

A glass of water sits on a cracked, dry, yellowish-brown earth. The background is a vast, flat landscape under a bright blue sky with scattered white clouds. The glass is partially filled with water, and the cracked earth is in sharp focus in the foreground.

GETTING
AGILE with
USER-CENTERED
DESIGN
CREATE SOFTWARE
USERS CAN'T LIVE WITHOUT

by
JON DICKINSON &
DARIUS KUMANA



The agile software development movement has made huge improvements in reliability when delivering software, increasing return on investment, and reducing the risk of building software. However, in a world of iPhones and Google apps, this may no longer be enough.

People are beginning to expect more from software. They expect it to work—not work in the sense that the software won't crash every half an hour, but work intuitively and hassle free, as if it were built just for them. As users become more computer savvy, they have a better understanding of how they expect programs to behave. More often than not, users know what they want to achieve. Any software that hinders them from efficiently achieving their goals will quickly be replaced. This is particularly true where users can exercise their freedom of choice between a number of alternatives, such as with Internet-based applications where a viable alternative is only a Web-search away. If you are building this kind of software, you need to tune in to your users quickly.

There is no doubt that to continue to provide value for our customers, we must continue to apply the principles of the agile development philosophy. But, in order for our software to be truly successful in the eyes of its biggest critics, we must endeavor to adopt a more user-centered approach.

What Is User-Centered Design?

User-centered design (UCD) can be applied to the design of anything that has a user—from mobile phones to kitchens. But when integrating UCD with agile practices, we are only concerned with the arena of software development. With agile development, the primary measure of progress must be related to *working software*. Once you adopt a user-centered philosophy, this is no longer the whole story.

Unlike agile, UCD is not focused on the customer—it is centered on the end-user. Furthermore, from a UCD standpoint, software is incidental; what is important is that end-users can easily and efficiently achieve their goals—with or without software.

Why Do We Need User-Centered Design?

There are a number of benefits that arise when advocating a user-centered phi-

losophy. User-based research provides a mechanism against which design decisions can be validated and tested. Evidence-based decisions mean that guess work is minimized. What to build becomes much less of a matter for debate. More importantly, by keeping a product's end-users at the heart of its design and development process, the end result is far more likely to be useful, usable, and meaningful.

The era of feature-centric development is coming to an end. Consumers are beginning to realize that more features do not always mean a better product. In the current maturing marketplace, quality of experience is far more likely to be a product differentiator than number of features—think iPhone vs. Nokia or Wii vs. PS3.

UCD provides a way to engineer these quality experiences. As such, it empowers development teams to create products and solutions that are competitive in today's discerning market. By embracing a UCD philosophy, one could argue (as Peter Merholz does in "Experience Is the Product" [1]) that we should not just create products and services—we should aspire to build better overall experiences where the value (both quantitative and qualitative) to all concerned is obvious.

What Is Agile Software Development?

Agile software development is a philosophy that provides a framework defined by a set of values and principles that attempt to focus software developers on delivering the most value possible to their customers.

Why Do We Need Agile Software Development?

Building software is a tricky business. There are a number of factors that contribute to this complexity:

- Writing software is technically complex and mentally challenging.
- People write software, and people do not perform in a consistent manner.

- When software is commissioned, people might know the broad problem to be solved but do not understand the intricate detail of how software can help.
- Software takes a long time to implement, and the context of the problem domain can change while the software is being built.

Agile recognizes these issues and provides some guidelines to help mitigate the risk they cause. When working to the agile principles, we focus on delivering software to the customers as early and as frequently as possible, allowing them to give feedback, validate, and adjust their perception of what it is they *actually need* from the software.

Commonalities Between UCD and Agile

A striking but sometimes overlooked similarity between agile and UCD is that both are often fundamentally misunderstood by people starting out on the path of adoption. This misunderstanding stems from the fact that both are frequently assumed to be *methodologies*—magical, step-by-step recipes that you can follow to guarantee project success. In fact, agile and UCD are both *philosophies*.

There are many different methodologies that implement the agile philosophy: Extreme Programming, Scrum, the Crystal family, etc. There are also several different interpretations of UCD. User-experience aficionados can learn from the way products and experiences are created by the likes of Adaptive Path, Cooper, Apple, Shedroff, Morville, Spool, Nielsen, and Norman (see the StickyNotes). While the specific methods are very different, the underlying philosophies are undeniably similar.

Both the agile and the UCD philosophies are iterative; they progress in small steps providing opportunities for verification and refinement along the way.

The Conflict Between UCD and Agile

There is a history of conflict between agile developers and user-experience designers. Agile software development is predominantly a developer-led phi-

losophy, while UCD is, in many organizations, championed by creatives. There will always be a healthy, natural tension between these left-brained and right-brained individuals.

If we examine the principles behind the agile and UCD philosophies, then we are much more likely to find tangible issues that can be addressed to facilitate agile developers and UCD practitioners working together to create quality software and robust user experiences, rather than falling back on the age-old argument that developers are from Mars; designers are from Venus.

To better understand the disconnect, let's compare some of the principles of the Agile Manifesto that cause conflict with the UCD philosophy.

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.—Agile Manifesto

The Agile Manifesto focuses directly on providing value for the customer, whereas UCD champions the end-user—the idea being to create software that users “cannot live without.” By delivering such intuitive software we cannot fail to deliver knock-on value to our stakeholders.

The equivalent primary principle for UCD might read:

Our highest priority is to help create an experience for end-users where they can achieve their goals easily and efficiently with minimal disruption to their mental model of the problem space.

