Flexible Requirement Development through User Objectives in an Agile-UCD Hybrid Approach

Begoña Losada
Department of Languages and
Computer Systems
University of the Basque Country
Spain
b.losada@ehu.eus

ABSTRACT

Agile and User-Centered Design (UCD) integration allows the advantages of both approaches to be taken in a hybrid method. The requirements will evolve and gradually adapt to the needs of users and customers, and UCD techniques will improve usability and User eXperience (UX) of the developed software. However, there has been much discussion about the challenges of this integration, because of the differing philosophies.

User stories and use cases are the typical artifacts for specifying requirements in agile methodologies. On the other hand, prototypes, Persona, scenarios, task analysis and usability testing are UCD techniques, frequently employed in Requirements Engineering. Numerous authors have studied the best way for these techniques and artifacts to be compatible in hybrid methods. However, the different approaches of Agile and UCD make it difficult to jointly apply and manage an agile-UCD project.

This article proposes User Objectives for the collection and development of requirements in an agile and user-centered project. The UOs collect the functional and non-functional requirements of the user's wishes. Completing a UO implies the orderly realization of three activities: Specification of Requirements, Presentation and Functionality. This enables, the flexible planning of an order in which to carry out activities, according to agile and UCD criteria. Moreover, the UOs are organized in different categories according to both their development and their relationship with the user. The UOs diagrams enable both the progress of the project to be visualized, thereby facilitating the modularization, prioritization and planning; and also the monitoring of the UOs evaluated with UCD techniques, as well as those that still need to be evaluated.[1]

Interacción 2018,
September 12–14, 2018, Palma, Spain

ACM ISBN 978-1-4503-6491-1/18/09...$15.00
https://doi.org/10.1145/3233824.3233865

CCS CONCEPTS

• **Human-centered computing~User centered design** • *Human-centered computing~HCI design and evaluation methods* • **Software and its engineering~Requirements analysis** • **Software and its engineering~Agile software development**

KEYWORDS

Agile development, user-centered design; human-computer interaction, software engineering,

ACM Reference format:

Begoña Losada, 2018. In *Proceedings of ACM Interacción 2018 conference, Palma, Spain. September 2018 (INTERACCIÓN 2018)*, 8 pages. https://doi.org/10.1145/3233824.3233865

1 INTRODUCTION

Agile and User-Centered Design (UCD) integration has been studied in depth in the last decade, and numerous literature reviews analyze it [7,[11], [24], [27], [45], [46], [47], [50]. These works highlight the contributions that both approaches bring to integration. Agile development is a trend in software process. However, it has gaps in the appropriate product specification to meet user needs and customer expectations [45], [46]. The integration of UCD improves the usability and UX of the developed software, adjusting the product to the user needs.

New hybrid methods have been proposed based on the most used agile methodologies, such as Scrum [44], [48] or XP [10], [18], [23].

In the reviews, the hybrid approach, Agile-UCD, is seen as a good solution, but one that still has to overcome many challenges: lack of time for upfront activities, difficulty of modularization, difficulty of prioritizing UCD activities, synchronizing efforts of UCD practitioners and developers, lack of documentation and also, the neglect of non-functional requirements [20] is pointed.

The biggest difficulty in solving these challenges is the different approaches regarding requirements gathering and upfront designs [47]. On the one hand, agile methodologies deliver pieces of functionality to customers in short iterations and requirement engineering is carried out on these just in time. Agile methods discard intense analysis and design activities at the beginning of

the project, as this would limit quick response to frequent changes in requirements [45].

On the other hand, User-Centered Design expends considerable effort on research and analysis before development begins. This approach is also referred to in the literature as big design up-front (BDUP) [11], and is advocated by some authors in hybrid proposals due to it having a positive effect on the mitigation of design error [21] and on synchronization on very large projects with the involvement of numerous teams [3].

Other authors propose an up-front approach first with an initial and limited vision of the functionalities, a minimal up-front interaction design [33] In this line, Da Silva [47] proposes carrying out this strategy one sprint ahead, that is, to carry out the UCD activities in an iteration prior to development. Sohaib suggests that usability engineering can be adapted to the agile context, by using a more iterative approach and testing throughout the project lifecycle [49]. De Bonte proposes two teams (engineering and design) working in parallel [19].

In our experience, the Agile-UCD integration model can be flexible, and its strategy should not be conditioned from the beginning. Not all agile-UCD projects are equal. All assume the involvement of users and, that there may be changes and these must be managed properly. However, there are projects in which it is possible to have an initial vision in enough detail, which allows global prototypes and collects most of the requirements at the beginning. Many agile projects conduct an upfront planning iteration (iteration zero) to define the overarching product vision and other business requirements for the project [53]. In these cases, a big up-front design or a minimal up-front interaction design approach would be appropriate, because it would help to improve that initial vision, to produce fewer subsequent changes, and to make a first estimate of the work.

