Comparing Requirements Decomposition Within the Scrum, Scrum with Kanban, XP, and Banana Development Processes

Davide Taibi[1(✉)], Valentina Lenarduzzi[1], Andrea Janes[1], Kari Liukkunen[2], and Muhammad Ovais Ahmad[2]

[1] Free University of Bolzano/Bozen, Bolzano, Italy
{davide.taibi,valentina.lenarduzzi,andrea.janes}@unibz.it
[2] University of Oulu, Oulu, Finland
{kari.liukkunen,muhammad.ahmad}@oulu.fi

Abstract. **Context:** Eliciting requirements from customers is a complex task. In Agile processes, the customer talks directly with the development team and often reports requirements in an unstructured way. The requirements elicitation process is up to the developers, who split it into user stories by means of different techniques. **Objective:** We aim to compare the requirements decomposition process of an unstructured process and three Agile processes, namely XP, Scrum, and Scrum with Kanban. **Method:** We conducted a multiple case study with a replication design, based on the project idea of an entrepreneur, a designer with no experience in software development. Four teams developed the project independently, using four different development processes. The requirements were elicited by the teams from the entrepreneur, who acted as product owner and was available to talk with the four groups during the project. **Results:** The teams decomposed the requirements using different techniques, based on the selected development process. **Conclusion:** Scrum with Kanban and XP resulted in the most effective processes from different points of view. Unexpectedly, decomposition techniques commonly adopted in traditional processes are still used in Agile processes, which may reduce project agility and performance. Therefore, we believe that decomposition techniques need to be addressed to a greater extent, both from the practitioners' and the research points of view.

1 Introduction

Eliciting requirements from customers is a complex task. In Agile processes, the introduction of the product owner usually facilitates the process, suggesting that the customer talk directly with the development team and thus reducing the number of intermediaries. However, the product owner, especially when he or she is not an expert in the project domain, reports requirements in natural language, in their own words, and often in an unstructured way.

The requirements elicitation process is up to the developers, who usually split it up into user stories in the case of Agile processes.

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 68–83, 2017.
DOI: 10.1007/978-3-319-57633-6_5

To the best of our knowledge, there are no studies that have attempted to understand how requirements are decomposed in Agile processes and, moreover, no studies that compare requirements decomposition among different Agile processes or other processes.

To bridge this gap, we designed and conducted the first such empirical study, with the aim of comparing the requirements decomposition process of an unstructured process and three Agile processes, namely XP, Scrum, and Scrum with Kanban [21]. We conducted the study as a multiple case study with a replication design [1] since it was not possible to execute a controlled experiment because of the unavailability of developers for the major effort required. We selected four groups of second-year master students as participants, which constitute a good sample of the next generation of developers entering the job market. They were perfectly suited for this task since the project did not require the use of new technologies unknown to the students, and they can thus be viewed as the next generation of professionals [10–13]. Students are perfectly suitable when the study does not require a steep learning curve for using new technology [13, 17].

We selected a project idea to be developed by means of an idea contest for entrepreneurs, selecting an idea from a designer with no experience in software development. This project idea was then developed by four teams using four different development processes. The requirements were elicited by the teams from the same entrepreneur who acted as product owner with all four groups.

The results show interesting differences regarding requirements decomposition. The team that developed in XP decomposed a lot more stories, followed by the one using Scrum with Kanban, then the one using Scrum, and finally the team using the unstructured process. Another interesting result is related to the development effort, which was perfectly inversely proportional to the number of user stories decomposed, resulting in the highest effort for the unstructured process and the lowest for the XP one.

This paper is structured as follows. Section 2 introduces the background and related work. Section 3 presents the multiple case study and Sect. 4 the results obtained. Section 5 describes the threats to validity and Sect. 6 draws conclusions and future work.

2 Background and Related Work

The term “user story decomposition” describes the act of breaking a user story down into smaller parts [8]. User stories are typically decomposed into parts that have a scope that is large enough to provide value to the customer but small enough so that the effort for implementing the story can be estimated with a low risk of being wrong. A story with a smaller scope is likely to be less complex than a story with a large scope. Moreover, if the scope is large, more things can go wrong, e.g., unknown details might emerge, the architecture may be inadequate, and so on [4]. Altogether, the expectation is that it should be easier to estimate the effort for developing a small story than that for a large one. As a consequence, sprint planning, i.e., defining which stories the team should be able to complete during a sprint, is more likely to be accurate with small user stories.

Additionally, developing stories with a smaller scope allows the team to complete a user story more often than if it were to develop only a few large user stories. This allows

it to regularly deliver business value to the customer, with the consequence that the customer can provide feedback earlier, allowing the team to learn faster which requirements the system being developed should fulfill.

