Lecture Notes in Computer Science. Springer. Extreme Programming and Agile Methods - XP/Agile Universe 2004. page 50-59. November 2004.

An Agile User-Centered Method: Rapid Contextual Design

Hugh Beyer[1], Karen Holtzblatt[1], Lisa Baker[2]

[1] InContext Enterprises, Inc., 2352 Main St., Suite 302, Concord, MA 01742
{beyer, karen}@incent.com

[2] LANDesk Software, Inc., 698 West 10000 South, Suite 500, South Jordan, Utah 84095
lisa.baker@landesk.com

Abstract. Agile methods have proven their worth in keeping a development team focused on producing high-quality code quickly. But these methods generally have little to say about how to incorporate user-centered design techniques. Also the question has been raised whether agile methods can scale up to larger systems design. In this paper we show how one user-centered design method, Contextual Design, forms a natural fit with agile methods and recount our experience with such combined projects.

Introduction

Agile software development methods propose a new approach to the old problem of quickly developing reliable software that meets users' needs. But their strong focus on improving the engineering process neglects questions about how other disciplines fit in, and how agile methods fit in with the larger organization. The role of user-centered design, user interface design, and usability in an agile team is unclear. It is also unclear how well the approaches work with larger teams and projects [1, 2].

At one level there should be no problem—the developers of agile processes are very clear that developers should work closely with their customers [3]. Customer orientation is built into the basic method. Rather than provide complex techniques for requirements elicitation or user research, agile approaches make users part of the release planning and iterative development process. But where does this leave the techniques of user-centered design? Perhaps they should be used before agile methods start—perhaps user-centered design techniques are superceded by continual user contact throughout the project. How exactly is this relationship to be reconciled?

Furthermore, there are deep philosophical differences between agile methods and most other software engineering methodologies. User-centered design tends to assume that initial research, design, and planning happens at the start of the project. This may well look to agile practitioners like "big design up front"—a bad thing in the agile programming rule book. To reconcile these different world-views we, along with people such as Constantine [4] and Kane [5], have looked at the issues of incorporating user-centered design and usability into an agile development effort by including additional techniques.

In this paper we analyze the underlying assumptions of agile methods from the point of view of classic user-centered design. Each of these assumptions presents a process challenge, and agile methods incorporate ways to solve them. But the challenges themselves are not new. In developing Contextual Design (CD), our customer-centered systems definition method [6], we encountered many of the same issues—as, indeed, any practitioner must. In our work with teams committed to XP as a development methodology we have adapted our approach to a quick-turnaround, short-development-lifecycle project. This has given us some insight into strengths and weaknesses of both approaches. In this paper, we will show how integrating the approaches can cover the fast-turnaround iterative project as well as the large-scale, high-impact, enterprise project, and fill the gaps in agile methods.

The Power of Agile Methods

Agile methods [3, 7, 8] are based on what we will call *axioms*—principles or assumptions that are almost self-evident. Each axiom presents a challenge to the traditional development process, and agile methods propose a solution to each.

The axioms we will deal with in this paper are:

Axiom 1: Engineers engineer. Software engineers write code. Anything outside this discipline is not something engineers should be expected to do.

Agile methods respond by putting processes in place (such as the XP's Planning Game for release planning [3, p.86]) that allow engineers to interface with those who have the design and usability skills needed to complete the project successfully..

Axiom 2: The only one who knows what the user needs is the user. Developers certainly do not know what users need and in fact are *not* the user..

So agile methods make the user part of the development team. They create a very close relationship with the project's users, assign a user representative who spends considerable time representing the users to the team, or assign an actual user to sit with the development team.

Axiom 3: Development requires some sort of plan, but plans always change.

Therefore, agile methods do the minimum of up-front planning so that as little time as possible is spent on a plan that will soon be obsolete.

Axiom 4: Any project will miss the mark to some degree, and the only way to know if something works is to test it. The sooner project results are tested, the sooner errors and discrepancies are discovered.

