

Usability/User interface Design in Agile Processes

Anne Fuller
University of Wollongong

School of Information Technology and Computer Science
University of Wollongong
Wollongong, New South Wales
Email: annef@uow.edu.au

Abstract

The increasing popularity of the “agile” software development methods has prompted claims that such methods compromise the usability of the delivered product. However, this need not be the case. While some authors have suggested remedial extensions or additions that run in parallel with the chosen method, we believe that any agile method, properly applied, will produce software equally as useable as that produced by any other method.

The aim of this paper is twofold. Firstly we present a survey of many of the arguments made in the literature, thus bringing together a number of differing concerns. Then we set out to present our arguments debunking these claims.

Keywords: Usability design, user interface design, agile development

INTRODUCTION

The previous decade or so has seen the emergence of many development methods aimed at streamlining the development process. Although a number of so-called “agile” methods also surfaced in this period, the term was not in general use prior to the publication of the Agile Manifesto in 2001 (Highsmith, 2001).

These leaner methods have either abandoned or reduced the importance of many aspects of more traditional development methods. In particular “big upfront design”, where the entire interface for the finished product is traditionally designed prior to implementing any functionality is no longer the favoured approach. This has led to some authors questioning the ability of agile processes to adequately deal with issues such as usability.

The next section provides a brief overview of agile development. This is followed by a summary of the concerns raised in the literature regarding the lack of interface design in agile methods. The subsequent discussion presents arguments aimed at dispelling those concerns.

AGILE DEVELOPMENT

The Manifesto for Agile Software Development introduces the “four values”. The practising software professionals who have signed the manifesto value certain elements of development more highly than others. These are, individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan. This is not meant to imply the items on the right are unimportant, rather it reflects that, when forced to make a choice between each pair of items, the item on the left will be preferred (Fowler and Highsmith, 2001).

The first value, emphasizes that, although tools and processes are important, it is the individuals making up the development team and their relationships have a greater impact (Fowler & Highsmith, 2001, Abramsson et al, 2002, Cockburn, 2002). Thus an “undocumented process with good interactions” is preferable to a “documented process with hostile interactions” (Cockburn, 2002).

The second value reflects the belief that no amount of documentation can be aggregated into a product. Documenting the results of analysis, design etc. may serve as an indication of what might be built, thus may have some value for the development team, but running software is the only deliverable of value to the customer and the only true measure of what has been built (Cockburn, 2002). Therefore each team “needs to determine for itself what documentation is absolutely essential” (Fowler & Highsmith, 2001).

The third value does not advocate the abandonment of contracts. Rather it suggests that contracts alone are insufficient (Fowler and Highsmith, 2001). Continuous collaboration is the only way the team can fully understand their customer's needs (Fowler and Highsmith, 2001) while fostering a good relationship with the customer may save difficult contract situations (Cockburn, 2002).

This final value recognises the volatile nature of today's business environment and that during a project's lifetime the customer's needs are likely to change as a consequence of many external factors. Planning is constructive and provides an overall direction for a project, however the team that cannot, or will not, respond to the customer's changing requirements risks delivering a product of limited value (Fowler & Highsmith, 2001). Thus, while all Agile methods include planning elements, they also recognise it is equally important to be able to adjust those plans as necessary (Abramsson, 2002, Cockburn, 2001).

An agile project starts by building something simple that works. Additional functionality is added in successive iterations, always at the behest of the client, in collaboration with the agile team. Every component is thoroughly tested, and the system is regression tested continuously. Increasingly elaborate versions of the product are released as the project proceeds.

Agile methods include eXtreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), the Crystal family of methodologies, Pragmatic Programming and Feature Driven Development. A representative of each of these methods is a signatory to the Agile Manifesto and among the founding members of the Agile Alliance (Cockburn, 2002). Other methods such as the Rational Unified Process and Open Source Development are also sometimes classified as Agile (Abrahamsson et al, 2002).

INTERFACE AND/OR USABILITY DESIGN IN AGILE METHODS

Hudson (2003) argues that the XP philosophy of producing working code then refactoring to improve design is problematic. He claims there is a lack of general understanding that the interface has a major impact on system architecture. The structure of the user interface must be done "before the system architecture is decided". While acknowledging that interfaces can be refactored, Hudson cautions that users need consistency, and major changes to the interface can result in a number of expensive side effects. He further claims the so-called "user representative" on an XP team is rarely a typical user, instead lists a variety of motivations for choosing a representative, most of which have little to do with his/her familiarity with the needs of the average user.

Similarly, Constantine (2002, is concerned that refactoring or "later refinement" of system architecture necessitates changing the interface that users previously learned. Refactoring that affects the user interface as well as the internal structure of a system "imposes a huge burden on users". Constantine goes on to describe what he terms an "agile incarnation" of usage centered design. However the process described, as he himself subsequently admits, does not include any systematic consideration of interface architecture either.

In fact Constantine's Agile Usage Centred Design sounds remarkably like XP! Users, designers and developers collaborate to produce a set of index cards that inventories roles that users can adopt in the planned system. These role cards are prioritised, and an inventory of tasks to support the roles is created, again using index cards. The task cards are prioritised and organised into co-operating groups. A paper prototype is produced and revised and refined. Finally the interface itself is constructed. Constantine states "... the assumption here is that development proceeds through successive release cycles". Fair enough. Then he continues "successive iterations will pick up additional roles and task cases to guide refinement and expansion of the user interface" and "on each successive iteration ... roles, tasks and co-operating clusters are reviewed and refined as needed before the next set of paper prototypes is produced".

