10.20378/irb-58404

Continuous User Experience Development

Kati Kuusinen
Tampere University of Technology
Tampere, Finland
Korkeakoulunkatu 1, FI-33101 Tampere
kati.kuusinen@tut.fi

Abstract. Continuous approaches for software engineering such as continuous planning, development, and operations are becoming increasingly popular in agile software companies. It means that also user-centered design practitioners and practices need to adapt to both possibilities and challenges the increasingly rapid and more tightly integrated software engineering environment induces. Such issues include planning continuously throughout the life cycle instead of upfront planning, delivering user value whenever something is ready instead of delivering working software at the end of time-boxed iterations, and experimenting with real users instead of conducting traditional user studies and tests. In this position paper we discuss how user experience work can be organized with continuous software development.

1 Introduction

Continuous software engineering activities are means to enable rapid development and release cycles in companies. They seek to interweave activities that traditionally have been disconnected in the software lifecycle such as planning, design, and implementation [5]. Continuous software engineering activities often combine automation in software development and deployment processes in a way that minimizes the time required for such practices as integration, verification, deployment and delivery of software [5]. In addition to automated practices, continuous can also refer to practices conducted manually on a steady basis [5]. Fitzgerald et al. [5] define that continuous development consists of the following continuous activities: integration, deployment, delivery, verification and testing, security, and compliance. In addition to plain software development activities, the entire software life cycle can include continuous activities related to planning, operations, and improvement [5].

In relation to agile software development methodologies, the concept of *continuous* can be equated with the concept of *flow* used in Lean development [5], and continuous integration is a practice utilized in Extreme Programming [1]. Bosch [2] claims that continuous integration has no business value if the

organization does not follow agile working practices. Continuous development practices can be utilized with any agile development methodology. However, as there is a constant possibility to deliver whenever something is ready, continuous delivery can diminish the idea of time-boxed iterations utilized, for instance, in Scrum.

In the context of user-centered design (UCD) and user experience (UX) work, continuous software engineering offers benefits such as easier to arrange user experiments. As the development pipeline offers easier deployment of features whenever they are ready for user testing or production, features can be exposed to real usage within a target user group. In case of failure or, for example, after an A/B testing period, features can be withdrawn as easily from the system. Thus, feedback from real users' actual usage can be collected more rapidly and easily than with conventional methods. On the other hand, there is less time for keeping UX design activities ahead of development in continuous software development.

In this paper, we discuss ways to realize UX work in continuous software engineering. We call the approach continuous UX development (CUD). Our focus is in aligning the practices of UX work and software development in a way that forms a coherent interplay between the disciplines allowing UX specialists and software developers to work as a single cross-functional team.

2 Previous Research on Agile UCD

According to a recent systematic literature review [3], current approaches for agile UCD emphasize the role of keeping the UX design work ahead of development, often by iteration or more. In addition, most of the research on agile UCD integration considers it necessary to conduct some design upfront work before starting agile development iterations [3]. Sy [7] has introduced the most commonly referred framework that implements both of those principles. Benefits of such approach include that it offers UX designers time for user studies and design work with less time pressure from the developer side. However, the approach has also its limitations. Despite of its popularity, research conducted in companies utilizing the one iteration ahead approach report challenges in deciding on the scope of design upfront work, in chunking the UX design to suit agile iterations, and in synchronizing the UX design and software development tasks during the iterations [6]. In addition, the approach decreases agility of the project since the team needs to plan the work at least two iterations ahead. Also, since the UX design is created in advance, it might be more prone to changing conditions that can lead to design waste. Moreover, there is some evidence that an approach of UX designers and developers

working as a single cross-functional team might be more efficient than the one iteration ahead approach [4].

3 Continuous UX Development

Since continuous development practices aim at mitigating boundaries between different activities and phases of software life-cycle and at delivering whenever something is ready, we see it necessary to involve the UX specialist in the development team itself. In this setting, UX design work should be conducted in the cross-functional development team on as-needed basis; the one iteration ahead approach is not feasible with continuous development. Next, we describe how we see continuous UX development (CUD).