So agile methods focus on short release cycles and short iterations. Engineers receive user feedback consistently throughout the development cycle, rather than near the end at formally defined "alpha" or "beta" stages. Each iteration is a coherent subset of functionality that can be tested directly by the users.

These axioms cover the part of the process we wish to consider here—the part which defines the interface between the development team and the rest of its parent organization. Similar axioms could be defined for the rest of the agile development practice. We leave elucidating these as an exercise for the reader.

A Customer-Centered Response

Agile methods provide a response to the issues raised by each of these axioms that make sense from an engineering perspective. (As Alan Cooper said, these methods seem to have been created by engineering to defend itself from the failings of the parent organization [9].) But the community of user-centered researchers and practitioners have been dealing with variations of these issues for years [10, 11, 12, 13]. What is the user-centered design perspective on these problems?

Contextual Design is a well-respected user-centered design method that has been around for over 10 years. It provides techniques covering the entire front end of systems design, from figuring out who your users are and how they work through designing the detailed user interface and functionality of the system. CD has been used on very large scale projects—defining complex business processes, corporate web sites, and portals—on software tools, and on small, self-contained parts of many products and systems.

The techniques available in the full Contextual Design process are as follows:

Contextual Inquiry: Field interviews with users in their work places while they work, observing and inquiring into the structure of the users' own work practice. This ensures that the team captures the real business practice and daily activities of the people the system is to support, not just the self-reported practice or official policies..

Interpretation sessions and work modeling: Team discussions in which the events of the interview are retold, key points (affinity notes) are captured, and models representing the user's work practice are drawn (including as-is sequences of tasks). This disciplined, detailed debriefing allows the team to share the findings, build a common understanding of the user, and capture all the data relevant to the project in a form that will drive the design.

Consolidation and affinity building: The data from individual users is consolidated to show a larger picture of the work of the targeted population. The affinity notes from all users are brought together into an *affinity diagram*, a hierarchical representation of the issues labeled to reflect user needs. Work models are consolidated, showing the common work patterns and strategies across all users—the "as-is" work process.

Visioning: Together, the team reviews the models and invents how the system will streamline and transform the work users do. This is captured as a hand-drawn sketch on flip chart paper. This *vision* represents the big picture of what the system could do to address the full work practice. It can subsequently be broken down into coherent subsets so that the vision can be implemented over a series of releases. Alternatively, in a smaller project the team may simply brainstorm solutions.

Storyboarding: The new design for work tasks are sketched out using pictures and text in a series of hand-drawn cells. A storyboard includes manual practices, initial UI concepts, business rules, and automation assumptions. This becomes the "to-be" work model.

User Environment Design (UED): A single representation of the system showing all functions and how they are organized into coherent places in the system to support user intent. The UED is built from the storyboards. This ensures a large system is coherent and fits the work. It provides a basis for prioritization and rational segmentation of the system.

Paper prototypes and mock-up interviews: User interfaces are designed on paper and tested with the system's actual users, first in rough form and then with more detail. This ensures the basic system function and structure work for the users, and that the basic UI concept is sound.

A Contextual Design project exploits just those CD techniques needed for the project at hand. (Rapid CD uses just a few of them.) But each technique exists to solve a particular problem in managing a development project, just as the techniques of agile methods do.

Looking back at the Agile Axioms we can now discuss them from a customer-centered perspective.

Axiom 1: Separating Design from Engineering

Much of the distinctiveness (and much of the value) of agile methods comes from the clear separation of responsibilities they bring to the development process. Developers write code and design the implementation of systems—that is what they are good at. They are not good at understanding how people work, creating effective and intuitive user interfaces, or making an interface usable.

The great strength of agile methods is that they focus the engineers on doing what engineers do best. The weakness of agile methods is that they give little guidance in figuring out what to tell the engineers to do.

In particular, they do not attempt to cover questions such as:

What is the scope of the system? What problem will it solve, and what tasks will it affect? Will these tasks be streamlined through the introduction of technology or replaced entirely by automated system? What will the overall effect on the business be? Developing User Stories depends on these questions having been answered already.

