Grace de la Flor, Marina Jirotka (2009):
Blurring the distinction between software design and work practice.
In Christopher Scaffidi, Gunnar Stevens (Eds.),
International Reports on Socio-Informatics (IRSI),
Proceedings of the Work-in-Progress Session of the Second International Symposium on End User Development
(Vol. 6, Iss. 1, pp. 24-31)

Blurring the distinction between software design and work practice

Grace de la Flor and Marina Jirotka
Oxford University Computing LaboratoryWolfson Building, Parks Road, Oxford, UK
grace.de.la.flor@comlab.ox.ac.uk;marina.jirotka@comlab.ox.ac.uk

Abstract. As scientific software is made available within grid infrastructures, EUD increasingly becomes an important activity to support because scientists need to retain a significant amount of control over the code they use to develop experimental workflows and computational models. We present preliminary findings from two case studies where project teams modified the software engineering lifecycle through the implementation of a reconceptualised notion of pair programming as a means in which to facilitate EUD.

Keywords: Empirical Studies, Workplace Studies, Computer Supported Cooperative Work, Requirements Engineering.

Introduction

End-user development has been defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact" [1]. End-users can modify software based upon the computing expertise they may have or the type of modification required; "from customization to component configuration and programming" [2]. In this paper, we present findings from our ongoing empirical studies of end-user development (EUD) practices within e-

Research projects. Specifically, we focus on the use of Agile methods [3], and pair programming in particular, as a means in which to facilitate EUD. Thus, enabling researcher communities to take an active role in the design of scientific software.

e-Research is a revolutionary new approach to distributed research collaboration implemented through large-scale, multi-disciplinary, grid infrastructures that support research efforts in the natural sciences, social sciences, the arts and the humanities. e-Research systems include two key technical artefacts; grid infrastructures which process, transport and store data using distributed high performance computing resources, and software applications that assist and extend the ways in which researchers communicate, collaborate and perform work activities. To achieve the e-Research vision, projects initially focused on developing technical solutions to generic technical requirements such as; the design of large federated databases, data compression and transfer techniques, and security mechanisms in distributed architectures [4]. Even when these technical successes are achieved however, in some cases, e-Research applications have not been adopted by their intended user communities as they challenge the conventions and work practices of researchers [5]. Whilst barriers to uptake and challenges to adoption exist for many reasons, including those of large-scale, distributed project management [6], in this paper we explore the ways in which the user-centred design processes has been extended to meet those challenges.

For this research, we are engaged in ongoing ethnographic fieldwork [7] to understand how software engineering practices employed in e-Research projects might influence technology outcomes. We present two case studies where end-user developers collaborate with software engineers to design e-Research applications. In each case study, pair programming features as a key activity which serves to engage end-users directly in the design and coding of scientific software. Interestingly, these projects have adapted the concept of pair programming, described in Agile and XP, from conducting sessions exclusively amongst pairs of software engineers to working in hybrid pairs made up of domain researchers and software engineers working side-by-side in the production of code.

This reconceptualised notion of pair programming, as a means in which to facilitate EUD, challenges traditional notions of the types of activities that might be included within a software engineering lifecycle. It also challenges traditional notions of the user-centred design process as the degree and level of granularity to which end-users participate in the design process is greatly extended. We present preliminary findings of how this new type of hybrid pair programming may contribute to EUD through two case studies; the Cancer, Heart and Soft Tissue Environment (CHASTE) project, which is developing scientific software for computational biology and the UK Network for Earthquake Engineering

Simulation (UK-NEES) project which is developing a system to link together three UK earthquake engineering laboratories to enable real-time, distributed experiments.

1 Using EUD Practices to Identify Requirements and Design for Usability

In each case study, EUD practices were used to both identify system requirements and design software for usability. A hybrid approach to pair programming has exposed the software development process to account for emerging requirements at the level of code. It has also increased the likelihood of software usability as both the scientist and the software engineer work collaboratively to design technical solutions from the practical standpoint of its actual use within the research setting. This may seem like a risky approach to take when projects are under time constraints and include a diverse and geographically distributed group of stakeholders. On the contrary, it gave software engineers hands-on access to the everyday working practices of scientists which has proven to be far more valuable than designing exclusively through specification documents.