When a project is initiated, a mini-team starts to work towards a minimum viable working prototype of the software. The mini-team can consist of, for example, one UX specialist and one to two developers. The team together with users works towards understanding the project vision and most critical user paths. This can be done in, for instance, repeated workshops. Between the workshops the UX specialist designs the user flow and drafts user interface that are iterated together with the users in following workshops. Simultaneously, the developer(s) build up the technology stack and the pipeline for continuous development, and experiment with possible technologies they will utilize in the project. In addition, the team, together with the product owner, creates the initial backlog. A minimum viable prototype is an early version of minimum viable product that realizes the design idea. The minimum viable prototype can be, for example, a clickable software prototype with simulated backend that realizes user paths of most important use cases. We believe that a software prototype that allows actual user interaction is the best for communicating the design idea for the user.

After the minimum viable prototype is validated with users, the team size is readjusted to meet the capacity requirements of the project. The team starts to work towards the first production version of the system. The UX specialist either implements the user interaction or pairs up with a front-end developer. The user interaction is built based on continuous communication within the team and together with users when needed. When the team cannot solve a UX issue, developers start to build the next task on the priority list and the UX specialist investigates the problem until a solution is found. The team continues with similar setting during forthcoming iterations.

In addition to working software, we recommend allowing the delivery of working prototype and partially functioning features for gaining user feedback. That means the system can contain both fully working features and

forthcoming features that are delivered for some user groups in order to allow getting early user feedback before launching the feature. This approach is especially beneficial for situations where actual A/B testing is not applicable due to smaller number of users.

The CUD approach we suggest in this position paper offers an alternative to the commonly recommended one iteration ahead approach. Our aim is to offer a way to minimize the required amount of user studies and instead focus on building the user interaction modularly feature by feature. The approach necessitates that the cooperative development team includes a UX specialist. Moreover, it requires a mindset that allows trial and error: when the UX design requires alterations, the user interface will be iterated and refactored. Compared to the one iteration ahead approach, we expect CUD to offer better visibility to the common vision, or the big picture, of the project. Moreover, we expect CUD to improve the team communication and increase developers' commitment towards UX tasks. In addition, we expect to get feedback from actual usage faster than in the one iteration ahead approach. While user studies are valuable, it is the actual use that really validates the viability of the system.

4 Background and Framing of the Position Paper

We base this position paper on our studies of UX development in various software companies. However, we have seen the approach in use as such in none of those companies. Our background is in studying the development of enterprise software and work-related tools. Thus, the approach we present here has been influenced by development of practical-oriented software tools. Therefore, to be transferable to leisure system development, such as game development, the approach may require some changes. First, the focus on tasks directs towards defined user paths. Second, it is usually known in work-related software development who the users will be. Moreover, enterprise software is often built to solve a current business problem of the user organization. Finally, the UX work in enterprise software often focuses on fluent user interaction and practical quality of use whereas focus of leisure systems development can be more in the hedonic. Thus, we believe that UX work of leisure systems should involve more robust methods for evoking users' emotions and hedonic experiences. In addition to leisure systems, considerations related to scalability and special application areas such as safety-critical remain future work.

5 Conclusions

This position paper introduces an alternative to the popular one iteration ahead approach for integrating user-centered design with agile software development methodologies. Moreover, the approach is compatible with continuous software engineering practices. We suggest to involve a UX specialist as a member of the development team, and to conduct UX work in the same iteration together with development practices. To succeed with such approach, the whole team, together with the user, needs to cooperate on daily basis. Furthermore, upfront studying and planning should be minimized and instead trial and error in design and development activities should be allowed.

References

[1] Beck, K. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999. 224p.

[2] Bosch, J. Continuous Software Engineering: An Introduction. In J. Bosch (ed.), Continuous Software Engineering, Springer International Publishing Switzerland, 2014. pp. 3-13.

[3] Brhel, M., Meth, H., Maedche, A., and Werder, C. Exploring principles of user-centered agile software development: A literature review. Information and Software Technology, 61, 2015. pp. 163-181.

[4] Ferreira, J., Sharp, H., Robinson, H. User experience design and agile development: managing cooperation through articulation work. Software: Practice and Experience 41, 9, 2011, John Wiley & Sons. pp. 963-974.

[5] Fitzgerald, B., and Stol, K.-J. Continuous software engineering and beyond: trends and challenges. In Proc. 1st International Workshop on Rapid Continuous Software Engineering – RCoSE 2014. ACM, New York, NY, USA, 2014. pp. 1-9.

[6] Salah, D., Paige, R. and Cairns, P. A systematic literature review on agile development processes and user centred design integration. Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering – EASE'14. ACM, 2014. Article 5, 10 p.

[7] Sy, D. Adapting usability investigations for Agile user-centered design. Journal of Usability Studies 2, 3, 2007. pp. 112–132.