A popular technique for decomposing user stories is "User Story Mapping" [9], which decomposes the stories from the user's point of view, i.e., it decomposes the flow of user activities "into a workflow that can be further decomposed into a set of detailed tasks" [8]. User Story Mapping uses the terms "activity", "task", and "subtask" to describe high-level activities (e.g., "buy a product"), tasks (e.g., "manage shopping cart"), and subtasks, i.e., the decomposed user stories, which are the ones assigned to developers (e.g., "add product to shopping cart"). Rubin uses the terms "epic", "theme", and "sprintable story" to apply it within Scrum [8].

Outside of an Agile context, the decomposition of requirements into different parts has been discussed to prepare their technical implementation: for example, [14] describes techniques used in service-based applications to decompose complex requirements in order to reuse relatively simple services; in [15], the authors develop a technique for matching parts of the requirements to COTS components; in [16], the authors discuss how to decompose architectural requirements to support software deployment in the cloud; in [17, 20], the authors study conflicting requirements; in [18], the authors propose an extension to UML to allow decomposing use case models into models at several levels of abstraction; and in [19], the authors decompose requirements to identify security-centric requirements.

All these examples rather describe decomposition as an activity to devise a specification that describes the system to be built. Within an Agile context, decomposition is used to reduce the risk of providing a constant flow of value; therefore, user stories are typically decomposed following the principle that each one should deliver value to the customer. To the best of our knowledge, no peer-reviewed works exist that describe decomposition techniques used within an Agile context. However, various other techniques worth mentioning have been developed by practitioners, who describe them on their blogs. In the following, we will describe the approaches they propose.

As a general approach, Lawrence [3] suggests two general rules of thumb: choosing a decomposition strategy that allows deprioritizing or throwing away parts of a story, thus isolating and removing unnecessary smaller parts of a larger user story, and then choosing a strategy that results in equally sized small stories. Verwijs [5] distinguishes between two ways to break down user stories: horizontal and vertical. Horizontal breakdown means dividing user stories by the type of work that is needed or the layers or components that are involved, e.g. separating a large user story into smaller user stories for the UI, the database, the server, etc. He suggests avoiding this type of break-down as user stories will no longer represent units of "working, demonstrable software", as it will be hard to ship them separately to the user, as it increases bottlenecks since developers will tend to specialize in types of user stories, e.g., the "database guy", and as it is hard to prioritize horizontally divided stories. Verwijs suggests breaking down user stories "vertically", i.e., "in such a way that smaller items still result in working, demonstrable, software [5]." Recent works also support Verwijs proposal, suggesting to decompose the user stories incrementally, starting from the minimum viable product [16] and decomposing each functionality vertically, so as to also improve the user stories

effort estimation accuracy [7, 15] and the testing easiness [20]. However, this process is more suitable for projects started from scratch with SCRUM instead of project where SCRUM has been introduced later [14].

As these are specific techniques for decomposing a large user story into smaller ones in an Agile context, we integrated their proposals into the following list:

1. Input options/platform [5]: decompose user stories based on the different UI possibilities, e.g., command line input or a graphical user interface;
2. Study conjunctions and connecting words (like "and") to separate stories [4];
3. Data types or parameters [3, 5]: user stories are split based on the datatypes they return or the parameters they are supposed to handle; for example, during a search process, one could define different user stories for the different search parameters the user is allowed to define;
4. Operations, e.g. CRUD [3, 5]: whenever user stories involve a set of operations, such as CRUD (create, read, update, delete), they are separated into smaller versions implementing each operation separately;
5. Simple/Complex [3, 5]: a complex user story is decomposed into a simple, default variation and additional variations that describe special cases or additional aspects;
6. Major effort [3, 5]: a complex user story is decomposed into smaller ones isolating the difficulty in one user story;
7. Workflow steps [3–5, 8]: the temporal development of the user story is studied by imagining the process that the typical user has to follow to accomplish the user story in order to develop (smaller) step-by-step user stories;
8. Test scenarios/test case [4, 5]: user stories are divided based on the way they will be tested. If they will be tested by first executing a sequence of steps and then executing another sequence of steps, these two groups of steps will be implemented as two separate user stories;
9. Roles [5]: one functionality is formulated as separate user stories describing the same story for different user roles (personas) in each user story;
10. Business rules [3, 5]: user stories are extended by "business rules", i.e., constraints or rules that are defined by the context in which the system has to be developed, e.g., a specific law that has to be fulfilled, and the single constraints and rules are used to formulate more fine-grained user stories;
11. Happy/unhappy flow [5]: separate user stories are created for successful variations and unsuccessful variations of the user story;
12. Browser compatibility [5]: if there is a large effort connected to particular technologies, e.g., a text-based browser, [5] recommends splitting user stories according to browser compatibility. Having separate user stories for different browsers allows the product owner to prioritize the work;
13. Identified acceptance criteria [4, 5]: acceptance criteria are defined for user stories that can be used to develop a refined set of (smaller) user stories;
14. External dependencies [5]: user stories can be separated based on the external systems to which they have access;
15. Usability requirements [5]: user stories are separated based on particular usability requirements, e.g., particular implementations for color-blind users;