1.1 The CHASTE Project

The Cancer, Heart and Soft Tissue Environment (CHASTE) project, is an e-Research project developing scientific software for computational biology. The system provides researchers access to a grid infrastructure where complex models of biological functions can be processed and visualised. The project specifically wanted to assess how agile methods could be incorporated into the software engineering lifecycle. They achieved this by fostering a close collaboration between heart and cancer modellers, who have expertise in the research domain including identifying appropriate algorithms to include in a biological model, and software engineers, who have expertise in the optimisation of code that would be migrated over to the grid infrastructure. The project has reported that using agile methods has been far more effective in the design of software than plan-driven software development methods because [8]:

- It enables quick integration of new members typical in academic projects
- The quality of code increases through the sharing of different types of expertise
- It encourages rapid code development using short timeframe iterative cycles based upon user stories
- It enables the design of adaptable and extensible code that can be modified as requirements change based upon scientific discoveries in the field

Pair programming has provided a forum in which cancer and heart modellers, with the assistance of software engineers, can develop the code base together. The project has adapted the concept of pair programming so that it could be made useful for dispersed project members through the introduction of the peer review of code. Peer review allows software engineers to comment on the robustness of code initially developed by the cancer and heart modellers. It also provides researchers opportunities to better understand and extend the capabilities of their code for specific research purposes. Perhaps most importantly for scientists, hybrid pair programming has meant that the project has been more responsive to changes within the research domain as both technology and the science progress. The fragment of interaction below provides an example of the different types of expertise required to produce both scientifically meaningful and computationally efficient code.

Figure 1. Referring to a journal article (left image). Consulting with a colleague (centre image). Turning to the code (right image).

In this example different experts are consulted at different times. As the mathematician and software engineer read through a biology journal article clarification was needed in order for them to better understand the mechanics of cell division. In this case, an expert, a co-author of the paper, was onsite and could instantly clarify their query. This demonstrates the speed in which requirements can precisely be identified so that the appropriate software could begin to be developed that afternoon. Having access to different types of expertise as the code is written provides project members with opportunities throughout the day in which to query each other. This has resulted in the design of software that more accurately reflect end-user needs and where common understandings about the system's purpose and functionality is achieved more quickly. If the software engineers were asked to rely solely on written materials such as a specification document or journal article the software may take longer to produce and it may need to be re-coded so that it matches scientists' requirements accurately.

Within computational biology the design of computer models, sharing visualisations and the analysis of large datasets have become central activities that support scientific research [9]. In these circumstances, end-user programming increasingly becomes an important activity to support within the project lifecycle. Currently, scientists retain a significant amount of control over the code that they use for research purposes. This software has, up until now, run on single desktop

machines or local systems. However, as locally produced and routinely modified programs migrate over to distributed grid infrastructures they require a degree of re-coding in order to work efficiently within them. As e-Research projects support this transition, hybrid pair programming has proved effective for both scientists who want to retain a working knowledge of the code and software engineers who are interested in implementing efficiently systems.

1.2 The UK-NEES Project

The UK Network for Earthquake Engineering Simulation (UK-NEES) project is developing a system which will link together three UK earthquake engineering laboratories in the UK to enable real-time, distributed, hybrid[1] earthquake experiments [10]. In this project, civil engineering researchers work closely with software engineers to optimise existing code that will need to run in a distributed architecture and to develop new software that will give users access
to a 'virtual lab'. When a researcher was asked how he is involved in the process of gathering requirements he replied:

> It's difficult, I am the end-user but I'm also probably one of the main people involved in actually setting the whole thing up (UN03a-08).

From the researcher's point of view "setting the whole thing up" includes developing code to meet the project's requirements and designing a usable application that other researchers, not involved in the project, could operate. Earthquake engineering researchers routinely produce code as part of their research process. For example, in order to conduct a hybrid earthquake test an experiment must include both a numerical model of a structure and a computationally produced workflow that will execute and record the experimental procedure. Earthquake experiments tightly couple the computer system with experimental procedure and so domain researchers expect to be able to modify software features for their research purposes.

In the UK-NEES project researchers developed storyboards and incorporated pair programming practices into the software engineering lifecycle. Storyboards have been used to communicate the requirements of researchers through diagrams which represent work processes and experimental workflows. The researcher states that:

> I kinda wrote this [requirements] down for him [software engineer], in a pictorial form similar to the scribbles you saw there. [He] knows what I'm looking for and then [he] will go and hopefully program a portal up like that (UN03b-08).

This practice is similar to the agile methods technique of developing 'user stories' where 'customers' produce short descriptions of how they would like the system to function [3]; the customer in this case being the scientist. Additionally,

[1] Hybrid' refers to the design of an experiment that consists of both a numerical model and a physical specimen.

hybrid pair programming has facilitated close collaboration between domain scientists and software engineers.

> For the critical parts then we both sit together and I say; 'This is what needs to be done' and 'We should kind of program it in this way' ... cause [the software engineer] doesn't have an idea about how the test should work. He doesn't know anything about the civil engineering side of things ... I put my suggestion down and he'd put his suggestion as to how you can maybe improve that and then once we get it into a format that works then [the software engineer] will code it and I'll have a look at the code [and] if it is doing what it's supposed to be doing then it's good and we'll use it. (UN03c-08).

