2015 Agile Conference

Development of Complex Software with Agile Method

Alan Braz
IBM Research - Brazil
Avenida Tutóia 1157, 04007-005
São Paulo, SP, Brazil
alanbraz@br.ibm.com

Cecília M. F. Rubira
Institute of Computing
State University of Campinas
P.O. Box 6176, 13083-852,
Campinas, SP, Brazil
cmrubira@ic.unicamp.br

Marco Vieira
Department of Informatics
Engineering
University of Coimbra
3030, Coimbra, Portugal
mvieira@dei.uc.pt

***Abstract*—Agile Software Development (ASD) has been on mainstream through methodologies such as XP and Scrum enabling them to be applied in the development of complex and reliable software systems. This paper is the end result of the Master's dissertation of the main author, and proposes a solution to guide the development of complex systems based on components by adding exceptional behavior modeling practices to Scrum, resulting in the Scrum+CE method (Scrum with Exceptional Behavior).**

In order to evaluate the proposed method, a synthetic controlled experiment was conducted with three groups.We compared the efficiency of the new process in relation to plain Scrum and the results were the production of a better quality software but with less features implemented during the same amount of time.

***Keywords*—Agile Software Development, Scrum, Component-Based Development, Reliability, Exception handling, Software Engineering**

I. Introduction

The Software Engineering has been suffering changes in its methods and practices due to the growing need to develop systems in shorter periods with lower costs and satisfying quality. By contrast, the complexity of these systems is increasing as well as the need for them to be reliable. Dependability is no longer a mere nonfunctional requirement just inherent to critical systems, such as a flight controller or a financial system, but of all software systems that require a certain robustness to not expose sensitive information and maintain a service dependable as much as possible. To address these issues, ideas like Component-Based Development (CBD) [1] and Architecture-Centric Development [2] have been applied with relative success on developing more reliable software.

However, there is a trend of using "lighter" methods of development and project management software. This set of principles and practices called lightweight were named Agile Methods or Agile Software Development (ASD) methods by several professionals who met in 2001 to formalize what had already been doing for some years, as Extreme Programming (XP) [3] and Scrum [4], resulting in the Agile Manifesto [5].

Nevertheless, this simplicity is often confused with informality and lack of rigor, especially regarding the activities of modeling, documentation of functional and nonfunctional requirements and architectural design.

This article proposes a solution named Scrum+CE to guide the development of robust component-based systems that adds some practices of MDCE+ to the Scrum framework. These practices affect the Pregame and Game phases in the aspect of discovering and modeling exceptional conditions in the format of Exceptional Stories and more detailed Acceptance Tests. Scrum+CE also inserts a required artifact of the High Level Architecture and the new role of Architecture Owner.

The methodology used to validate the proposed solution was *Synthetic Environment Experiments* [6], where a smaller version of Scrum and Scrum+CE were executed. This reduction had to be made due to the availability of human resources and time to execute the experiment.

The validation hypotheses were that using Scrum+CE would result in the delivery of (i) less Story Points, since the requirements are more detailed; and (ii) better quality of the final software in terms of less defects, once it will drive to a more robust exception handing code.

II. Related work

Radinger and Goeschka [7] proposed an approach to integrate the ASD and the CBD in small and large projects by combining the technical and organizational issues of both approaches.

Nord and Tomayko [8] explored the relationships and synergies between design and analysis focused on the architecture-centric method and XP Agile methodology, noting that the latter emphasizes the rapid and flexible while the first priority to the design and infrastructure.

Behrouz [9] have discussed and shown that despite the different goals, a combination of Software Reliability Engineering (SRE) and the Agile practice of Test Driven Development (TDD) have improved the reliability of the developed software. In this case, we did not focus on TDD approach to increase dependability, but we achieved it by adding elements of exception handling at early phases of development process.