16. SEO requirements [5]: user stories are separated based on search-engine-optimization requirements, e.g., separate landing pages for specific keywords;
17. Break out a Spike [3]: a user story that is not well understood is divided into one that develops a prototype, a so-called "spike", and one that implements the actual functionality; and
18. Refinement of generic words (like "manage") into more concrete user stories [4].

3 The Multiple Case Study

As stated in the introduction, the objective of this research is to compare the requirements gathering processes and user story decomposition in Agile and unstructured development processes. Therefore, we designed this study as a multiple case study with a replication design [1].

As depicted in Fig. 1, we first identified the research questions, then selected the case studies and their design.

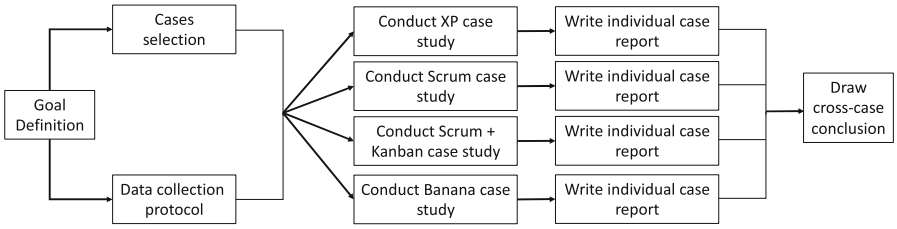

Fig. 1. Study design (adapted from [1])

In this section, we present the study process we adopted, the goal, the research questions, and the metrics for the case study. This is followed by a description of the designs used for the case study, the measurement instruments, and the results obtained.

3.1 Study Goal

According to our research objective, we formulated the goal of the case study following the GQM approach [2] as follows:

Analyze requirements decomposition in user stories (or tasks)
For the purpose of comparison
With respect to the granularity
From the point of view of software developers
In the context of Scrum, Scrum with Kanban, XP, and an ad-hoc development process

Note that in the case of Agile processes, we refer to user stories, whereas in the case of the ad-hoc development process, we refer to tasks. This leads to the following research question:

RQ1: How does the requirements decomposition of user stories differ between the four considered development processes?

For this research question, we defined six metrics:

M1: Number of requirements: the number of requirements provided by the product owner;
M2: Number of user stories: the number of decomposed user stories;
M3: Number of tasks/user stories per requirement: describes how each requirement was decomposed in each team for each requirement;
M4: Total effort (actual development time): time spent (hours) to develop the whole application;
M5: Total effort for each requirement: time spent (hours) to implement requirements among the different teams;
M6: Total effort per task/user story: time spent (hours) to implement each task/user story; and
M7: Strategy used to decompose requirements into tasks or user stories: strategy described in the literature used to decompose each requirement, assigned to two researchers of this paper studying the names of the decomposed requirements.

3.2 Study Design

We designed our study as a multiple-case replication design [1]. We asked four development teams to develop the same application. The four sub-case studies are sufficient as replications since the teams have similar backgrounds. All the teams received the same requirements provided by the same entrepreneur, who acted as product owner and within the same timeframe.

One team was required to develop the project in Scrum, one in Scrum with Kanban, another one using XP, and the last one was free to develop using an ad-hoc process, as shown in Fig. 2. We call the ad-hoc development process the "Banana" process, since this term is used among practitioners to describe processes that produce immature products, which have to "ripen" after being shipped to the customer, like bananas.

