Agile user-centred design

Marc McNeill

Thoughtworks,
9th Floor Berkshire House
168-173 High Holborn
London, WC1V 7AA

Agile methods are becoming increasingly common in application design, with their collaborative customer focus and iterative, test driven approach. Whilst they share many common principles, it is rare for Agile methods to incorporate user-centred design and human factors approaches. Similarly, there are many agile techniques that are well suited to user-centred practices. This paper discusses how the two approaches can be incorporated. It introduces practical techniques such as the use of stories to capture information needs; collaborative planning; visual modelling; rapid, time-boxed iterations; stand-ups and retrospectives. It advocates how using such techniques, useful and usable applications can be developed at greater speed with less business risk.

Introduction

Martin (2002) identifies common fears that are present on many projects; the project will produce the wrong product, the product will be of inferior quality, the project will be late, the team will work excessive hours, commitments will be broken and ultimately the project will be painful for all involved. Processes, constraints and deliverables are added to projects to help mitigate these fears; however they often become an end to themselves, making projects even more cumbersome and likely to fail. In an effort to overcome this project overload, a group of industry experts came together as the Agile Alliance and drafted a manifesto for a new way of developing software (Beck et al 2001). Key to the manifesto were;

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

The intention was not to deny value in the things on the right hand side but to place greater value on the things on the left hand side.

Agile methods are lightweight software development processes that employ short iterative cycles, involve users to establish, prioritise and verify requirements and rely on knowledge within a team rather than documentation (Boeham and Turner 2004). They have developed in reaction to traditional software engineering that is seen as overly bureaucratic and slow. Rather than investing a large amount of time in up-front design, rigidly capturing and documenting requirements, Agile methods are adaptive and people orientated. Whilst change is an inevitable and often painful aspect of IT projects (e.g. scope "creep/ reduction) Agile welcomes it, allowing the project to adapt to changes as and when they happen.

Agile and User-centred design

User-centred design (UCD) shares many of the characteristics of Agile (Table 1), however they are rarely combined and there can be conflict between the two practices (e.g. Nelson 2002). The greatest source of contention is whether UCD processes constitute "big up-front design", an anathema to Agile. Agile practitioners argue that traditionally an inordinate effort goes into the design which will undoubtedly change as the project develops. From a UCD point of view there is an inherent risk in this approach. Focussing upon building discrete functional components to be stitched together as they evolve (rather than considering the application a holistic user experience from the outset) risks delivering a product that is inconsistent and confusing. This almost inevitably results in an inefficient, error prone and ultimately unfulfilling user experience.

Table 1 Similarities between features of Agile methods and User-centred design

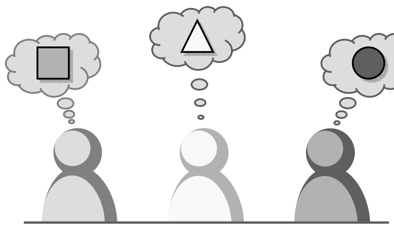
Principle	**Agile**	**User-centred design**
Customer Focus	All activities are focused on providing tangible business value. The customer is typically defined as a representative from the business	All activities are focused on providing (business) value through ensuring a useful, usable and engaging product. The customer is not defined as just the project stakeholders, but the end users as well.
Iterative Development	Early and frequent delivery of working software (often weekly) contributes to project visibility, reduces project risk via regular feedback, fosters continuous improvement and enables early realisation of business benefits.	Develop, test and refine the user interface via regular feedback to the end users. The focus is upon the business risk as well as the technical risk. When lo-fi prototyping with storyboards iterations are typically one to two day cycles.
Test-Driven Development	Testing plays an integral role in every phase of the project life cycle.	User testing plays an integral role in the development of the interaction design.
Collaboration	Collaboration between customers, product managers, analysts, developers, and QA maximises team efficiency.	Even more collaborative with the sharing of ideas and models in addition to stories and code.
Visibility	All stakeholders are provided with maximum visibility into project progress via regular showcases and retrospectives as the project progresses.	More rapid visibility; interaction design is the premier communication tool, defining the outward appearance of what the product will do.

Given the similarities between Agile principles and UCD principles, it is argued that UCD need not be seen as big up-front design, rather a "quick start" to galvanising project success. The overall agile process, that of rapid iterations delivering value to customers is in fact very compatible with the UCD approach. This paper introduces how the approaches can be combined, and how agile techniques can be used to increase UCD input into projects.

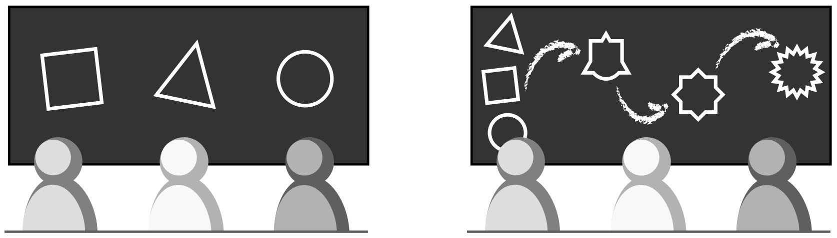
Agile user-centred design processes

Communication

At the centre of Agile user-centred design is facilitated communication. Rather than producing long, wordy documents which are so often produced at the early stages of a project this process instead uses visual techniques that are engaging and allow all stakeholders to give rapid feedback.

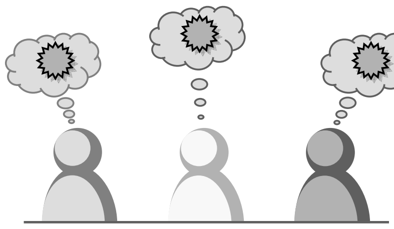


Starting with findings from Contextual Inquiry techniques (Beyer & Holtzblatt 1998), identifying "roles and goals," (how different persona may use the system), process modelling and simple tools such as whiteboards and PowerPoint, an understanding of the issues are elicited and shared with all stakeholders.