III. Background

A. Agile Method

Agile Software Development (ASD) is a set of methodologies guided by four values and twelve principles defined by the Agile Manifesto [5]. These values are:

978-1-4673-7153-7/15 $31.00 © 2015 IEEE
DOI 10.1109/Agile.2015.18

1) **Individuals and interactions** over processes and tools
2) **Working software** over comprehensive documentation
3) **Customer collaboration** over contract negotiation
4) **Responding to change** over following a plan

The twelve principles expand the values in more details by giving more emphasis to the left items of values than the right ones.

B. Scrum Methodology

Scrum is an agile software management framework, that promotes an iterative and incremental development. It was introduced in 1995 by Schwaber [4] proposing a process with three phases: Pregame (Conception or Initiation), Game (Development) and Postgame (Closure or Rollout).

Despite the Scrum has evolved a lot since the Schwaber's 1995 [4] paper in terms of removing the software engineering practices and focusing on team organization and project management, for the sake of this work, we choose to use the first academic reference due to the need to handle exceptional behavior at the architecture level. This decision was made because the High Level Architecture, in our proposed solution, must begin before the first development Sprint, there is, at the Pregame.

C. Methodology for Exceptional Behavior Definition (MDCE+)

Ferreira [10] presented a methodology for building fault-tolerant systems using techniques of exception handling.This methodology is named MDCE, acronym in Portuguese for "Methodology for the Definition of Exceptional Behavior". It extends the Unified Modeling Language (UML) with new stereotypes.

Brito [11] extended MDCE defining MDCE+ which aims to systematize the modeling and implementation of exceptional behavior in the development of component-based systems. The emphasis on architecture has enabled a better analysis of the flow of exceptions that occur between architectural components, resulting in efficient handlers, and anticipating the correction of possible specification faults.

IV. The Proposed solution

The proposed solution is an adapted agile development process based on Scrum process that adds some MDCE+ practices and techniques to increase the robustness of the developed complex product. The name of this proposed process is Scrum+CE (the CE comes from Exceptional Behavior in Portuguese) and it adds practices at the Pregame and Game phases of Scrum in order to raise, detail and document the exceptional behavior in the format of Exceptional Stories and Acceptance Tests more judicious inside the regular User Stories [12], and also refines and documents the architecture highlighting the exceptional components.

Figure 1 shows the phases of Scrum with the activities from MDCE+ in gray. Scrum+CE doesn't add new phases to Scrum, therefore it still has Pregame, Game and Postgame. Table I shows the relation between the phases of Scrum and MDCE+. Note that due to the iterative aspect of the Game phase, composed by sprints, some MDCE+ phases will be repeated.

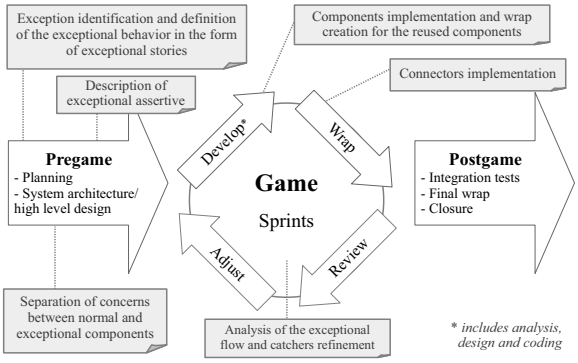

Fig. 1. MDCE+ practices affecting Scrum phases. Extended from [4]

TABLE I. Relation between MDCE+ and Scrum phases.

Scrum Phases	Scrum Events	MDCE+ Phases
Pregame	Planning	1. Requirements specification and analysis 2. Management aspects definition
	System architecture / high level design	3. Architecture design
Game	Sprint Planning	1. Requirements specification and analysis 3. Architecture design 4. System analysis 5. System design
	Sprint	6. Components implementation 7. Components integration
	Sprint Review	1. Requirements specification and analysis 8. User-acceptance release
Postgame	Integration tests	7. Components integration
	Wrapping	8. Production Release deploy
	Closure	

A. Pregame

1) Planning: Besides the regular activities of the Planning phase of Scrum there were added the following activities (i) identify the exceptions and define the exceptional behavior in the form of Exceptional Stories and (ii) describe the exceptional assertive in the form of Acceptance Tests either at the normal behavior User Stories or at the introduced Exceptional Stories.