This quote is a good example of how the domain researcher and the software engineer share their expertise with each other to produce code that is both efficient to run in a distributed system and relevant to research purposes. Interestingly, when the domain researcher was asked if he would call this type of collaboration 'pair programming', he indicated that he had never heard of such a practice and that he had no knowledge of 'agile methods'. In this case, agile-like practices emerged naturally from within the project unlike the CHASTE project where project members had a specific interest in evaluating agile methods.

2 Discussion

In both case studies, project teams modified the software engineering lifecycle through the implementation of a reconceptualised notion of pair programming as a means in which to facilitate EUD. The case studies also demonstrate how the user-centred design process was extended so that end-users could contribute the appropriate degree and level of granularity to the design of software. While each project has transformed traditional notions of the ways in which to manage large-scale software development projects, they have also brought to the forefront the requirement that applications must support EUD practices long after these large-scale systems have been embedded into the research communities that use them.

As scientific software is made available within grid infrastructures, EUD increasingly becomes an important activity to support because scientists need to retain a significant amount of control over the code they use to develop experimental workflows and computational models. However, their primary objective is to conduct domain-specific research. In such circumstances, the challenge becomes one in which researchers maintain an appropriate balance between the time they spend coding and conducting research. Introducing hybrid pair programming as means to facilitate EUD may provide a solution to this challenge.

Our preliminary findings are of particular interest to us as requirements engineering researchers as we investigate new approaches to designing and

managing e-Research applications for usability[1]. Usability, has traditionally been defined as a quality that can only be evaluated once a system has been developed and where interface design guidelines can be used to determine measurable characteristics of usable systems [11]. Our findings suggest the contrary, that usability is an emergent property which cannot be located as something that is built into software but rather as something that can only be found in the emergent practices of end-users [cf. 12] as they interact with software artefacts. Hybrid pair programming also provides us with an extended notion of Participatory Design [13] where domain researchers engage as active participants in collaboration with software engineers; working side-by-side in the production of code. Such close collaborations between scientists and software engineers also foster informal communication and cooperation, two factors attributed to bringing about the design of usable software [14].

In future research we intend to conduct a comparative analysis of the organisation of software engineering practices across projects [cf. 15] that use hybrid pair programming as a means in which to facilitate EUD. We will examine how such partnerships increase the usability of software and include analysis of similarities, differences and issues.

3 References

1. Lieberman, H. et al. (eds.) End User Development. Springer, London (2006)
2. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A.G., Mehandjiev, N.: Meta-Design: a Manifesto For End-User Development. Communications of the ACM, Vol. 47(9), pp. 33--37 (2004)
3. Cockburn, A.: Agile software development: the cooperative game, 2nd edition. Addison-Wesley (2007)
4. Hey, T. & Trefethen, A.: e-Science and its Implications. Phil Trans. Royal Soc., 361. pp. 1809--1825 (2003)
5. Bos, N., Zimmerman, A., Olson, J., Yew, J., Yerkie, J., Dahl, E., Olson, G.: From shared databases to communities of practice: A taxonomy of collaboratories. Journal of Computer-Mediated Communication, 12(2) (2007)
6. Lloyd, S. & Simpson, A.C.: Project management in multi-disciplinary collaborative research. In Proc of International Professional Communication Conference (IPCC). Limerick (2005)
7- Randall, D., Harper, R., Rouncefield, M.: Fieldwork for Design. London: Springer-Verlag (2007)
8. Pitt-Francis, J. et al: Chaste: using agile programming techniques to develop computational biology software. Phil. Trans. R. Soc. A,366, pp. 3111--3136 (2008)
9. Carusi, A, Jirotka, M.: Parameters and visions: dataflows in computational and mathematical biology, The Oxford e-Research Conference. 11-13 September 2008. Oxford, UK (2008)
10. Ojaghi, M. et al.: Grid Based Distributed Hybrid Testing. Procs of the UK e-Science All Hands Meeting, Nottingham, UK, pp.190--196. (2007)

[1] Embedding e-Science Applications - Designing and Managing for Usability project. Grant No. EP/D049733/1.

11. Shackel, Brian (ed.): Man-Computer Interaction: Human Factors Aspects of Computers and People. The Netherlands, Sijthoff and Noordhoof Publishers (1981)
12. Zemel, A., et al.: "What are We Missing?" Usability's Indexical Ground. JCSCW, 17. pp. 63--85 (2008)
13. Bødker, K., Kensing, F., Simonsen J.: Participatory IT Design Designing for Business and Workplace Realities. MIT Press, USA (2004)
14. Nørbjerg, J & Kraft, P.: Software practice is social practice. In: Y. Dittrich, C. Floyd and R. Klichewski, (eds), Social Thinking-Software Practice. MIT Press, USA (2002)
15. Button, G and Sharrock, W.: Occasioned practices in the work of software engineers, In: Jirotka, M. & Goguen, J. (eds) Requirements engineering: social and technical issues. pp. 217--240. Academic Press, San Diego, CA (1994)