Once the issues are clearly articulated, facilitated workshops are run to create solutions. The output of this process may initially be process mapping, but rapidly develop into storyboards; low fidelity prototypes / visual representations of how the GUI may appear.

After several iterations, often shared with a wider user community, a genuinely shared understanding of the problem, solution and approach are gained.



For example, in developing a new account opening application for a bank, storyboards helped refine the proposition. The requirements capture process was significantly shortened; rather than eliciting requirements in a void, a tangible model enabled all stakeholders to see what they might get (and change it accordingly). Perhaps the most powerful result of the storyboards was the ability to place them in front of end users and ask them to complete simple processes in a linear fashion. This user testing rapidly demonstrated that the new process would significantly reduce the time to complete the process by more than 50% providing greater validity to the business case.

Stories

Stories support the storyboards. They are small pieces of discrete functionality appearing in the storyboards, relating them to business value with testable criteria. Thus rather than a requirement for "the application to be easy to use", it must be expressed in criteria that can be tested, such as the user can complete a goal within *n* seconds.

In its most elementary form a story identifies who wants the story, what it needs to do and why it is valuable to have: As a [type of user] I want [some particular feature] so that [some benefit is received]. For example: "As a bank customer I want to view my current account balance so that I know if a recent cheque has cleared."

Collaborative planning

It is a painful reality that not every requirement will make it to the final application. Functionality is stripped out as the project progresses and time frames and budgets get stretched. GUI requirements stated up front may not reflect downstream changes in the project.

In Agile the stories are written on index cards and are physically shuffled according to their priority and business value. This business value is usually driven by the "so that" statement of the story. This prioritisation exercise helps inform the sequence that stories are developed. This is not to say that some low value stories may not be played. Placing the user at the centre of the design may require lower priority stories to be included to enable a coherent user journey. Thinking in terms of feature usage and criticality helps inform this process (Patton 2005). The development team, by estimating the effort required to complete each story, set a cut-off point for the number of stories that can be addressed in that release / iteration. A release strategy is crafted and stories are 'played' in short weekly iterations during the release. The storyboards inform the usability of iteration output, enabling the user interface to be continually evolving, but always focussed upon the end user.

Rapid, time-boxed iterations

Applications are developed in Agile through small, regular incremental iterations, continually testing both the form (i.e. is it delivering on business requirements,) and function (i.e. does the code work). Storyboarding allows the application form to be tested quickly and cheaply, ensuring that the development iteration focuses upon delivering quality code with minimal need for re-work because it does not meet the customer or client expectations.

Showcases

At the end of each iteration there is a showcase where all stakeholders are invited to trial the stories that have been developed. This often includes end users who can validate the usability as it is being developed. If usability is not inherent, stories may not be signed off and require further iterations to get right.

Stand-ups

Every morning the team has a stand-up meeting. These are focussed meetings that communicate daily status, progress, and plans to the team and any observers; identify obstacles more quickly so that the team can take steps to remove them; set focus for the rest of the day and increase team building and socialization (Yipp 2006). Often during stand-ups interface issues will be identified. Rather than leaving the developers to interpret ambiguities in style guides or usability guidelines, the stand-up offers the UCD team member the chance to help work through GUI issues as they come up rather than at the end of the iteration.

Retrospectives
Retrospectives are held anywhere from weekly to monthly to assess how well the team is working with regards to its process. It is an opportunity to take the time to discuss "what has gone well", "what we should do differently" and "what puzzles us" in a structured manner. This is extremely helpful for the team to adjust its process (McKinnon 2006) and provides a voice for the UCD team member that is often lost in projects. Key to Retrospective success is the "safety valve"; attendees anonymously identifying how honest and comfortable they feel with their feedback. For example a scale from "No Problem, I'll talk about anything" to "I'll smile, claim everything is great and agree with managers".

Conclusions

Agile methods are gaining acceptance in IT organisations as an efficient and effective means to developing applications that deliver on the business's requirements. Agile is usually a development-centric philosophy, espousing engagement with the business and using stories and code as the model for communication. User-centred design extends the approach; rather than using code as the model it uses visualisation to articulate the solution. Through collaborative workshops, creating stories and translating them into storyboards and low-fidelity prototypes enables iterations to be showcased on a daily rather than fortnightly basis. Engaging all stakeholders in the process ensures that when the developers start cutting code the focus will be on ensuring code quality, mitigating the risk of business driven changes that could not be articulated without having something tangible to evaluate.

References

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, G., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., 2001 *Manifesto for Agile Software Development.* http://Agilemanifesto.org/

Beyer, H, & Holtzblatt, K (1998) *Contextual Design - Defining Customer Orientated Systems.* (Morgan Kaufman, CA.)

Boeham, B. & Turner, R. 2004, *Balancing Agility and Discipline*. (Addison-Wesley).

Martin, R. C., 2002 *Agile Software Development, Principles, Patterns, and Practices*. (Prentice Hall).

McKinnon, T. (2006) *Retrospective agility*, Objective View, **8**, 10-17 http://www.ratio.co.uk/objectiveview.html

Nelson, N. [2002] Extreme Programming vs. Interaction Design, http://www.fawcette.com/interviews/beck_cooper/default.asp

Patton, J. (2005) *It's All in How You Slice It*, Better software, January 2005 16-40

Yip, J. (2004) *Patterns For Daily Stand-Up Meetings*, http://www.thoughtworks.co.uk/PatternsDailyStandupJason%20Yip.pdf