During the review of the Product Backlog, all stories should be revisited in order to add extra Acceptance Tests, derived from the Exceptional Stories, resulting in a formalization of the exceptional assertive and in consequence generating a more robust test set.

2) Definition of "Done": Exceptional behavior should also be explicit regardless the definition agreed by the entire team. The Architecture Owner has the responsibility of adding the following to it: "...and all exceptions were properly handled..".

3) System architecture / high level design: Just like Planning, this phase still formatted by the Scrum activities plus a new role, the Architecture Owner, and a new mandatory artifact, the High Level Architecture document. Both additions have brought the concepts of Architecture-Centric Development to Scrum+CE. The architecture has to follow the concepts of CBD and have at least a component-level architecture

diagram, the most important architectural decisions, used technologies and frameworks, and an initial data model. It is recommended to document it in a collaborative tool, like a Wiki. This would allow the document to be highly available, flexible and support the Agile Manifesto principle that states: "The best architectures, requirements, and designs emerge from self-organizing teams".

B. Game

Scrum+CE is also an iterative process in the format of time-boxed sprints with length between two and four weeks that repeat continuously until the implementation of every requirement of the Product Backlog or the Product Owner declares that the current Increment is valuable as a final product. With that, the Scrum events, structure and activities remain the same.

The MDCE+ practices, highlighted at Figure 1, will affect the Product Backlog with the addition of Exceptional Stories, and the Sprint Backlog, with the related tasks dedicated to the implementation of exceptional behavior. Once inside the backlogs, either exceptional stories or tasks will the treated by the Development Team as ordinary ones.

C. Postgame

The Postgame still with the wrapping and preproduction activities to the end-user release and did not have any change in relation to Scrum.

V. Validation

In order to validate the benefits and applicability of Scrum+CE, we designed a controlled experiment in which a information-based software system, with dependability requirements relating to data consistency, were implemented. The experiment consists of an *object* (P1), that is, the software system in Java for the web platform along with its predefined architecture following the Model-View Controller (MVC) [13] pattern, and three *subjects*, that is, three professional groups (G1, G2 and G3) with experience this technology that will be part of the Development Team, as described by Table II;

TABLE II. Experiment modeling.

Object	G1	G2	G3
P1	O1: Scrum	O2: Scrum+CE	O3: Scrum+CE

Using this format, G1 was the control group, and consequently G2 and G3 were the experimental groups. This kind of experiment followed a controlled method called Synthetic Environment Experiments [6] where a reduced version, in terms of duration time, of Scrum and Scrum+CE were executed. So each Sprint will last 1 week (4 hours per day during 5 coonsecutive days), with a 2-hour virtual work-day including a 3-minute Daily Scrum meeting.

All the professionals selected to implement the project of the experiment have at least 3 years of working experience developing software in Java and basic training in Agile. The fact that all participants were proficient in Java and Scrum, helped to strengthen the internal validity of the experiment, that is, the results would be influenced by the development process instead of their previous knowledge.

The Pregame and Postgame phases were done by the author of this paper, while the Game phase was done by the participants. The three implemented source-codes were available to the execution of functional tests and the collection of metrics used further in the result analysis.

A. Pregame execution

In this phase it was defined the product Vision and Product Backlog with all stories prioritized and estimated in Story Points. The role of Architecture Owner of Scrum+CE was conducted by the author for the groups G2 and G3, as well as the roles of Product Owner and Scrum Master for all three groups. This participation was necessary to control the variables, requirements, artifacts and adherence to processes by groups, thus, maintaining the validity of the experiment. Every artifact was presented to each group independently.