This is just like XP, only applied to designing the user interface. If, for any particular project, the interface is considered a critical indicator of project success, then we would expect that this would have been identified early on, and the interface would have been given the appropriate priority, and its design approached in a very similar manner to that described by Constantine. In addition, recall Fowler and Highsmith's comment regarding the second value of the Agile Manifesto. Each team "needs to determine for itself what documentation is absolutely essential" (Fowler & Highsmith, 2001). Constantine's Agile Usage Centred Design *is* documentation as the interface is not actually implemented. Clearly the agile team should decide how much or how little is required for a given project.

Kane (2003) recommends adding an approach called Discount Usability Engineering to agile methods. Kane also recognises the similarity between DUE and agile development "both in spirit and tools". His

main argument supporting his call for inclusion of DUE appears to be simply that usability is “rarely mentioned explicitly”. In fact a number agile methods do specifically address design of the user interface. DSDM, for example, includes usability prototypes for exploring the user interface (Stapelton, 2003), while Crystal specifically lists UI designer and usage expert as roles in a development team the user interface design as a work product (<http://alastair.cockburn.us/crystal/individuals.html>).

When discussing results of several XP-style projects, Jeff Patton (2002) suggests that, despite producing consistent, easy to understand interfaces, users found “actual business processes ... hard to accomplish and some business processes ...left out completely”. The issue of missing business processes is separate. In XP the customer makes the decisions regarding the functionality to be included, albeit in consultation with the XP team. If essential functionality is missing this suggests either an inappropriate client representation on the XP team or a failure to communicate effectively on the part of the XP team itself. The inappropriateness of XP for projects that are not fully supported by management has ever been advertised.

DISCUSSION

Most of the concerns regarding the usability of software produced using agile methods have been raised by proponents of user centred or usage centered design. The main focus of their wrath has been XP, rather than the other Agile methods, and we have previously described examples of agile methods that do consider usability or interface design. While it may be true that XP does not include any specific mention of usability or user interface design, this does not mean that these issues are ignored. Instead attention to UI design is implicit in all of XP’s practices.

A number of XP project experience reports examine their approach to usability and/or GUI design. For example, Koenig and Cunningham (2003) discuss using XP to develop their XpPlanit tool. The work was split into two parts, the backend processing and the front-end graphical user interface (GUI). Their article discusses some issues writing automated tests for the GUI, however they conclude that they were successful. Similarly Mitchell et al (2001) describe the development of the Reactor5 Server, a tool for business process automation. They show that various XP practices correspond to scenario based design of the human computer interface as described by Carroll (1999). Carroll is cited as stating that “scenarios are stories” and, of course, user stories form the basis of customer requirements in XP.

It must also be remembered that the various books published in the XP series are guidelines to help experienced practitioners adopt XP. None actually describe *the* XP process. As with any methodology, exactly how XP is implemented in the software development process will depend on the project, the client and the development team. Advice is given as to when XP should not be considered: an inexperienced team is one deterrent. An experienced project manager and experienced team members will recognise the importance of interface design. Thus UI will be allowed for in the development process, albeit in a manner that complies with agile and XP principles.

If an XP project, or indeed any other software project, fails to deliver a quality user interface, then the failure is not necessarily with the process. Instead it is in the way the process was implemented. The manager of an XP project may choose to ignore attention to user interface design in just the same way that the manager of a heavyweight process may choose to reduce coding or testing or any other task in an effort to meet a deadline. In both cases problems may result from the management decision. The process should not be blamed for management failure.

CONCLUSION

The rise of in popularity of the agile methods, and their lightweight approach to software development, has prompted some criticism of their ability to deliver “usable” software. This criticism is for the most part unfounded. Many agile methods specifically consider user interface design as part of the process.

Although none of XP’s practices specifically address the UI, its entire approach, user stories, user collaboration, test first strategy, can all be applied to interface and usability design. There is no need to add HCI/Usability extensions to XP, or any other agile method. We should simply trust the team and the team’s management to take the appropriate action when required.

“Adding weight to a methodology is not likely to improve the team’s chances of delivering.”

REFERENCES

- Highsmith Jim (2001) History: The Agile Manifesto, <http://agilemanifesto.org/history.html>
- Abramsson P, Salo O, Ronkainen J & Warsta J (2002) Agile Software Development Methods: Review and Analysis Espoo 2002, VTT Publications 478.

- Cockburn, Alistair (2002) Agile Software Development. Addison Wesley
- Hudson, William (2003) Adopting User-centred design in an Agile Process: A Conversation, www.syntagm.co.uk/design/articles/ucd-xp03.pdf
- Constantine, Larry (2002) Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. Information Age, Aug/Sep, 2002
- Kane, David (2003) Finding a Place for Discount Usability Engineering in Agile Development: Throwing Down the Gauntlet, Proceedings of the Agile Development Conference, 2003, pp 40 – 46
- Stapelton, Jennifer (2003) DSDM: Business Focused Development, Addison Wesley.
- Koenig, Dierk & Cunningham, Graham (2003) Extreme programming(XP) – What is it? www.xpplanit.com/article.pdf
- Mitchell S, Auernheimer, B & Noble D (2001) Scenarios, Tall Tales, and Stories: Extreme Programming the Oak Grove Way. Presented at XP Universe conference, 2001. www.xpuniverse.com/2001/pdfs/XPU04.pdf
- Carroll J. M. (1999) Five reasons for scenario based design. Proceedings of the 32nd Hawaii International Conference on Systems Science, IEEE Computer Society Press

COPYRIGHT

Anne Fuller © 2004. The author assigns to OZCHI and educational and non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The author also grants a non-exclusive licence to OZCHI to publish this document in full in the Conference Papers and Proceedings. Those documents may be published on the World Wide Web, CD-ROM, in printed form, and on mirror sites on the World Wide Web. Any other usage is prohibited without the express permission of the author.