In other cases, an agile project is approached with a very limited idea of the functional and non-functional objectives to be met. In these cases, the project usually starts with one or several small functionalities and, in the successive deliveries and meetings, new possibilities of the product are discovered. In this case, a little up-front interaction design or an implicit interaction design approach would be appropriate.

Integration proposals usually suggest the use of classic artifacts which are proposed by most agile methodologies to gather the requirements: use cases and user stories. Curcio presented the results of a systematic study on requirements engineering in an agile software development context [17]. He concludes that requirement engineering in the context of agile software development is still immature and requires more research.

We propose a new artifact to gather the requirements in an Agile-UCD methodology: User Objectives. Its scope, development and typology facilitate a flexible integration according to the needs of each project.

In the following section, Agile and UCD requirements engineering is analyzed, and the most common techniques in the different activities are identified. Then, the usual proposals in the requirements specification are reviewed. Thereafter, we focus on explaining our proposal: The User Objectives as a flexible guide in an agile-UCD process. We will show two development examples to clarify this proposal. Finally, the conclusion summarizes the lessons learned and the contribution of this work.

2 AGILE - UCD REQUIREMENTS ENGINEERING

Brooks said that the hardest part of building a software system is deciding what to build [9]. Therefore, the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements [9].

On the other hand, Sohaib's review identifies that one of the most important challenges in the agile model is how to identify the requirements of a system as accurately as possible from a customer who is not the current end user [49].

Requirements development is subdivided into elicitation, analysis, specification and validation disciplines [1]. In agile methodologies, these activities are not separate phases, but they can be performed in parallel for different requirements of the project [46].

Requirements elicitation encompasses all of the activities involved with discovering requirements, and takes either a usage-centric or a product-centric approach [53]. The usage-centric strategy emphasizes the understanding of user goals in order to derive the necessary system functionality. The product-centric approach focuses on defining features that developers expect will lead to marketplace or business success. Strategies focused on the product often implement features that are not needed and left without implementing other that the user considers necessary [53]. Interviews, brainstorming, ethnography, and prototyping, are the most common usage-centric techniques for this purpose. Also, the Personas technique developed by the human-computer interaction (HCI) discipline is useful for eliciting requirements in a usage-centric manner. A Persona is a description of a representative member of the user class [16], [33]. Personas helps to focus software analysis and design on the features and goals of the product's end user [16].

Requirements analysis involves reaching a richer and more precise understanding of each requirement, decomposing high-level requirements into an appropriate level of detail. Analysis models can reveal incorrect, or missing requirements. Such models include data flow diagrams, entity-relationship diagrams, state-transition diagrams, state tables, dialogue maps, decision trees [3], and others such as tasks analysis [42]. Other, more agile techniques are UI prototypes [2].

Requirements specification involves representing and storing the collected requirements knowledge in a persistent and well-organized fashion. Use case and user story are the most preferred artifact in agile methods [52]. This is not surprising because these two methods are very well known due to their prominent comprehensibility, decomposability and interactivity.

Finally, Requirement validation confirms the requirement for a product. It implies translating the details into acceptance tests and criteria to confirm that a product meets customer needs.

3 PROPOSALS FOR REQUIREMENTS SPECIFICATION

Although there are other proposals that will be named later, we will focus first and in more depth on the use cases and the user stories, as these are more typically employed to collect the Agile and Agile-UCD requirements.

The idea of employing the use cases to describe the functional requirements was introduced in 1987 by Ivar Jacobson [25], one of the main contributors of UML and Unified Proces (UP). A use case is a collection of related success and failure scenarios, which describe the actors using a system to meet an objective. A scenario is a specific sequence of actions and interactions between the actors and the system under study; it is also called use case instance. It is a particular history of the use of a system. The concept of Jacobson's use case has had a great influence and has been widely recognized.

It is important to note that an actor and an end user are not necessarily the same. An actor represents a class of external entities (people frequently, but not always) that has a role in the context of the use case [43].

In the UP model, the use cases are written over several iterations. There is a thorough workshop at the beginning to detect actors and use cases. In the elaboration phase there are multiple iterations and requirements workshops where the use cases are refined. At the end of the elaboration, 80% to 90% of the use cases are written in detail. During the construction, the writing of minor use cases will take place. At this stage, most of the main functional and non-functional requirements should have stabilized in an iterative and adaptable manner [25]

User stories are the most frequent requirements notation in agile projects [5]. On their introduction in 1999, Kent Beck provided no concrete description of what user stories were other than defining them as one thing that the customer wants the system to do [4]. Mike Cohn coined the following definition of user stories: a concise description of functionality that will be valuable to either a user or purchaser of a system or software [14].

It still has to be ensured that these user stories are of sufficient quality. Cohn outlines some guidelines for writing good user stories in 2004 based on INVEST [14]; more recently, frameworks such as QUS (Quality User Story) [5] have been developed.