These design questions fall into the domain traditionally known as "systems analysis" or "requirements analysis" and aren't covered by agile methods at all. Some organizations cover these areas by having product marketing scope out the product release according to business and marketing needs. They then hand off a high-level overview of the user stories to engineering.

CD provides Contextual Inquiry, work modeling, consolidation and affinity building, and visioning to support these activities.

What should the basic function and structure of the system be? Designers must decide how a system will be structured to support user tasks, how functions will be grouped into coherent interfaces to support coherent parts of a task, and what those functions need to do.

There's no good support for these design activities in traditional programming methodologies because they were not a major issue in traditional forms-based systems. These issues arise from the sophistication available in modern user interfaces and the close integration between the system and the work it supports.

The new concepts of "user experience architecture" and "user interaction" have been coined to cover these activities. Agile methods do not provide specific techniques to address them. Instead, agile methods send each completed iteration off to designated customers so engineering can receive feedback earlier in the process.

Engineering then cleans up and fixes the UI afterward based on user feedback. (The XP saying is: the earlier you get it to the customer, the earlier you can fix it.)

In CD, we support them directly through Storyboards and User Environment Design. These define exactly the level of structure needed to provide coherent and effective User Stories.

What is the user interface? Screen layout, button labels, tree views, drag and drop, hyperlinks, pulldown menus, etc. must be assembled into coherent, pleasing, and transparent screens that support the user's work in the new system.

This is classic user interface (UI) design, and it is the orphan child of software development methodologies. Is it design? Is it analysis? Does a requirements specification include the UI or does it not? No one seems quite sure.

Regardless of where UI design is situated organizationally, a successful agile development project depends on the skill being available, even if it does not explicitly assign such a role. Because the UI is the interface between the user and the system's function, the only way to test the system is through the UI. Effective UI design is a prerequisite for agile methods to work, but the methodology provides no separate focus or testing. In past projects, we have included user interface mockups as part of the story definition and acceptance testing criteria for the iteration. We found even this practice to be incomplete—it focuses on individual stories and does not create an explicit, cohesive view of the product and work practice.

CD recognizes and resources UI design as a separate discipline—which fits the agile approach of focusing the developers on developing code. The UED (when needed) or vision provides requirements for UI design and the paper mockups permit the UI to be considered and tested on its own, independent of the developer's underlying implementation.

Taken together, these elements of CD dovetail to fill in the gaps left in agile processes. They support the thinking and design process needed for the release planning and User Stories to be effective.

Axiom 2: The User is the Expert.

Agile and customer-centered approaches agree in recognizing the user as the final authority and only arbiter of what makes a good system. But how to make this voice heard? The agile approach is to put the user on the team, and if there are multiple users, assign one as the representative who makes all choices. [3, p. 68] But there are some drawbacks to this approach:

Users cannot articulate their own work practice. When asked what they do, people give their impression of what they do—which may or may not be accurate [10, 14]. In one example from our own experience, system managers told our project team that most of their job was troubleshooting. Later field interviews revealed that in fact, they spent most of their time doing simple tasks such as installs and user management.

A "representative" user never is. No one person can embody all the users of a real enterprise system. While interviewing several users reveals a common work practice, no *one* user ever represents that work practice in its entirety. Other stakeholders in the system—secondary users, management, upstream and downstream roles in the process—must also be considered. Furthermore, the more that person becomes part of the engineering organization, the less useful they are as a user surrogate. They learn too much about the technology and they become invested in the team's thinking. They become more empathetic to the engineer's challenges and less connected to the challenges they faced in their previous job. We found that our own user representatives are just too nice to us.

Customers are not designers, any more than engineers are. They know where their shoes pinch but not what the technical possibilities are, and they cannot easily envision what a future work practice might be. An experienced person's work is habitual and automatic, which makes it hard to envision a new and different world supported by an unknown technology.

So what will make the user a powerful voice on the team? Customer-centered design has been wrestling with this question for years. CD's answer is: if you want to learn about the customer's work, apprentice yourself to your customers. Don't ask them to talk about their work out of context. Go to their workplace and watch what they do, discussing it with them. Represent their voice in the data and design to the data.