1) Scrum groups division: The participants were invited to participate in a "*Scrum in Practice*" training. they had to fill out a registration form composed by knowledge and experience questions about Agile, Scrum and Java development and 12 participants were selected. It was created a technical score for each participant. The scores were distributed among three groups. The scores from G1 to G3 respectively were: 253 (36%), 225 (32%) and 233 (33%). G1 had the highest relative score so it was intentionally choose to be the control group.

2) System architecture / high level design: The architecture of the system was presented for the three groups during the first Sprint Planning Meeting and followed a simple four layer model as illustrated at Figure 2. In this case, the layers User Session and System Services were unified and further layers of server side followed the MVC [13] design pattern. This architecture was used in Scrum+CE groups G2 and G3. Even not required in Scrum, the same diagram was given to the control group G1, but without the two exceptional components.

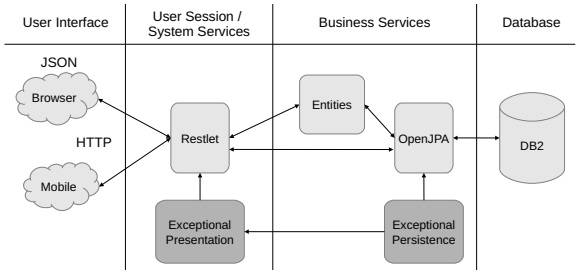

Fig. 2. Component-based software architecture diagram exposing the exceptional components used by the experimental groups G2 and G3.

From the CBD standpoint, each layer was treated as a component, and the goal of Game phase was the implementation of such components.

B. Game execution

The experiment itself was implement in a format of a 40-hour *Scrum in Practice* training. The time was divided in 2 sprints of 20 hours, and each sprint was composed by 7 virtual days of 2 hours each. With that, it was possible to simulate rigorously the Scrum and Scrum+CE processes. The activities of the three groups were conducted in parallel with a difference

of about ten minutes between a group and another due to movement of the author between the rooms.

1) Sprint 1: At the beginning of the experiment, all participants were gathered in the same room where they were notified about their respective groups and directed to a exclusive meeting room. After that the groups started the Sprint Planning Meeting and then followed the schedule described at Table III. Since no initial code was provided, all groups selected only two User Stories to be implemented in this first Sprint.

At the end of the fifth day, playing the role of Product Owner, the author made an individual Sprint Review Meeting with each group when they demonstrated the features implemented during this sprint. Only G1 delivered both stories. The other two groups, delivered only one story each.

So at the end of this sprints the Product Backlog of the groups was different. At this point, G1 had two stories completed, and G2 and G3 only one.

TABLE III. SCHEDULE OF SPRINT 1.

Day	Activity	Duration
1	Reception and organization	2 hours
	Planning Meeting	2 hours
2	Daily Scrum	3 minutes
	Implementation day 1	2 hours
	Daily Scrum	3 minutes
	Implementation day 2	2 hours
3	Daily Scrum	3 minutes
	Implementation day 3	2 hours
	Daily Scrum	3 minutes
	Implementation day 4	2 hours
4	Daily Scrum	3 minutes
	Implementation day 5	2 hours
	Daily Scrum	3 minutes
	Implementation day 6	2 hours
5	Daily Scrum	3 minutes
	Implementation day 7	2 hours
	Review Meeting	2 hours

2) Sprint 2: It had a slightly different schedule but followed the same structure of Table III besides the first and last days. Day 1 was composed by a Planning Meeting of 2 hours, a Daily Scrumof 3 minutes and the Implementation day 1 with 2 hours. Day 5 had a Review Meeting with 3 hours and a Retrospective that took 1 hour.

In this sprint, the groups started the Sprint Planning Meeting reviewing their own updated Product Backlog and selecting the stories that they would commit with the Product Owner to deliver at the end of Sprint 2. At the last day, we gather all participants at the same room and made a collective Sprint Review Meeting. Every group presented the results of their implementation and the author, again as Product Owner, validated them all. The list of the stories delivered by each groups is at Table IV.