Working software is the primary measure of progress.—Agile Manifesto

The concern here is the definition of working. The aim of this principle when it was defined was to get software development teams out of the mindset that creating a lot of design documentation before writing any code was helping to meet the project deadline. The goal of a software development project is to produce functioning software, not UML diagrams. From a UCD perspective, software that simply works is a secondary measure of progress. Much more important is whether the software helps the users achieve their goals.

The equivalent UCD principle might read:

The satisfaction of end-user needs (user goals) balanced with the achievement of business goals is the primary measure of success.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.—Agile Manifesto

Agile development calls for early delivery of the software to the customer. This is *not* synonymous with releasing to end-users. The customer's public release strategy can be entirely separate from this process. Early delivery to customers allows beta tests or usability trials to be performed and the customer to realign his priorities based on the findings.

If we release too early to the market, the end-user experience could be poor. In some situations, this could be seriously detrimental to the impact of the product on the market and ultimately the bottom line. For example, in 2005, the early release of handsets that were touted as “feature complete” and “bug free” cost the mobile phone industry \$4.5 billion:

“One in seven mobile phones are returned within the first year of purchase ... 63% of the devices being returned are done so without fault.”

“Most of these issues may be addressed through stringent device testing and usability modeling prior to the launch of the mobile device.” [2]

Common Issues When Integrating UCD and Agile

Design without constraints

When the design of a system is created with users and customers but without regular feedback from a development team, there is significant risk of a design being proposed and approved when no one has any idea of how long it will take or how much it will cost to implement. When the estimate does come in, and it's far above the amount expected, it is understandable that the customer will feel let down by the development team. To compound the problem, the design team will often be frustrated that the only way to deliver the software within the real budget will be to “water down” its design. As stated in the Agile Manifesto: *Business people and developers*

must work together daily throughout the project.

In this context, it is often considered that the UCD practitioners would operate as part of the customer team [3] to help drive the direction of the product.

Validate with real-world usage

UCD is an iterative process that calls for designers to validate and refine their design regularly. While the design is being created, many techniques are used to perform this validation that provide value through quick and short feedback loops. Despite this, we maintain that the design of a successful user experience cannot be completed without feedback from people using the real system. This is where we get benefit from the primary agile principle mentioned earlier: *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*

To be able to perform and respond to this level of validation, it is necessary to have the design team integrated with the development team to run the tests and modify its design based on the findings.

Handover is the enemy of understanding

Whenever one team is responsible for creating something it will hand over to another team, there is a risk that the receiving team won't understand the theory or mental model behind the item. This is a common problem even when the two teams have similar backgrounds, but it is compounded when the teams have very different skill sets and backgrounds, as is the case with development and design teams. When designers want developers to implement their ideas, it is not enough to pass on screen shots or scribbles. There must be a shared understanding of the problem domain and the theory describing the reasons why certain decisions have been made. Like the Agile Manifesto says: *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*

No time to iterate

A common flaw, articulated by both Alistair Cockburn [4] and Jeff Patton [5], is that on an agile development team

people forget to iterate, or at least forget to plan to iterate.

It is quite common for an agile software development team to forget that the point of all the feedback is to allow customers to inspect and verify the features that they have requested. The result is that it comes as a shock to the customer that valuable features must be postponed when a recently implemented feature needs to be iterated over again.

When taking a user-centered approach to agile development, it is critical that we make time to iterate so we can respond to user feedback without pushing out features that we have told the customers they will receive. There are a couple of strategies to manage this: Take a percentage of your initial estimate and add it to the schedule, or consider Alistair Cockburn's approach of creating three cards for user rights [6] by planning the implementation and two additional iterations of each story.

However you solve this problem, the first step is recognizing that while providing shorter feedback loops to the customer and users is valuable, it doesn't count for much if we don't plan to get the benefit from the feedback we receive. Keep in mind the importance of the agile principle that says we should: *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*

Divided responsibilities; divided teams

Agile developers often focus more on the delivery of software than on *valuable* software. Following the "you aren't gonna need it" [7] mindset, agile developers run the risk of just developing the minimum of what they are asked without actively taking the responsibility to volunteer information regarding alternative approaches to the product owner or UCD practitioner. We must actively challenge the mindset of divided responsibility—"You spec and design it; we'll build what you spec." Everyone should work toward the shared vision of a successful experience.

The flip side of this is the UCD practitioner or product owner's not actively seeking collaboration from the devel-

opment team and, therefore, not being aware of the technical constraints upon the proposed design.

This is addressed by the agile principle: *Business people and developers must work together daily throughout the project.*

Conclusion

In today's marketplace, experience is king. In the eyes of end-users, what makes software great is their perceived experience. So how do we build great software? As one would expect, there is no magic formula.

Not only does agile provide us with tangible software sooner but it also provides the transparency and constant feedback against which we can validate and steer decisions. User-centered design can help us remain focused on the goals, frustrations, and desires of our end-users.

A successful combination of these two philosophies, while sometimes painful for the practitioners involved, will result in the creation of product experiences that provide the "wow" factor to captivate the discerning users of today's marketplace. {end}

REFERENCES:

- [1] www.core77.com/reactor/06.07_merholz.asp
- [2] www.wdsglobal.com/news/whitepapers/20060717/20060717.asp
- [3] www.agileproductdesign.com/blog/emerging_best_agile_ux_practice.html
- [4] alistair.cockburn.us/index.php/Incremental_vs_iterative_development
- [5] www.stickyminds.com/s.asp?F=S13178_COL_2
- [6] alistair.cockburn.us/index.php/Three_cards_for_user_rights
- [7] c2.com/xp/YouArentGonnaNeedIt.html

Sticky Notes

For more on the following topic go to www.StickyMinds.com/bettersoftware.

- Further reading