Axiom 3: Keep Up-Front Planning to a Minimum

Agile methods distrust up-front planning intensely. Business needs change, the business climate changes, or the customer discovers they did not really understand what they needed after all. So agile methods view time invested in up-front planning as probable time wasted. Up-front planning is kept short, relatively informal, and focused on just the next release—which is also short.

CD takes a different tack towards the relationship between requirements and plans, because of its different starting point:

Work practice changes very little over time. The initial focus in CD is on understanding customer work practice, as opposed to defining a particular system. The work patterns, strategies, and intents discovered during this phase are fundamental to how people work and change little over time or across a wide range of users. It is feature implementations or underlying technology that changes rapidly. When we develop the full set of CD models to represent a user population, we find those models are still useful years later. We return to them when starting new projects serving the same population. These models are *more* accurate and robust than the voice of a single user representative who has not made a formal study of the work. Compared to hiring an on-site user, the CD models are useful longer and over time cost significantly less.

A solid understanding of the user leads to speed. Once the work practice of the user is understood, rapid iterations of design and implementation become possible. The data drives the current release, suggests the focus for the next, and provides a base understanding for each iteration. It has been our experience that even in well-functioning agile teams, velocity slows each time the engineers come upon a new storyline. They question the details (or lack thereof) included in the story definitions and question the basis of the user requirements.

'Minimum planning' depends on project scope. A small, quick-iteration project only needs a small amount of planning. Field studies of 5-8 users, a quick consolidation and brainstorming, all completed within a week or two—this is enough for a fast-turnaround release and makes the user presence on the project unnecessary. But a large project introducing disruptive technology will require much more up-front data gathering and planning and will need more time to budget resources and market priorities. Bankston's concept of an "architectural spike" [15] is a useful way to think about this more detailed planning.

So the CD philosophy is: Do only the planning you need to do; use team-based, high-communication processes to drive quickly to a common understanding of the problem and the solution; and create only the artifacts you need for the next step of design.

Axiom 4: Work in Quick Iterations

The classic design methodologies are frequently criticized for lacking prompt feedback—a two-year project would typically get its first real customer feedback a year and a half into it, when it first went to field test. At this point, it was far too late to discover that the project should have been addressing a different problem, or that it was addressing the right problem the wrong way. Any user feedback was thrown onto the feature pile for the next release. To compensate for this, agile methods prescribe rapid iterations in development (we plan 3-week iterations) and short release cycles.

But creating code—no matter how fast it is created—only to rework it later is wasteful. Instead, CD starts the iterations even before coding begins.

Test and iterate the spec before code. XP's "Developer's Bill of Rights" states: "You have the right to know what is needed, with clear declarations of priority in the form of detailed requirements and specifications." But the specification will never be correct unless it is tested with users to ensure it fits their real—as opposed to espoused—needs.

But how can a user test a specification? Few people can foresee the impact of a proposed system change merely by hearing such a change described. Instead, CD uses paper mockups to present a proposed new system to its users as though it were real. Users can work through their own actual work problems in the paper system, with the designer manipulating it to show the system's responses. When problems arise, user and designer change the system together, immediately, to better fit the user's work practice. Rough paper mockups are sufficient to show that the right system, the right structure, and the right functions are being defined.

Test and iterate the UI before code. Once the basic system has been proved, paper mockups can be refined to represent the proposed user interface. This ensures the basic UI concept is suitable before the team starts iterating it in code.

Test and iterate in code when needed. Once the spec and basic UI are in place, agile development's short iterations become the central driver for testing. Each iteration is a working version of the system, albeit with limited functionality. These iterations can be used to test the final UI and actual behavior of the system with customers to ensure low-level usability concerns are identified and dealt with early.

Testing iterations with customers as they are completed and using the results to refine the team's direction is good. But there's no reason to wait until after code is cut. CD makes the fast iteration and course-correction part of the specification process itself.