The Sprint Retrospective was a collective session where we discussed how was the feeling of using an agile method with time-boxed events to develop complex software. Then it was reveled to the participants that the process used by the groups were different.

C. Postgame execution

The first activity of this phase was to build a set of functional tests to run against each source code available. It was created several test cases, based on the Acceptance Tests, for each delivered story.

The metrics collected in this phase were divide in three categories: (i) number of requirements; (ii) quality of requirements; and (iii) quality of code.

1) Requirements delivered metrics: A User Story was only considered completed if it respected the Definition of "Done". Table IV shows the entire Product Backlog with the stories delivered by each group and its correspondent Story Points.

TABLE IV. USER STORIES DELIVERED BY GROUP.

Story	Points	G1	G2	G3
1	20	●	●	●
2	5	●	●	●
3	8	●	●	●
4	3	●	●	●
5	3	●	●	●
6	2	●		●
7	1	●	●	●
8	3			
9	5	●	●	●
10	2	●		
11	2			
12	5			
13	2			
Total stories		9	7	8
Total points		49	45	47

2) Requirements quality metrics: Once the architecture was designed to interact with the system using HTTP requests, it was built a test suite to run the same set of tests against the different codes automatically. Table V consolidates the results of the functional tests showing the number of tests executed against each group code, the amount of failed tests (defects) and the fail rate.

TABLE V. NUMBER OF TESTS EXECUTED AND DEFECTS FOUND BY GROUP.

Metric	G1	G2	G3
Tests executed	189	149	161
Failed tests	66	44	21
Fail rate	35%	30%	13%

3) Code quality metrics: The delivered codes were submitted to a static analysis in order to evaluate the code quality. It was used an automated and open-source toll called Sonar [14]. Table VI shows some relevant metrics provided by this tool.

TABLE VI. CODE QUALITY METRICS.

Metric	G1	G2	G3
Lines of Code (LOC)	2232	1984	1950
Number of Classes	27	47	38
Number of Exceptions	1	21	2
Native *catch* blocks	53	18	33
Created *catch* blocks	9	12	6
Cyclomatic Complexity (CC) [15]	446	305	288
CC by class	15.9	4.5	7.2
CC by method	5.0	2.4	2.5

VI. RESULTS ANALYSIS

Due to the size of the experiment it was not possible to make a statistical analysis of the collected data. Thus, a quantitative analysis was performed comparing the control group (G1) and each of the experimental groups (G2, G3), resulting in two sets of results: G1-G2 and G1-G3.

A. Requirements delivered

Figure 3 shows the comparison between the User Stories and Story Points delivered.

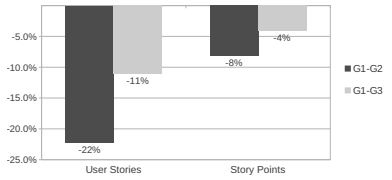

Fig. 3. Stories relation between G1 and G2, and between G1 and G3.

Analyzing these numbers, Scrum+CE resulted in fewer Story Points (6.1% on average) delivered once the two experimental groups gave fewer stories (16.7% on average) which was expected and should be compensated with a better code quality.

Another metric that supports this issue is the amount of Lines of Code (LOC) of the delivered systems, presented in line 1 of Table VI. It highlights that the code delivered by G1, which implemented a greater number of stories, was higher than the other groups. The relative size of G2 to G1 was 11% smaller, and to G3, 13% smaller. With that, the size of the system in LOC is directly proportional to the amount of functionality delivered.

B. Requirements quality

The amount of tests executed, listed at Table V, were directly proportional to the delivered requirements at Table IV. Since only delivered User Stories were tested, G2 and G3 were exposed to fewer amount of tests.