Both, use cases and user stories, are focused on understanding what different types of users need to accomplish through interactions with a software system. However, the two processes move in different directions. With use cases, the Business Analyst structures the information collected according to a use case template. From the use case specification, the BA can derive the functional requirements that developers must implement, and a tester can identify tests to judge whether the use case was properly implemented [53]. On the other hand, a user story is a concise statement that articulates something a user needs and serves as a starting point for conversations to flesh out the details. Rather than specifying functional requirements, agile teams typically elaborate a refined user story into a set of acceptance tests that collectively describes the story's «condition of satisfaction». The power of both use cases and user stories comes from their user-centric and usage-centric perspective [53].

User stories were created specifically to address the needs of agile developers and offer the advantage of simplicity and conciseness, but they have a drawback with respect to the use cases. Use cases provide project participants with a structure and context that a collection of user stories lacks. Use cases provide an organized way for the analyst to lead elicitation discussions as a starting point for planning and discussion. User stories do not replicate that structure and rigor, so it is easier for the team to miss some acceptance tests.

There are technical benefits to use cases, too. They reveal some of the important domain objects and their responsibilities to each other. Developers using object-oriented design methods can turn use cases into object models such as class and sequence diagrams.

One drawback of the two majority proposals is the omission of non-functional requirements. Most agile practitioners were more concerned with FRs than NFRs [52]. The neglect of non-functional requirements is reflected in the internal quality of the system, i.e. the maintainability and the test capacity, and the external quality as usability [24]. Farid have proposed agile choice cases for NRF modeling [20].

In addition to using use cases and user stories, requirements engineering offers other possibilities, such as data flow diagrams as the basic tool of structured analysis [53], or features [41]. A feature is a grouping of system capabilities that provides value to a user. In the context of an agile Project, features could encompass an individual user story, multiple user stories, an individual epic, or multiple epics.

With the emergence of hybrid Agile-UCD approaches, some proposals from Usability Engineering have appeared which are adapted for this purpose.

Constantine proposes essential use cases, whose writing avoids the details of UI and focuses on the user's real intention [15].

Castro proposes PersonaSE, which combines the Persona technique with the construction of mock-ups and their formalization through use cases [13].

Other authors propose to develop scenarios [19] or to develop prototypes directly [2], [30].

The results of various reviews show that there are difficulties with requirements documentation in agile models, that cause missing, inadequate or incomplete requirements [24], [45], [46]. Also, Blomkvist [6] reports that developers did not read personas, scenarios and effect maps. They recommend that UCD specialists should translate this information into user stories. By contrast, Cajander reports that there are difficulties to describe usability aspects in such a way, and to their mind, usability needs to be addressed on a higher level [12].

As an alternative to the use cases, the user stories and the other proposals, we present the User Objectives, which are explained below.

4 USER OBJECTIVES, THE AGILE-UCD PROCESS GUIDE

In the agile process, a project is performed in an iterative and incremental way, carrying out division strategies of the problem in the form of units of development, and integration strategies to advance in the increment. Along the way there are numerous modifications or extensions on the product obtained. The agile model needs to continually carry out planning and prioritization activities to decide on the work to be developed. In traditional agile methods, requirements are prioritized by customers who focus on business value or on risk [18]. Use cases and user stories are usually a common means of materializing this process, although UCD criteria is not taken into account and they do not allow the incremental process to be easily reflected.

4.1 Definition of User Objective

An UO expresses a user's desire, such as "buy a t-shirt" or "reserve a meeting room in a workplace". Formally, the UOs are labeled with consecutive numbers <i>, a name and a brief description of the desire user, in first-person, according to the following format: {UOi-name: brief description}. For example,

UO15-Buy a product on offer from home: I want to enter from my home on the Web Store, to visually distinguish the products on offer and choose one for purchase.

A User Objective would appear to have a similar definition to a use case or a user story. However, a UO has notable differences in its scope, development and classification.

4.2 Scope

A UO encompasses one or more functional requirements and zero or more non-functional requirements. For example, in a request to buy and sell flats, the user does not only want to "look for an apartment", but also wants: "Using my mobile phone and in a simple and secure way, look for an apartment, ask for the price and compare this price with the price of my apartment". This UO groups simple UOs such as: "find an apartment", "ask the price", "value an apartment", "consult information about credits". The features "using my mobile phone"," in a simple way", "in a secure way" are non-functional requirements associated with these UOs.

4.3 Development of a UO by activities

Once formalized, the UOs undergo three different development activities: Specification of the requirements (A1), Presentation (A2) and Functionality (A3). The goal of each of these activities is:

- A1. Specification of Requirements. This is a specification in terms of the interface by describing the behavior of the user (actions to be performed in the interface) and the System (reactions of the application in the interface). These descriptions are only a part of the functional requirements, since they only indicate the reactions in the interface. That is, they do not include the control reactions of the application, such as "record the purchase" or "save the order in the database", typical of the use cases specifications. In addition, the specification must indicate the behavior of the application on the interface (navigation) in alternative or erroneous situations. The initial specification is completed with some graphic aspects that will help to configure a simple prototype or simple draft. This helps stakeholders to have a first global view of the appearance and behavior of the UO, but without specifying the precise characteristics of the interface and business logic. This activity covers the following objectives: (1) Facilitate the formalization of the capture of requirements, bringing its validation closer to the stakeholders, and (2) Review interface aspects and business logic in early stages with designers and developers. The appropriate techniques to carry out this activity are the graphical models of task analysis, which have tools that facilitate their writing, discussion and evaluation [37], [42]. In some projects, where there are many integrations that involve continuous changes, or with fewer resources, the direct translation in prototypes of low quality, in the style of suggested by Ambler [2], may be adequate and sufficient. Then, this specification is saved in the form of acceptance tests.
- A2. Presentation of a UO, in which the concrete graphic elements, partially collected in activity A1, are fixed. In this activity, a description is assumed on a specific platform, for example, the graphic interface of a mobile. In addition, a concrete interaction technique that supports the abstract interaction described in A1 is indicated, for example, that the selection is made with a radio-type button. The objective of this activity is to obtain high fidelity prototypes or implementation of views in Model-View-Controller architectures. Standards inspection and heuristic evaluations are examples of suitable techniques for evaluating usability in these prototypes [31]. In these evaluation processes it is also interesting to take into account the accessibility evaluations.
- A3. Functionality. This activity is based on the behavior characteristics defined in A1. Specification of the Requirements, and completes the functionality on the result of activity A2. Presentation of the UO, previously evaluated. Its objective is to obtain an increment in the finished product. Its validation is mainly done through executable tests (as is common in Software Engineering), in order to verify the expected output results from the input data. At this point, validating the acceptance tests [29], specified in activity A1, contributes consistency to the activities and consequently verifies that the final result is adjusted to the end user needs. In addition, log analysis and performance measurement, in combination with the cognitive walkthrough and scenarios, are useful for estimating efficiency and effectiveness [51]. In addition, checking the end user satisfaction of the achieved product, especially for Direct UOs or merged UOs that involve one or more Direct UOs, is an important aspect that can be done with questionnaires. The System Usability Scale scale (SUS) [8] has been adopted as a measure of standard usability of the user experience, and consists of ten

statements that users value from a range of five values. Other proposals based on SUS are the Computer Usability Questionnaire (CSUQ) [35] and the Post-Study System Usability Questionnaire (PSSUQ) [36]. Both are very similar questionnaires to evaluate a system at the end of a usability study; CSUQ is administered online and PSSUQ in person. More recently, UMUX (Usability Metric for User Experience) [22] is presented as a more compact alternative, based on the usability definition ISO 9241-11

4.4 Classification: Types of User Objectives

New developments (UOs) can be requested directly by the user or induced by process strategies. These developments go through evaluation and validation processes by stakeholders (including the end user, especially in Direct UOs). The User Objectives can be:

- **UO Direct** or direct desire of the user.
- **UO Indirect**, or internal need of the development process, derived from Direct or Indirect UOs. These needs may cause a division or a fusion of UOs. In case of division it would give rise to several indirect UOs Division and in case of fusion, an Indirect UO Fusion would be obtained.
- **UO Increment**, or increase in functionality associated with a UO Direct or Indirect.
- **UO Incremented**, or expansion of a Direct or Indirect UO with one or more UO Increments.
- **UO Reused**, or UO previously made in the same or in another project.

In the previous classification, two UO typologies can be observed: The first one which organizes the UOs by the source that brought about their creation and the second, by the identity or autonomy of the UO. According to the first, if the source is the user they are Direct UOs, while if it is the developers, they are Indirect UOs. According to the second classification, there are complete and fictitious identity UOs. The identity of the UO is usually complete, that is, it is a self-contained desire or need that could be a deliverable in the sense of agile methodologies. Those that partially express an increment associated with an existing UO, the so-called UOs Increment, are considered fictitious identities. When these Increment UOs are associated with a full identity UO they result in an Incremented UO, also with full identity. Finally, the Reused UOs are those UOs of complete identity, which have been previously defined in the current project or in another project. These types are displayed graphically as shown in Figure 1.

The developments associated with the Direct UOs should be thoroughly taken into account in the evaluations, especially those concerning usability, since those UOs represent the tasks that the end users will perform in the interface. Direct UOs must always be evaluated by end users, while Indirect UOs can be evaluated by experts. In this way, users will participate only at crucial moments, and developments considered internal (Indirect UOs) will be evaluated by members of the development team, experts in the type of evaluation involved.

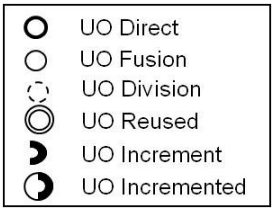

Figure 1. User Objectives Types

5 VISION OF THE METHODOLOGICAL PROCESS

Although the InterMod methodology [38] started out as an agile-UCD proposal for the development of interactive applications through UOs, the UO concept is not limited to this model and its adaptation is possible in any agile process, such as XP, Scrum or Kanban, which habitually employ user stories and do not usually include UCD activities.

The adaptation of the UOs in an agile model is briefly shown in Figure 2-The process begins by analyzing the overall project, by capturing the first user needs or desires. The Personas technique is appropriate for this [16]. At this stage, the general vision of the system and the global aspects of the design are also determined, which will allow a subsequent, iterative and incremental development, with guarantees of coherence.