Building a New Process

The above discussion suggests how a customer-centered design approach such as CD can coexist with agile methods—indeed, how the two complement each other so well that they form a very strong combination. *Rapid CD* [16] is a fast, effective, customer-centered way to design a product when quick turnaround is desired.

Here is an overview of the process, step by step, with typical time estimates for each step. We assume a team of 2 UI designers working with a team of developers:

1. Set project focus. Determine the complexity of the project and the level of innovation required. Identify the 1 or 2 key customer roles this product release will support and plan customer visits. (½ day for discussions, but expect 2-3 weeks to set up visits from a standing start. Once you have relationships and organizational expertise, it is easier.)
2. Do Contextual Inquiries with potential customers. Gather data from at least 3 people in each role. In a week, a team can do 8 interviews with people from 4 organizations and interpret that data, producing affinity notes and sequence models (as-is tasks). Ideally, this is done in a cross-functional team of UI people, marketing, and developers. In practice, we find that developers are usually finishing up their previous project and we bring them up to speed later. (1 week)
3. Build an affinity showing the scope of issues from all customers, and sequence models (task models) showing how specific tasks to be supported by the project are now done. This is a representation of the "as-is" user work practice. (3 to 4 days)
4. Introduce the larger team (including the full development team) to the customer roles and customer data. Summarize key findings, then walk the team through the affinity to allow the team members to comprehend the customer environment. Ask each team member to note questions and design ideas.
5. The full team identifies what issues will be addressed by the project. Collect issues from the affinity, choosing the most critical issues that can be addressed within the project scope. Brainstorm ideas of how to better support the work. Record and save big ideas for future high-impact projects. (2 days)
6. Build User Stories in response to these issues. User Stories are guided by the sequence models and show how the system will resolve the issues.
7. Run a release planning process on the User Stories. Use conceptual diagrams and high level UI mockups to facilitate team communication. Without a completed UI the team can't know exactly how difficult implementation will be, but within the context of an organization the team can know the typical complexity of the UI's they define, so they can supply a rough estimate. Organize the User Stories into Iterations, groups of stories that deliver coherent subsets of function.
8. Prioritize and eliminate stories as necessary to meet resource budget for the release. (We always save some budget for additional user stories that will reveal themselves once we begin getting user feedback.)
9. Design detailed user interfaces to support the User Stories in the first Iteration. UI design is its own discipline—don't mix it with the implementation work of coding the User Story. (1-2 days)
10. Test UIs with users in paper with mock-up interviews. User Stories are a fairly fine-grained definition of system functionality; many User Stories can be covered

in a single paper prototype test. Test these UIs with 3 to 4 users and use the results to refine the design. Do a second round of tests with a more detailed UI if you have the time and resources. A third round of testing will happen with live code. (2 weeks for both rounds)

11. Provide the User Stories and completed UIs to the development team for implementation. With detailed UIs, developers can very accurately cost their work for the iteration. In addition, testing can incorporate UI mockups into their acceptance tests, providing development with a clear end point to their task.
12. During implementation of the first Iteration, the UI team develops the UIs for the second Iteration's User Stories and tests them with users in 2 rounds of paper before the code for the first Iteration is complete.
13. When the code for the first Iteration is completed, the UI team gives developers the next set of stories and UIs and the developers start on the second Iteration.
14. Meanwhile, the UI team designs the UIs for the third iteration and tests them with users in paper. Simultaneously, if desired, they test the running code of the first iteration with users to get quick feedback on the actual product. (Our projects have done this with customers every second or third iteration.)
15. At the end of the second iteration, the UI team gives the User Stories and UI designs for the third iteration to the development team and the testing feedback is incorporated into the plan. This process repeats until the release is done.

    If user testing suggests changes to future User Stories, the changes are made and the work estimate for those stories changed if necessary. When user feedback indicates you must change work the team has already done, plan additional User Stories and schedule them in as needed. (Be aware this *will* happen as the system comes alive and low-level issues reveal themselves. Be careful not to schedule yourself too tight in your initial resourcing. The team will need to save some of its resource budget to accommodate these additional stories.)