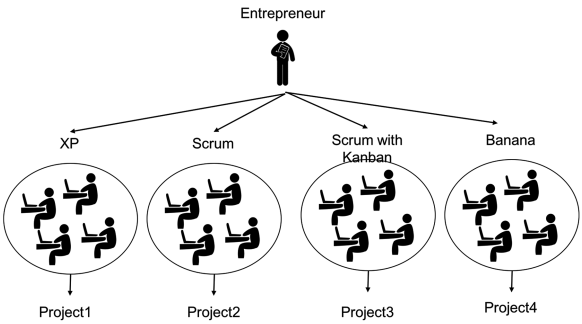

Fig. 2. Study design

As the population of our study, we selected four groups of master students in computer science from two universities involved in the Software Factory Network [6]. One group was from the Master in Computer Science curriculum of the University of Bolzano-Bozen (Italy) and the other four groups were from the Master in Information Processing Science curriculum of the University of Oulu (Finland). The groups had very similar backgrounds since they were all master students in computer science and both universities have similar programs due to their participation in the European Master in Software Engineering (EMSE, http://em-se.eu/) program and having taken classes on agile software development and software engineering. International students took part in this project, originating from Finland, India, Nepal, China, Russia, Bangladesh, Germany, Italy, and Ghana.

The students were randomly assigned to each group taking into account that each team needed to have at least one experienced developer.

The groups were asked to develop the same application. The application requirements were proposed by the product owner, a designer from Bolzano with no experience in software development who described the requirements to the groups with the same schedule and using the same terminology.

The developers elicited the requirements, translating them from the "designer language", a non-technical language, to a more technical one. The groups working with Scrum (with and without Kanban) and XP decomposed the requirements into user stories, while the group using "Banana" decomposed them into tasks.

The developed project. The teams were required to develop an Android application called Serendipity. The idea was selected in a contest for entrepreneurs, where entrepreneurs were asked to submit the minimum viable product [16] description of their project ideas that could be implemented in the software factory lab (http://ideas.inf.unibz.it/). Serendipity is an Android application and a web application intended to share a set of sounds in a specific location, so as to have the user recall special moments by listening to the sounds.

The entrepreneur, a designer from Bolzano, initially defined the project idea as: "Serendipity means "fortunate happenstance" or "pleasant surprise". This project is meant to be an experience that mixes places and sound to enable you to see places you usually go to with new eyes, in a more poetic, more ecstatic way. While taking walk, you will have access to six music tracks, developed from the actual ambient sound of those places themselves. I specifically chose very popular meeting points in my town (Bolzano), where many people go without even realizing anymore what the place looks like. On a map displayed on your smartphone, these locations are highlighted. When you arrive there, you can listen to the soundtrack created to allow you to enjoy the moment. It should be a discovery process. The perk is that this concept is applicable to any city/place – it would be nice to spread it and let the sound go local".

The entrepreneur acted as product owner and described the project to the groups, which elicited the requirements (Req) independently. The requirements were intentionally stated such as to allow vertical break-down [5] decomposition and were proposed to the groups within this timeframe:

Week #0:

Req 1: Minimal Viable Product, with all pages with fake content. The parts of the product comprised: Sign-in/Login; Maps; Listen to sound; Record sound; Rules; and About.
Req 2: Show the list of available sounds on a map.
Req 3: Allow only registered users to record tracks.
Req 4: The main sound can only be played when the user is exactly in the correct location.

Week #3:

Req 5: No more than three sounds allowed within a radius of 300 m.
Req 6: Sounds cannot be downloaded but only played.
Req 7: Any user (registered or not) can listen to sounds.
Req 8: Users are allowed to listen to an ambient sound within a radius of 300 m from the main sound.

Week #5:

Req 9: Play a notification when entering the notification area, so as to alert the user to a sound in the neighborhood.
Req 10: Due to the lack of accuracy of GPS signals in smartphones, the main sound must to be playable within a radius of 10 m instead of only at the exact point, as previously required in Req 6.

Week #7:

Req 11: Create a "liking" system for each sound, allowing users to "like" a maximum of one sound per spot. In this way, sounds with a lower number of likes can be replaced by new sounds after three weeks.
Req 12: Create a web application to allow users to login to their profile with the only purpose of uploading sounds, especially for professional users who would like to upload high-quality or edited sounds.
Req 13: Allow users to register with their Facebook account.

The teams in Oulu that started the development in February were asked to develop the same tool with the same requirements proposed with the same schedule. To ensure the correct succession of requirements and to prevent the development of the previous project in Bolzano to influence the entrepreneur's perception of her project, we recorded every requirement elicited in Bolzano so as to ask her to request the same things without revealing any details to the other teams.

3.3 Study Execution

The web application was developed at the Software Factory Lab of the two participating universities. The participants were initially informed about the study and about the usage of the collected data. The development took place at the University of Bolzano-Bozen (Italy) from October 2015 until the end of January 2016 and at the University of Oulu

from February 2016 to the end of April 2016. The groups were required to spend a minimum effort of 250 h on their work.