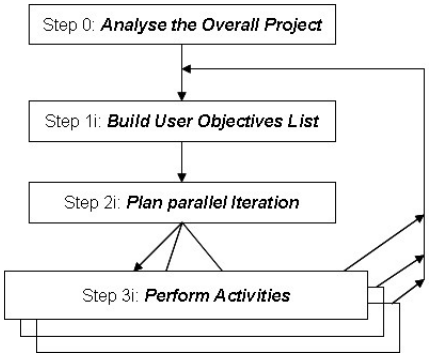

Figure 2. Simplified scheme of the Agile-UCD process

Throughout the process, new user needs are formalized in UOs. A set of planning and development rules helps to make strategy decisions, according to customer and developers interests, and also, to the UCD perspective that establishes a specific order in the realization of the activities. According to the UCD recommendations, once the requirements of a certain UO have been specified and evaluated (A1), its interface must be generated and validated (A2), before carrying out its coding (A3). Unlike UCD, in an agile-UCD methodology, this order is restricted to each individual UO, and not to the global project.

Although traceability is one of the best ways to track the impact of requirements changes Jayatilleke indicates that the cost of traceability can discourage a team as the benefits are not realized immediately [27]. The UOs diagrams is an agile tool that enables both the progress of the project to be visualized step by step, thereby facilitating the modularization, prioritization and planning; and also the monitoring of the UOs evaluated with UCD techniques, as well as those that still need to be evaluated.

Next, two different cases of development and planning of activities, using diagrams of UOs, are shown. In the first, a project with UCD big up-front activities is described. In the second, a project in which the UCD activities of a delivery are performed just in time, before coding and delivering, is explained. Figure 3 shows how the realization of activities A1, A2 and A3 is visualized in the diagrams:

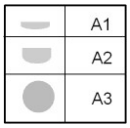

Figure 3. Visualization of development activities in a UO Diagram.

5.1 First case: Up-front vision

Figure 4. shows the development diagram of a project with a fairly clear initial vision,

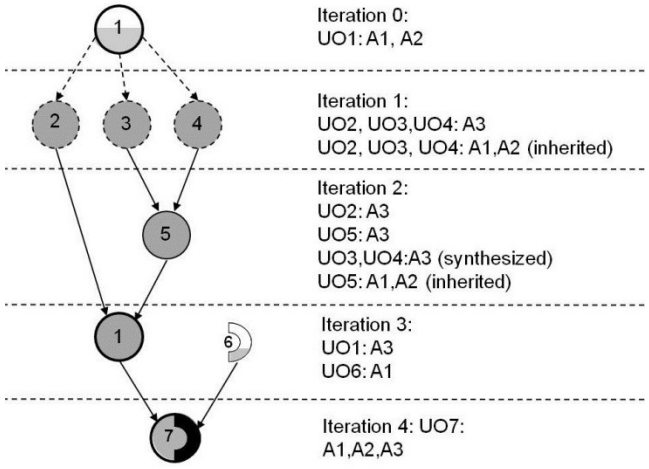

Figure 4. An UO Diagram for an up-front project.

This project starts with a UO1-Direct objective. Its requirements are specified (A1) and a presentation is made, which is evaluated in a prototype (A2). It is an up-front vision of the global project, carried out in iteration 0. In iteration 1, it is decided to divide the complexity of the implementation into UO2, UO3 and UO4, in which the functionality is developed (A3). These UOs do not need to perform activity A1 and A2, since they inherit their validation from UO1. In Iteration 2, activity A3 for UO2 continues to be developed, and an UO5 Fusion is created, which inherits A1 and A2 from UO1 and in which A3 is performed. If there have been changes during the integration activity in UO5, these changes are synthesized by UO3 and UO4.

In iteration 3, the fusion of UO2 and UO5 is carried out, carrying out activity A3. In addition, UO6 begins to be specified, a need to increase functionality in UO1. Finally, in iteration 4, activities A1, A2 and A3 are carried out in the Incremented UO7, from the development of UO1 and the need for the UO6 Increment.

5.2 Second case: Incremental vision, just in time

Figure 5 shows the development diagram of a project, which is characterized by an initial ignorance of the requirements, and in which the scope is discovered step by step.

In this case, the project starts with UO1, an initial objective, small, that is developed completely. Activities A1, A2 and A3 are carried out in order. It is a little up-front vision, carried out in iteration 0.

In the following iterations, the progress of the project is made through small increments on the development made for UO1. In the Increment UOs (for example UO2 and UO3 in iteration 1), only activity A1 is performed and, in the Incremented UOs, activities A1, A2 and A3 are performed. As a result of these activities, activities A2 and A3 of the UOs Increment are considered to have been carried out.