The fail rate is the metric that best represents the results of the experiment and helps to validate the hypothesis in which the use of Scrum+CE over Scrum would increase the "code quality generated by decreasing the number of defects". The fail rate of each group was 35%, 30% and 13% respectively for G1, G2 and G3, as shown in Table V. To support this analyses, we can rely on Table VI to check that G2 and G3 took different approaches to design the exceptions (Number of Exceptions and Created *catch* blocks). In other hand, they delivered equivalent size in terms of Lines of Code (LOC) and Cyclomatic Complexity (CC).

Considering that a failed test generates a defect, these results indicate a validation of the hypothesis, since the experimental groups had lower fail rates than the control group. Therefore, the usage of Scrum+CE promoted the creation of a better quality code when compared to G1.

VII. Consolidated results

As a result of the experiment, we demonstrated that the expectation over the method Scrum+CE was proved once there was **delivered 6.1% less Story Points than using Scrum**, but with a **decrease of 13.5% in the number of defects found**. Therefore, using the Scrum+CE delivered less Story Points but with a better quality code (fewer defects) in relation to the development that used the Scrum.

VIII. Conclusion

Despite the size of the experiment, in terms of number of participants and duration, we couldn't apply any statistical analysis. So we were unable to generalize this results to every kind and size of complex software project with some reliability requirement. During this work we realize how hard and expensive it is to make software engineering experimentation due to the need of people availability for a long period of time. Even with this challenges, we considered a great experience during the author's master degree dissertation work.

The same experiment could be replicated with more groups or even bigger groups, implementing the same project requirements, for the same time-frame, in order to collect more data, and eventually make it possible some sophisticated statistical analysis at the results. This is a suggestion of future work that would reinforce this study.

This work also inspired the software engineering teachers of our institute to remodel the software engineering practice courses of the Computer Science and Computer Engineering undergraduate courses. We supported this changes at the course agenda and execution for both 2014 semesters obtaining amazing results and feedbacks from the students and teachers. These experiences will be published in the near future.

References

[1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

[2] I. Sommerville, *Software Engineering*, 9th ed. Harlow, England: Addison-Wesley, 2010.

[3] K. Beck, *Extreme Programming Explained: Embrace Change*, US ed. Addison-Wesley Professional, Oct.

[4] K. Schwaber, "SCRUM Development Process," in *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 117–134.

[5] K. Beck *et al.* (2001) Manifesto for Agile Software Development. Accessed: 17 Apr 2015. [Online]. Available: http://agilemanifesto.org

[6] M. V. Zelkowitz and D. Wallace, "Experimental Validation In Software Engineering," *Information and Software Technology*, vol. 39, pp. 735–743, 1997.

[7] W. Radinger and K. M. Goeschka, "Agile software development for component based software engineering," *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Oct. 2003.

[8] R. L. Nord and J. E. Tomayko, "Software Architecture-Centric Methods and Agile Development," *IEEE Software*, vol. 23, no. 2, pp. 47–53, Mar. 2006.

[9] B. Far, "Software Reliability Engineering for Agile Software Development," *20th IEEE Canadian Conference on Electrical and Computer Engineering CCECE*, pp. 694–697, 2007.

[10] G. R. M. Ferreira, "Tratamento de exceções no desenvolvimento de sistemas confiáveis baseados em componentes," Master's thesis, IC, Unicamp, Dec. 2001.

[11] P. H. S. Brito, "Um Método para Modelagem de Exceções em Desenvolvimento Baseado em Componentes," Master's thesis, IC, Unicamp, Oct. 2005.

[12] M. Cohn, *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004.

[13] S. Stelting and O. Maassen, *Applied Java Patterns*. Prentice Hall Professional, 2002.

[14] SonarQube™. Accessed: 16 Oct 2012. [Online]. Available: http://www.sonarsource.org

[15] T. J. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.*, vol. 2, no. 4, pp. 308–320, 1976.