The estimated working time used for both cases is 3.5%.

TABLE II. EFFORT REQUIRED

	Hours/ week	Weeks/ sprint	Hours/ meeting	Minutes/ daily report
Scrum	40	2-3	4	15
Moonlighting Scrum	4-10	1-2	2	8

Sprint planning, review, and retrospective meetings in Scrum take four hours per sprint, with sprints lasting from two to three weeks and effort ranging from 3.3% to 5% [12][14].

In Moonlighting Scrum, meetings take suggested two hours with an approximate effort ranging from 6.6% to 12.5% (Table II).

Taking into account the communication issues in a highly distributed team with non-overlapping hours, communication time does not grow significantly, ranging from a maximum of 8% in Scrum to a maximum of 15.5% in Moonlighting Scrum (Table III).

TABLE III. ESTIMATED COMMUNICATION

	Meeting time	Reporting time	Overall time
Scrum	3.3% - 5%	3%	6.3%-8%
Moonlighting Scrum	6.6%-12.5%	3%	9.6%-15.5%

Moonlighting Scrum is applicable to a wide range of projects, from university- and research-based projects to open source projects. In general, the process requires more relative effort for communication than Scrum but allows developing code in a controlled and structured way. The process is applicable whenever we are faced with distributed developers working during non-overlapping hours.

IV. APPLICATION OF MOONLIGHTING SCRUM

Moonlighting Scrum has been applied for the initial development of the software project Technology Repository and Process Configuration Framework [6]. The development started in February 2013 and the first version of the tool was released at the end of May 2013.

 ISBN: 978-1-61208-304-9

A. Team Organization

The development team is composed of six people. Some of them are employees of Fraunhofer Institute for Experimental Software Engineering IESE, while the others work for the "Software Engineering Research Group - Processes and Measurement" of the University of Kaiserslautern. All developers have an intermediate level of experience in software development, while none of them has any experience with agile methodologies.

The developers work part-time, with weekly working hours ranging from four to ten, and together spent a total of 39 hours per week on this project.

In order to manage the whole project, from development to communication aspects, we adopted a development infrastructure covering several aspects. The team meets in person during the sprint meeting or, in exceptional cases, via video conference, whereas online reports are recorded in a forum by creating a post for each report.

Sprint retrospectives, planning, and retrospective discussions are led by means of an online integrated tool [19], which allows us to record sprint reports, manage product backlog, and draw burn-down charts.

In addition to this infrastructure and in order to increase collaboration between team members, we also set up a Subversion [13].

B. Process Measurement and Improvements

In order to answer the research questions (RQ1 and RQ2), we defined a Goal-Question-Metric measurement plan that allowed us to derive appropriate productivity and communication metrics that impact on effectiveness and efficiency (Table IV).

To define a usable measure for productivity, we considered User Stories (US) as the basic measurement unit.

Since the development is carried out by means of a Rapid Application Development (RAD) Tool (Microsoft Visual Studio 2012), we do not collect code metrics such as lines of code, code complexity, or other metrics because the vast majority of the code is generated automatically by the RAD tool; however, we did define metrics.

Communication time is expressed in terms of time needed to write the online reports and attend the sprint meetings. Total communication time is calculated summing up these two per person. As an example, the training sprint meeting lasted 120 minutes but considering that five people attended the meeting, the total time for the training sprint was 600 minutes (10 hours).

As shown in Table IV, we managed to achieve a sprint meeting duration of two hours or less, except for sprint 1 where the vast majority of topics involved training issues related to the previous training sprint. Total communication time (without reading the online reports) decreased and became stable after two sprints, with effort ranging from 13% to 18%.

On average, communication time required 17% of the total time: 16.4% for the sprint meetings, 0.6% for the online reports, and 83% for development. As a result of this experiment, communication time was slightly higher (17%) than expected (9.6%-15.5%).

TABLE IV. MOONLIGHT SCRUM COLLECTED DATA

	Productivity				Communication		
	# days per Sprint	Tot working hours	#Assigned User Stories	#Completed User Stories	Online report time (minutes)	Sprint meeting time (minutes)	Communication time/total time (%)
Training Sprint	10	16	3	3	8	120	63%
Sprint 1	9	50	4	3	26	150	26%
Sprint 2	10	56	6	5	30	120	18%
Sprint 3	14	78	7	6	22	120	14%
Sprint 4	15	84	7	5	27	90	13%
Sprint 5	10	56	5	4	16	120	14%
Sprint 6	11	61	7	5	18	120	17%
Sprint 7	11	61	5	4	26	120	17%
Sprint 8	12	67	3	2	24	120	17%

The application of this process will continue for another three months for project maintenance.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a solution aimed at finding an adequate process for distributed teams with part-time developers working during non-overlapping hours who only have a small amount of effort available per week (ten or less hours per week). Our idea consists of making a trade-off between plan-based and agile development processes. The proposed process is an adaptation of Scrum aimed at optimizing the effectiveness and efficiency of the developers. This means that our goal is to optimize productivity by minimizing the amount of communication to the minimum necessary and maximizing the time invested in development. Our aim is to achieve this with the following instruments:

- Sprint planning, sprint reviews, and retrospective meetings are done in person or via video conference;
- Developers must work for a minimum of two continuous hours;
- Daily meetings are replaced by writing a report in an online forum every six working hours;
- Developers voluntarily report the effort they invest into development and reporting;
- Scrum Master performs code reviews.