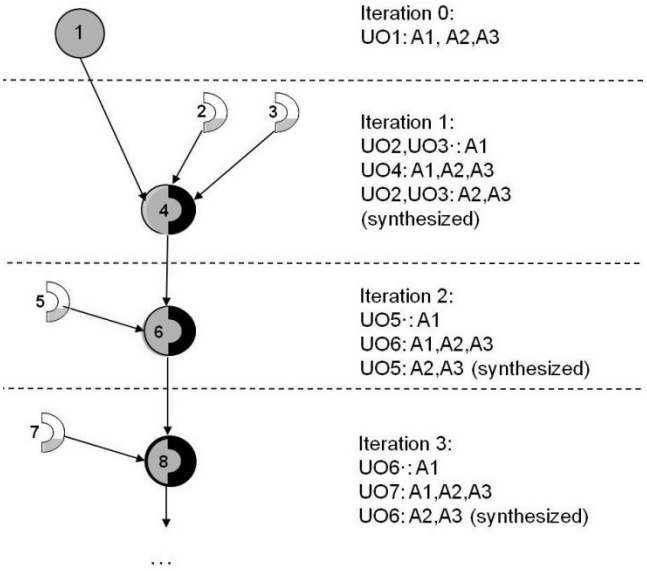

Figure 5. A UO Diagram for a just in time project.

UO8 is an UO Incremented Direct, that is, it represents a task of the end user and therefore, it must be evaluated with end users. On the other hand, UO4 and UO5 are indirect, so their evaluation is recommended, only by experts.

6 CONCLUSIONS

Although the hybrid Agile-UCD approach is a leading proposal, there is no agreement about the various ways and occasions on which UCD's practices should be included in the agile software development lifecycle.

The various techniques, opportunities, ways described in the literature, shows that there are several possibilities for Agile-UCD integration, all of which may be valid or not depending on the context. Here, a flexible agile-UCD integration type is proposed, and to this end, the User Objectives are presented as an alternative to the usual tools to collect the requirements.

In our experience, the Agile-UCD integration model can be flexible, and its strategy should not be conditioned from the beginning. If the project can clearly specify the requirements at the beginning, this is an advantage that can be used to apply UCD techniques to the up-front style. If this is not possible or convenient because there is ignorance in the requirements, they should be collected and updated throughout the project, and will only be coded when there is a high degree of certainty. That is, when they have been validated. Making them one step ahead or in the same iteration, will depend on the workload of the equipment and the development situation.

Completing a UO implies the orderly realization of three activities: Specification of Requirements, Presentation and Functionality. This allows, in a flexible way, the planning of an order in which to carry out activities according to agile and UCD criteria. In addition, the UOs are organized in different categories according to both their development and their relationship with the user. The UOs diagrams allow, on the one hand, the visualization of the progress of the project and they also facilitate modularization, prioritization and planning; and on the other hand, the control of the UOs evaluated with UCD techniques, and those that still need to be evaluated by end users (UOs Directs) or experts (UOs Indirects).

Previous projects have been made with manual drawings that presented the advance by UOs. In order to facilitate this work, we are building a tool that elaborates these diagrams of UOs. We anticipate using the tool in upcoming projects.

ACKNOWLEDGMENTS

This work is partially supported by the Department of Education, Universities and Research of the Basque Government under Grant No.: IT980-16..

REFERENCES

[1] lain Abran, Pierre Bourque, Robert Dupuis, and James W. Moore (Eds.). 2004. Guide to the Software Engineering Body of Knowledge 2004 version. Los Alamitos, CA: IEEE Press.

[2] Scott W. Ambler 2008 Tailoring Usability into Agile Software Development Projects. In: Law E.LC., Hvannberg E.T., Cockton G. (eds) Maturing Usability. Human-Computer Interaction Series. Springer, London

[3] Joy Beatty, Anthony Chen, 2012. Visual Models for Software Requirements, Redmond, WA, Microsoft Press

[4] Kent Beck. 1999. Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA,

[5] Niels Bik, Garm Lucassen and Sjaak Brinkkemper, 2017, A Reference Method for User Story Requirements in Agile Systems Development, 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal, 292-298. doi:10.1109/REW.2017.83

[6] Johan K. Blomkvist, Johan Persson, and Johan Åberg. 2015. Communication through Boundary Objects in Distributed Agile Teams. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15). ACM, New York, NY, USA, 1875-1884. DOI: https://doi.org/10.1145/2702123.2702366

[7] Manuel Brhel, Hendrik Meth, Alexander Maedche, and Karl Werder. 2015. Exploring principles of user-centered agile software development. Inf. Softw. Technol. 61, C (May 2015), 163-181. DOI=http://dx.doi.org/10.1016/j.infsof.2015.01.004

[8] Brooke, J., 1996. SUS: A quick and dirty usability scale. Usability evaluation in industry.