Three groups were composed of second-year master students in computer science at the University of Oulu (Finland), while one group was composed of second-year master students in computer science from the University of Bolzano-Bozen. The selected students represent typical developers entering the market. It is therefore interesting not only to understand how they break down requirements but also to observe their work processes. All of the teams had iterations lasting two weeks. The Banana team also met the entrepreneur every two weeks in order to be updated on the requirements.

The first group (Kanban, https://github.com/Belka1000867/Serendipity) was composed of five master students who developed in Scrum with Kanban. The second group (Scrum, https://github.com/samukarjalainen/serendipity-app and https://github.com/-samukarjalainen/serendipity-web) was composed of five master students who developed in Scrum with 2-week sprints, while the third group (XP, https://github.com/davidetaibi/unibz-serendipity) was composed of four master students who developed in Extreme Programming (XP). The fourth group (Banana, https://github.com/Silvergrail/Serendipity/releases) was composed of six master students who developed in an unstructured process, which we defined as "Banana" process.

3.4 Data Collection and Analysis

The measures were collected during meetings with the developers. They also used the collected data to draw burn-down charts and track results. We defined a set of measures to be collected as follows:

- number of sprints;
- opening and closing date for each user story;
- user story description;
- responsible developer for each user story; and
- the actual effort for each user story.

The requirements were elicited from the entrepreneur. However, to avoid interference with the development process, two researchers attended the requirements elicitation meetings and reported the requirements independently.

Three sets of decisions were used to measure pairwise interrater reliability in order to get a fair/good agreement on the first process iteration. In order to resolve any differences, where necessary, we discussed any incongruity to get 100% coverage among the authors.

We associated user stories/tasks with each requirement defined by the entrepreneur. Then we calculated sums, medians, and averages.

4 Study Results

The teams developed the project according to the assigned development process. The XP team developed with a test-first approach, while the two Scrum teams (Scrum and

Scrum with Kanban) developed test cases during the process. The Banana team developed a limited set of test cases at the end of the process. All four teams delivered a final product with the same set of features, with no requirement missing.

The three Agile teams delivered the first version of the product with a limited set of features that could be evaluated by the customer after two sprints, while the Banana team delivered the application, with nearly all the features implemented, only three weeks before the end of the development and then started to implement tests. This result was expected because of the structure of the process, since they decomposed the requirements by means of a horizontal break-down. For example, they developed the whole server-side application first, starting from the design of the database schema, and then the Android application connecting the frontend with the server-side functionalities.

The three Agile teams decomposed the requirements by means of a vertical breakdown [5], so as to deliver to the entrepreneur a working product with the required features as soon as possible. For example, Req 2 (Show the list of available sounds on a map) was decomposed by the XP team into: "Show a Google map centered on the user location in the Android app" and "Show existing sounds as map placeholders", while the Scrum team and the Scrum with Kanban team decomposed this into: "Show a Google map", "Centered on the user location in the Android app," and "Show existing sounds as map placeholders."

As expected, the groups decomposed the 13 requirements into different subsets of user stories/tasks. As reported in Table 1 and Fig. 3, the team working in XP is the one that decomposed the requirements with the lowest granularity (46 user stories), followed by the team using Scrum with Kanban (40 user stories) and the team using Scrum (27 stories). However, the team using the Banana approach decomposed the requirements into only 13 tasks. Moreover, they merged two requirements into one single task. Considering the number of decomposed user stories or tasks per requirement, the results are obviously similar to the total number of user stories and tasks reported.

Table 1. Summary of metrics results

Metrics	XP	Scrum	Scrum+Kanban	Banana
M1 (# of requirements)	13	13	13	13
M2 (# of user stories/tasks)	46	27	40	19
M3 (user stories per requirements)	3.54	2.08	3.08	1.46
M5 (effort per requirement)	23.04	36.77	24.46	48.85
M6 (effort per user story/task)	6.51	17.70	7.95	33.42
Total effort all user stories/tasks	299.5	478	318	635
Other effort	92	10	0	481
M4 (total effort entire project)	391.5	488	318	1116

Taking into account the required effort, the team developing with Scrum with Kanban was the most efficient one, spending a total of 318 h on development. The XP and Scrum teams followed with an effort of 391 h for XP and 478 h for Scrum. The Banana team, unexpectedly, spent dramatically higher effort (1116 h), nearly 3.5 times more than the teams developing with Scrum and Kanban. Considering the effort spent on other tasks not related to user stories, such as database design, server setup, and such, the team using Scrum with Kanban was also the most efficient one, spending no effort on these tasks.