This plan assumes a separate UI design team exists that will work out the details of the interface within the context of the User Stories. In practice, UI design is usually a separate skill held by different people on the team. Our experience is that this sort of handoff—once the developers have come to recognize the value of the skill—is very easy. (In fact, once developers figure out how much time and effort the UI designers save them, the developers are prone to complain that the UI designers haven't told them *enough* about what to do and have left them with too many choices.) We also find it promotes better understanding to have developers accompany the UI designer on some user tests.

For highly complex or highly innovative projects, a more traditional CD process can be used to drive a large scale design. In such a process, the full set of CD customer work models represents the complete "as-is" work practice to ensure the existing process is really understood. The CD vision is used to synthesize a coherent design response to the work problem.

The UED model becomes the key representation of the new system. It shows all the parts of the system and all the function, to maintain the coherence of the system as a whole. It also shows how the system can be broken into coherent chunks for implementation. Each of these chunks becomes input to an agile team—the release planning and user stories are oriented towards delivering that chunk and the UED keeps the work of the multiple teams in sync.

Conclusion

Agile methods are based on sound principles—so sound that when they are articulated, they seem impossible to argue with. If they are to be criticized at all, perhaps it is because many of the solutions agile methods promote draw on a limited set of techniques for managing organizations and understanding customers. In this paper, we have shown how incorporating customer-centered techniques such as CD provide additional solutions to the real problems recognized by agile methods. These solutions work in combination with agile methods' strengths resulting in a process that incorporates the customer voice, provides room for UI and user interaction design, and can address significantly large projects.

References

1. B. Boehm and R. Turner, "Observations on Balancing Discipline and Agility" presented at *Agile Development Conference 2003*, Salt Lake City, Utah, and archived at http://agiledevelopmentconference.com/2003/files/P4Paper.pdf
2. J. Grenning, Using XP in a Big Process Company, article at http://www.agilealliance.com/articles/articles/XPInABigProcessCompany.pdf
3. K. Beck, *Extreme Programming Explained: Embrace Change*. San Francisco: Addison-Wesley, 2000.
4. L. Constantine, "Process Agility and Software Usability: Toward Lightweight Usage-Centered Design," in ForUse Conference Proceedings, 2003.
5. D. Kane, "Finding a Place for Discount Usability Engineering in Agile Development: Throwing Down the Gauntlet," at the *Agile Development Conference 2003*, as archived at http://agiledevelopmentconference.com/2003/files/P5Paper.pdf
6. H. Beyer and K. Holtzblatt, Contextual Design: *Defining Customer-Centered Systems*, Morgan Kaufmann Publishers Inc., San Francisco (1997).
7. A. Cockburn, , *Agile Software Development*, Addison Wesley, Reading, MA, 2002.
8 J. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York: Dorset House, 2000.
9. A. Cooper, as reported by E. Nelson in *Extreme Programming vs. Interaction Design* at FTPOnline: article at http://www.fawcette.com/interviews/beck_cooper/default.asp
10. J. Whiteside, J. Bennett, and K. Holtzblatt, "Usability Engineering: Our Experience and Evolution," *Handbook of Human Computer Interaction*, M. Helander (Ed.). New York: North Holland, 1988.
11. L. Suchman, *Plans and Situated Actions*, Cambridge University Press, Cambridge, 1989.
12. T. Winograd, *Bringing Design to Software*, ACM Press, NY, NY, 1996.
13. P. Seaton and T. Stewart, "Evolving Task Oriented Systems," *Human Factors in Computing Systems CHI '92 Conference Proceedings*, May 1992, Monterey, California.
14. M. Polanyi, *The Tacit Dimension*, Routledge and Kegan Paul, 1967.
15. A. Bankston, *Usability and User Interface Design in XP,* article at http://www.ccpace.com/resources/documents/UsabilityinXP.pdf
16. K. Holtzblatt, *Rapid CD*, Morgan Kaufmann Publishers Inc., San Francisco (forthcoming).