[9] Frederick P. Brooks, Jr.. 1987. No Silver Bullet Essence and Accidents of Software Engineering. Computer 20, 4 (April 1987), 10-19. DOI=10.1109/MC.1987.1663532 http://dx.doi.org/10.1109/MC.1987.1663532

[10] David Broschinsky and Lisa Baker. 2008. Using Persona with XP at LANDesk Software, an Avocent Company. In Proceedings of the Agile 2008 (AGILE '08). IEEE Computer Society, Washington, DC, USA, 543-548. DOI: https://doi.org/10.1109/Agile.2008.91

[11] Leydi Caballero, Ana María Moreno and Ahmed Seffah, 2016. How Agile Developers Integrate User-Centered Design Into Their Processes: A Literature Review. International Journal of Software Engineering and Knowledge Engineering, 26: 1175-1202.

[12] Åsa.Cajander,Marta.Larusdottir and,Jan Gulliksen, -2013. Existing but Not Explicit-The User Perspective in Scrum Projects in Practice, IFIP Conference on Human.Computer Interaction.762–779

[13] John W., Castro, Silvia Teresita Acuña and Natalia Juristo Juzgado. 2008. Enriching Requirements Analysis with the Personas Technique. I-USED 2008.

[14] Mike Cohn, 2004. User Stories Applied: For Agile Software Development. Addison Wesley.

[15] Larry L. Constantine and Lucy A. D. Lockwood. 1999. Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. ACM Press/Addison-Wesley Publ. Co., New York, NY, USA

[16] Alan Cooper, 2004. The Inmates Are Running the Assylum.Why High-Tech Products Drive Us Crazy and How to Restore the Sanity. Indianapolis. IN: Sams Publishing

[17] Karina Curcio, Tiago Navarro, Andreia Malucelli, and Sheila Reinehr. 2018. Requirements engineering. J. Syst. Softw. 139, C (May 2018), 32-50. DOI: https://doi.org/10.1016/j.jss.2018.01.036

[18] Maya Daneva, Egbert Van Der Veen, Chintan Amrit, Smita Ghaisas, Klaas Sikkel, Ramesh Kumar, Nirav Ajmeri, Uday Ramteerthkar, and Roel Wieringa. 2013. Agile requirements prioritization in large-scale outsourced system projects: An empirical study. J. Syst. Softw. 86, 5 (May 2013), 1333-1353. DOI=http://dx.doi.org/10.1016/j.jss.2012.12.046

[19] Austina De Bonte, Drew Fletcher. 2014; Scenario-Focused Engineering: A toolbox for innovation and customer-centricity. Redmond, WA: Microsoft Press

[20] Weam M.Farid, Frank J.Mitropoulos, 2012. NORMATIC: A visual tool for modeling non-functional requirements in agile processes. Conf. Proc - IEEE SOUTHEASTCON. doi:10.1109/SECon.2012.6196989

[21] Jennifer Ferreira, James Noble, Robert Biddle. 2007. Up-Front Interaction Design in Agile Development. In: Concas G., Damiani E., Scotto M., Succi G. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2007. Lecture Notes in Computer Science, vol 4536. Springer, Berlin, Heidelberg

[22] Kraig Finstad., 2010. The Usability Metric for User Experience. Interacting with Computers 22, 323–327.

[23] Andreas Holzinger, Maximilian Errath, Gig Searle, Bettina Thumher and Wolfgang Slany, 2005. From extreme programming and usability engineering to extreme usability in software Engineering education (XP+UE->XU)," Proc, 29th Annual International Computer Software and Applications Conference (COMPSAC'05), IEEE Press, 169-172.

[24] Irum Inayat, Siti Salwah Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. 2015. A systematic literature review on agile requirements engineering practices and challenges. Comput. Hum. Behav. 51, 915-929. DOI: http://dx.doi.org/10.1016/j.chb.2014.10.046

[25] Ivar Jacobson. 1987. Object-oriented development in an industrial environment. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '87), Norman Meyrowitz (Ed.). ACM, New York, NY, USA, 183-191. DOI=http://dx.doi.org/10.1145/38765.38824