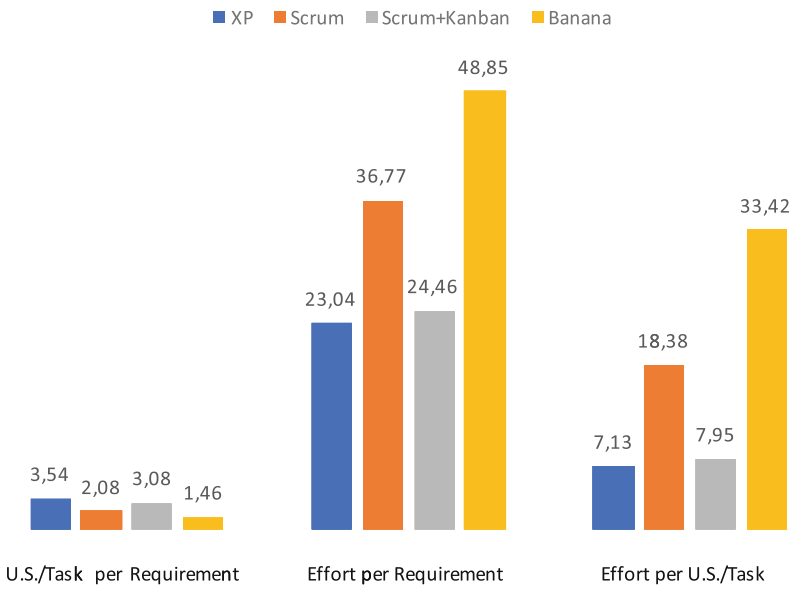

Fig. 3. Comparison of user stories and task decomposition and effort (hours)

The Scrum team only spent 10 h on other activities (2%), the XP team spent 92 h (23%), and the Banana team 481 h (43%).

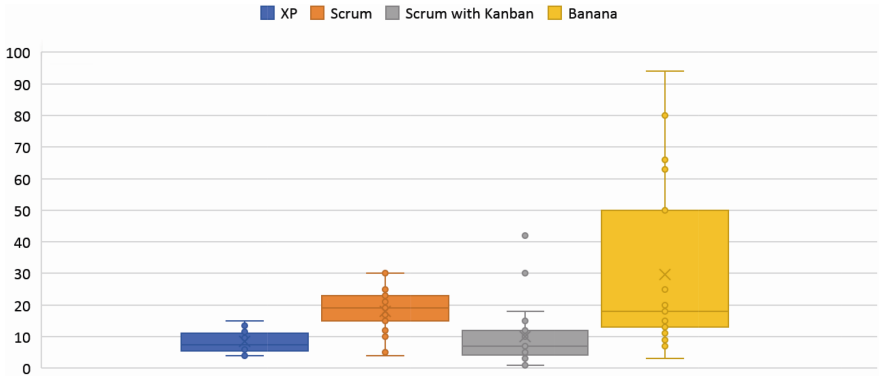

Fig. 4. Boxplot of the effort spent per user story/task

When analyzing the average effort spent to implement each requirement, the teams developing with XP and Scrum with Kanban obtained similar results, while the Scrum and the Banana teams spent similar amounts of effort per requirement, nearly 2.5 times more than the XP and Scrum with Kanban teams. Taking into account the distribution of effort depicted in Fig. 4, there is a similar distribution of effort spent on user stories

between the Agile teams, while, as expected, the Banana team had the highest variability of effort. Looking at the decomposition for each task (Table 2), other differences among the groups emerge. Req 6 was not implemented by all the teams since it was related to "not implementing" the download sound feature. The Banana team also considered zero effort for Req 7 since they merged the tasks with the activities related to Req 4.

Table 2. Effort and user stories/tasks per requirement

Requirement	XP			Scrum			Scrum with Kanban			Banana		
	Effort	# of user stories	Effort/ user story	Effort	# of user stories	Effort/ user story	Effort	# of user stories	Effort/ user story	Effort	# of tasks	Effort/ task
R1	40.5	6	6.8	81	3	27.0	77	5	15.4	140	5	28
R2	20.5	2	10.3	47	3	15.7	17	3	5.7	50	1	50
R3	94	11	8.5	57	3	19.0	112	9	12.4	150	3	50
R4	66.5	7	9.5	46	2	23.0	27	4	6.8	94	1	94
R5	9	2	4.5	16	1	16.0	5	1	5.0	19	1	19
R6												
R7	4	1	4.0	6	1	6.0	5	1	5.0			
R8	13	1	13.0	10	1	10.0	1	1	1.0	18	1	18
R9	17	2	8.5	12	2	6.0	15	1	15.0	25	1	25
R10	10.5	1	10.5	2	1	2.0	1	1	1.0	13	1	13
R11	10	1	10.0	21	1	21.0	2	4	0.5	21	2	10.5
R12	7	1	7.0	165	8	20.6	44	9	4.9	87	2	43.5
R13	7.5	1	7.5	15	2	7.5	12	1	12.0	18	1	18