The application of Moonlighting Scrum on a real project confirmed that the process can be successfully

 ISBN: 978-1-61208-304-9

applied in the university context and helps to keep track of the development steps and to maintain low communication effort.

The project where we applied Moonlight Scrum will continue for another three months for the maintenance phase.

As expected, the process helped us keep track of the development progress. After some initial training and after resolving some technology issues resulting from the new activities required from our developers as well as from the complexity of the domain infrastructure (cyber-physical systems), we were able to maintain low communication overhead.

In future, we will encourage our colleagues working on similar projects to use Moonlighting Scrum process to obtain more evidence to improve it. This should also be done with some open-source development as well as industrial projects to generalize the results.

In addition to this generic aspect we will also try to improve the approach by using other collaboration tools or improving the communication with an online-chat conference system.

ACKNOWLEDGMENT

This paper is based on research being carried out in the ARAMiS (BMBF O1IS11035Ü) project funded by the German Ministry of Education and Research (BMBF).

REFERENCES

[1] K. Beck andC. Andres,"Extreme programming explained. Embrace change" Addison-Wesley Boston, ed. 2, 2005, pp. 64-69.

[2] B. Boehm,"Get ready for agile methods, with care" Computer, vol. 35(1), Jan. 2002, pp. 64-69, doi: 10.1109/2.976920.

[3] B. Boehm and R. Turner,"Balancing agility and discipline. A guide for the perplexed" Addison-Wesley Boston, 2004.

[4] B. Boehm,"A spiral model of software development and enhancement" IEEE Computer, vol. 21(5), May 1988, pp. 61-72, doi:10.1109/2.59 .

[5] M. Ceschi, A. Sillitti, G. Succi, and S. De Panfilis,"Project management in plan-based and agile companies" IEEE Software, vol. 22(3), May-June 2005, pp. 21-27, doi:10.1109/MS.2005.75.

[6] P. Diebold, "How to configure SE development processes context-specifically?"Proc. International Conference on Product-Focused Software Development and Process Improvement (PROFES), Springer LNCS, June 2013, pp. 355-358, doi:10.1007/978-3-642-39259-7_33.

[7] German Federal ministry of the Interios, „V-Model XT. Definition and Documentation on the Web."Online, http://www.v-model-xt.de, July 2013.

[8] I. Gorton and S. Motwani,"Issues in cooperative software engineering using globally distributed teams" Information and Software Technology, vol. 38(10), Jan. 1996, pp. 647-655, doi:10.1016/0950-5849(96)01099-3.

[9] J. Hunt,"Agile software construction"Springer London, 2005.

[10] J. Kontio, M. Hoglund, J. Ryden, and P. Abrahamsson, "Managing commitments and risks: challenges in distributed agile development" Proc. International Conference on Software Engineering (ICSE), IEEE Press, May 2004, pp.732-733, doi:10.1108/ICSE.2004.1317510.

[11] M. Korkala, and P. Abrahamsson, "Communication in distributed agile development: a case study"Proc. EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE Press,Aug. 2007, pp. 203-210, doi:10.1109/EUROMICRO.2007.23.

[12] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi, "Applying SCRUM in an OSS Development Process: Empirical Evaluation." Proc. International conference on Agile Software Development (XP), Springer LNBI, June 2010, pp 147-159, doi:10.1007/978-3-642-13054-0_11.

[13] N. Sridhar, M. RadhaKanta, and M. George, "Challenges of migrating to agile methodologies." Communications of the ACM – Adaptive complex enterprises (CACM), vol. 48, May 2005, pp. 72-78, doi:10.1145/1060710.1060712.

[14] M. Paasivaara, S. Durasiewicz, and C. Lassenius, "Using scrum in a globally distributed project: a case study." Software Process: Improvement and Practice – Global Software Development, vol. 13(6), Nov. 2008, pp. 527-544, doi:10.1002/spip.v13:6.

[15] W. Royce, "Managing the development of large software systems: concepts and techniques" Proc.Technical Papers of Western Electronic Show and Convention (WESCON),IEEE Press, Aug. 1970, pp. 1-9.

[16] J. Sutherland,"Agile development: lessons learned from the first Scrum"Cutter Agile Project Management Advisory Service: Executive Update, vol. 5, 2004, pp. 1-4.

[17] J. Sutherland, G. Schoonheim, and M. Rijk, "Fully distributed scrum: replicating local productivity and quality with offshore teams." Proc. Annual Hawaii International Conference on System Sciences (HICSS), IEEE Press, Jan. 2009, pp. 1-8, 2doi:10.1109/HICSS.2009.225.

[18] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov,"Distributed Scrum: agile project management with outsourced development teams." Proc. Annual Hawaii International Conference on System Sciences (HICSS), IEEE Press, Jan. 2007, pp. 274a.

[19] RallyDev http://www.rallydev.com (Last access July 2013)

 ISBN: 978-1-61208-304-9