[26] Ivar Jacobson, Grady Booch, and James Rumbaugh. 1999. The Unified Software Development Process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[27] Shalinka Jayatilleke and Richard Lai. 2018. A systematic review of requirements change management. Inf. Softw. Technol. 93, C (January 2018), 163-185. DOI: https://doi.org/10.1016/j.infsof.2017.09.004
[28] Gabriela Jurca, Theodore D. Hellmann, and Frank Maurer. 2014. Integrating Agile and User-Centered Design: A Systematic Mapping and Review of Evaluation and Validation Studies of Agile-UX. In Proceedings of the 2014 Agile Conference (AGILE '14). IEEE Computer Society, Washington, DC, USA, 24-32. DOI=http://dx.doi.org/10.1109/AGILE.2014.17
[29] Lasse Koskela. 2008. TEST DRIVEN, Practical TDD and Acceptance TDD for Java Developers. Manning Publications Co.
[30] Thomas Memmel, Harald Reiterer, and Andreas Holzinger. 2007. Agile methods and visual specification in software development: a chance to ensure universal access. In Proceedings of the 4th international conference on Universal access in human computer interaction: coping with diversity (UAHCI'07), Constantine Stephanidis (Ed.). Springer-Verlag, Berlin, Heidelberg, 453-462.
[31] Jacob Nielsen, 1993. Usability Engineering, 1st ed. Morgan Kaufmann
[32] Jason Chong Lee and Scott McCrickard, 2007.Towards Extreme(ly) Usable Software: Exploring Tensions Between Usability and Agile Software Development, Proceedings of the AGILE 2007, .59-71 DOI:10.1109/AGILE.2007.63
[33] Dean Leffingwell,2011. Agile Software Requirements. Addison-Wesley
[34] Cynthia Y. Lester, 2011.Combining agile methods and user-centered design to create a unique user experience: An empirical inquiry, in 4th Int. Conf. Advances in Computer–Human Interactions, 16–21.
[35] James R. Lewis, 2001. Psychometric Evaluation of the CSUQ Using Data from Five Years of Usability Studies (TR 29.3418), IBM Voice Systems. IBM, West Palm Beach, Florida.
[36] James R.Lewis, 2002. Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies. International Journal of Human-Computer Interaction 14, 463–488.
[37] Begoña Losada, Maite Urretavizcaya, and Isabel Fernández De Castro. 2009. Efficient Building of Interactive Applications Guided by Requirements Models. In Proceedings of the 9th International Conference on Web Engineering (ICWE '9), Martin Gaedke, Michael Grossniklaus, and Oscar Díaz (Eds.). Springer-Verlag, Berlin, Heidelberg, 481-484. DOI=10.1007/978-3-642-02818-2_43 http://dx.doi.org/10.1007/978-3-642-02818-2_43
[38] Begoña Losada, Maite Urretavizcaya, Isabel Fernández-Castro, I., 2013. A guide to agile development of interactive software with a "User Objectives"-driven methodology. Science of Computer Programming 78, 2268–2281.
[39] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn Van Der Werf, and Sjaak Brinkkemper. 2016. Improving agile requirements: the Quality User Story framework and tool. Requir. Eng. 21, 3 (September 2016), 383-403. DOI: http://dx.doi.org/10.1007/s00766-016-0250-x
[40] H. Obendorf and M, Finck 2008, Scenario-based usability engineering techniques in agile development processes, Case study in Proc, Conference on Human factors in computing systems (CHI '08), ACM Press, , pp, 2 159-2166,
[41] Steve R. Palmer and Mac Felsing. 2001. A Practical Guide to Feature-Driven Development (1st ed.). Pearson Education.
[42] Fabio Paterno, 2000. Model-Based Design and Evaluation of Interactive Applications, 1st Edition. ed. Springer.
[43] Roger S. Pressman. 2010. Software Engineering: A Practitioner's Approach (7th ed.). McGraw-Hill Higher Education.
[44] Luis A. Rojas and José A. Macías. 2015. An Agile Information-Architecture-Driven Approach for the Development of User-Centered Interactive Software. In Proceedings of the XVI International Conference on Human Computer Interaction (Interacción '15). ACM, New York, NY, USA, Article 50, 8 pages. DOI: https://doi.org/10.1145/2829875.2829919
[45] Dina Salah, Richard F. Paige, and Paul Cairns. 2014. A systematic literature review for agile development processes and user centred design integration. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14). ACM, New York, NY, USA, Art 5, 10 pag. DOI:http://dx.doi.org/10.1145/2601248.2601276
[46] Eva-Maria Schön, Jörg Thomaschewski, María José Escalona. 2017. Agile Requirements Engineering: A systematic literature review. Computer Standards & Interfaces, Vol. 49, 79-91,
[47] Tiago Silva da Silva, Angela Martin, Frank Maurer, and Milene Silveira. 2011. User-Centered Design and Agile Methods: A Systematic Review. In Proceedings of the 2011 Agile Conference (AGILE '11). IEEE Computer Society, Washington, DC, USA, 77-86. DOI=http://dx.doi.org/10.1109/AGILE.2011.24
[48] Mona Singh. 2008. U-SCRUM: An Agile Methodology for Promoting Usability. In Proceedings of the Agile 2008 (AGILE '08). IEEE Computer Society, Washington, DC, USA, 555-560. DOI: https://doi.org/10.1109/Agile.2008.33
[49] Osama Sohaib, and Khalid Khan, 2010. Integrating usability engineering and agile software development: A literature review.2010 International Conference On Computer Design and Applications : V2-32-V2-38.
[50] H.F.Soares, N.S.R.Alves, T.S.Mendes, M.Mendonca, R.O.Spinola,2015. Investigating the Link between User Stories and Documentation Debt on Software Projects, in: 201512th International Conference on Information Technology –New Generations, IEEE: 385–390.
[51] Thomas Tullis and William Albert. 2008. Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[52] Wang, Xinyu & Zhao, L & Wang, Ye & Sun, Jie. (2014). The Role of Requirements Engineering Practices in Agile Development: An Empirical Study. Communications in Computer and Information Science. 432. 195-209. 10.1007/978-3-662-43610-3_15.
[53] Karl Eugene Wiegers, Joy Beatty,. 2014. Software Requirements (3 ed.). Microsoft Press, Redmond, WA, USA