Table 3 illustrates the various methods applied by the various teams to break down the requirements into user stories (or tasks for the Banana approach), i.e., the results of collecting metric M7. To obtain this table, two researchers studied the user stories and tasks provided by the teams and compared the approach adopted to break down the requirements with the approaches described in the literature. All disagreements in the classification were discussed and clarified based on the description of the broken-down user stories or tasks as well as the description of the approaches found in the literature.

To also be able to classify approaches not recommended in an Agile project, we added the three horizontal break-down strategies described by Verwijs [5]: divide user stories by (1) the type of work that is needed, (2) the layers that are involved, or (3) the components that are involved.

All teams used the approaches "Input options/platform" and "Conjunctions and connecting words". All Agile teams used the approaches "Data types or parameters", "Operations", and "Simple/Complex". Only the Banana team adopted the "Workflow steps" approach and only the XP team adopted the approaches "Test scenarios/test case" and "Roles". The approach "Major effort" was used by the teams XP, Scrum with Kanban, and Banana.

Unexpectedly, the Banana team was not the only one that adopted horizontal break-down approaches such as dividing user stories or tasks based on the layers of the solution, types of work, or components. Typically, Agile teams avoid such types of break-down since this contradicts with the principle that a user story should provide value to the user. We conjecture that the frequent application of horizontal break-down approaches by the

Scrum team was the reason for their bad performance in terms of total effort, compared to the other Agile teams. This also shows that the experiment was conducted with university students with little experience in the field. Nevertheless, their behavior is comparable to professionals at the beginning of their careers. We did not involve freshmen students in the study, as recommended by [10].

Table 3. Requirement decomposition strategies adopted by the studied teams

Strategy	XP	Scrum	Scrum with Kanban	Banana
Vertical decomposition strategies				
Input options/platform [5]	×	×	×	×
Conjunctions and connecting words [4]	×	×	×	×
Data types or parameters [3, 5]	×	×	×	
Operations e.g. CRUD [3, 5]	×	×	×	
Simple/Complex [3, 5]	×	×	×	
Major effort [3, 5]	×		×	×
Workflow steps [3–5, 8]				×
Test scenarios/test case [4, 5]	×			
Roles [5]	×			
Business rules [3, 5]				
Happy/unhappy flow [5]				
Browser compatibility [5]				
Identified acceptance criteria [4, 5]				
External dependencies [5]				
Usability requirements [5]				
SEO requirements [5]				
Break out a Spike [3]				
Refinement of generic words [4]				
Horizontal decomposition strategies				
Layers, e.g. database, GUI [5]	×	×		×
Type of work, e.g. testing, coding [5]		×		×
Components, e.g. server, client [5]		×		×

5 Threats to Validity

Concerning the internal validity of the study, even though we did our best to select developers with a similar background, the results could be partially dependent on the subjects. A replication study could confirm or reject our findings. Concerning the external validity of the study, the use of students to investigate aspects of practitioners is still being debated but considered very close to the results of real practitioners in the case of master students [9] and when one is interested in evaluating the use of a technique by novices or non-expert software engineers [10–13, 17].

6 Conclusion and Future Work

In this work, we conducted a preliminary multiple case study with a replication design with the aim of comparing the requirements decomposition process of an ad-hoc process and Extreme Programming, Scrum, and Scrum with Kanban.

With this study, we contribute to the body of knowledge by providing the first empirical study on requirements decomposition in the Agile domain.

To achieve this purpose, we first provided an overview of the different requirements decomposition techniques and then a description of the study we executed.

Although some results might depend on the participants' skills, we observed the usage of different decomposition techniques in our groups, which often adopted traditional decomposition techniques, which are more suitable for waterfall processes, in combination with other Agile techniques.

The teams developing with Scrum with Kanban and with XP decomposed the requirements into the highest number of user stories, while the team working with an unstructured process, as expected, decomposed the requirements into a very limited number of tasks. Two decomposition approaches were adopted by all processes, namely "Input options/platform" and "Conjunctions and connecting words". All Agile teams used the "Data types or parameters", "Operations", and "Simple/Complex" approaches, while, as expected, only the Banana team adopted the "Workflow steps" approach and only the XP team adopted the approaches "Test scenarios/test case" and "Roles".

Unexpectedly, the Banana team was not the only one that adopted horizontal break-down approaches such as dividing user stories or tasks based on the layers of the solution, types of work, or components. We suppose that the bad performance in terms of total effort of the Scrum team compared to the other Agile teams was probably due to the application of horizontal break-down approaches.

The main result of this work is that requirements decomposition is not only team-dependent but also process-dependent, and that therefore decomposition techniques need to be addressed to a greater extent in order to improve the efficiency of the development process.

Therefore, we recommend that developers investigate requirement break-down approaches more thoroughly and that researchers study the impact of different approaches, so as to identify the most effective ones in different contexts.

In the future, we plan to validate the results obtained with studies involving more students and practitioners and using larger projects.

References

1. Yin, R.K.: Case Study Research: Design and Methods, 4th edn. Sage, Thousand Oaks (2009)
2. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In: Encyclopedia of Software Engineering (1994)

3. Lawrence, R.: Patterns for splitting user stories. Agile For All Blog, 28 October 2009. http://Agileforall.com/patterns-for-splitting-user-stories/. Accessed 8 Dec 2016
4. Irwin, B.: Boulders to gravel: techniques for decomposing user stories. VersionOne Blog, 9 May 2014. https://blog.versionone.com/boulders-to-gravel-techniques-for-decomposing-user-stories/. Accessed 8 Dec 2016
5. Verwijs, C.: 10 useful strategies for breaking down large User Stories (and a cheatsheet). Agilistic Blog. n.d. http://blog.agilistic.nl/10-useful-strategies-for-breaking-down-large-user-stories-and-a-cheatsheet/. Accessed 8 Dec 2016
6. Taibi, D., Lenarduzzi, V., Ahmad, O.M., Liukkunen, K., Lunesu, I., Matta, M., Fagerholm, F., Münch, J., Pietinen, S., Tukiainen, M., Fernández-Sánchez, C., Garbajosa, J., Systä, K.: Free innovation environments: lessons learned from the software factory initiatives. In: The Tenth International Conference on Software Engineering Advances, ICSEA 2015 (2015)
7. Lenarduzzi, V., Lunesu, I., Matta, M., Taibi, D.: Functional size measures and effort estimation in agile development: a replicated study. In: 16th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP2015 (2015)
8. Rubin, K.S.: Essential Scrum: A Practical Guide to the Most Popular Agile Process, 1st edn. Addison-Wesley Professional, Boston (2012)
9. Patton, J., Economy, P.: User Story Mapping: Discover the Whole Story, Build the Right Product, 1st edn. O'Reilly Media, Inc., Sebastopol (2014)
10. Runeson, P.: Using students as experiment subjects – an analysis on graduate and freshmen student data. In: Proceedings 7th International Conference on Empirical Assessment & Evaluation in Software Engineering (2003)
11. Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. **28**(8), 721–734 (2002)
12. Tichy, W.F.: Hints for reviewing empirical work in software engineering. Empirical Softw. Eng. **5**(4), 309–312 (2000)
13. Salman, I., Misirli, A.T., Juristo, N.: Are students representatives of professionals in software engineering experiments? In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence (2015)
14. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Applying SCRUM in an OSS development process: an empirical evaluation. In: 11th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP 2010, pp. 147–159 (2010)
15. Diebold, P., Dieudonné, L., Taibi, D.: Process configuration framework tool. In: 39th Euromicro Conference on Software Engineering and Advanced Applications (2014)
16. Taibi, D., Lenarduzzi, V.: MVP explained: a systematic mapping on the definition of minimum viable product. In: 42th Euromicro Conference on Software Engineering and Advanced Applications 2016, Cyprus (2016)
17. Basili, V.R., Shull, F., Lanubile, F.: Building knowledge through families of experiments. IEEE Trans. Softw. Eng. **25**(4), 456–473 (1999)
18. Wang, H., Zhou, S., Yu, Q.: Discovering web services to improve requirements decomposition. In: 2015 IEEE International Conference on Web Services, New York, NY (2015)
19. Abbasipour, M., Sackmann, M., Khendek, F., Toeroe, M.: Ontology-based user requirements decomposition for component selection for highly available systems. In: Proceedings of the 2014 International Conference on Information Reuse and Integration (2014)

20. Morasca, S., Taibi, D., Tosi, D.: OSS-TMM guidelines for improving the testing process of open source software. Int. J. Open Source Softw. Process. **3**(2), 1–22 (2011)
21. Ahmad, M.O., Markkula, J., Oivo, M.: Kanban in software development: a systematic literature review. In: 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 9–16, September 2013