Becoming Agile

Usability & Short Project Cycles

BY GWEN PEARSON & SUSAN PEARSALL

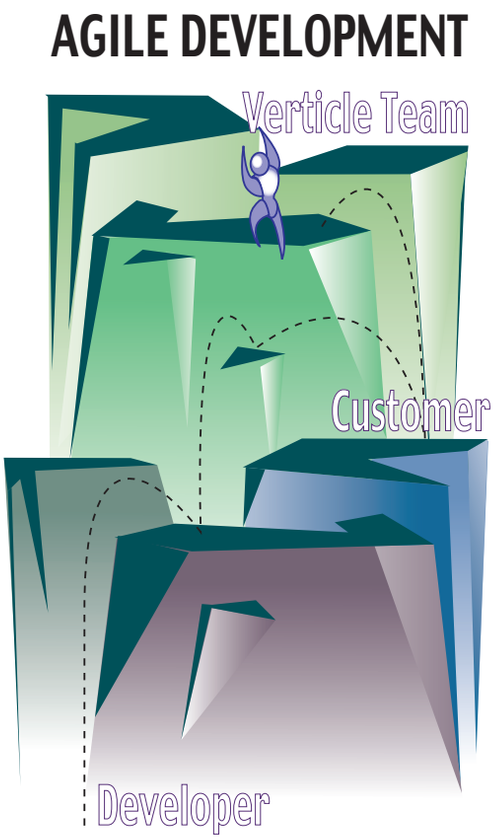
As usability professionals, we need to identify our role and contributions in the larger project team. We often have to show how our work can be accomplished without adding more time to the overall schedule. This task is especially challenging as cycle times have become shorter and shorter.

The push for shorter schedules and customer-focused deliverables has led the development community to try new approaches. In the 1990s, the agile development approach arose in recognition of the continuously changing behavior of technology and the business environment. The traditional waterfall development processes followed a linear process that attempted to capture and document all the requirements and specifications before code development began. The traditional approach did not meet the needs of product development teams who had to respond more quickly to changes in customers' business models.

In agile development, on the other hand, the team assumes that the environment is constantly changing; exploration and iteration are necessary; collaboration among individuals with unique strengths is the best approach; and requirements are minimalist guidelines.

This basic philosophy is described in the Agile Manifesto's principles of agile development.

1. First priority is to satisfy the customer.
2. Welcome changing requirements.
3. Deliver working software early and frequently.
4. Have business people and developers work together daily.
5. Select motivated individuals and create a supportive environment.
6. Use face-to-face conversation to communicate.
7. Measure progress by working software.
8. Maintain a sustainable pace of development.
9. Attend continuously to good design.
10. Focus on the important work and minimize nonessential work.
11. The best work will come from self-organized teams.
12. Retrospect and make adjustments at regular intervals.

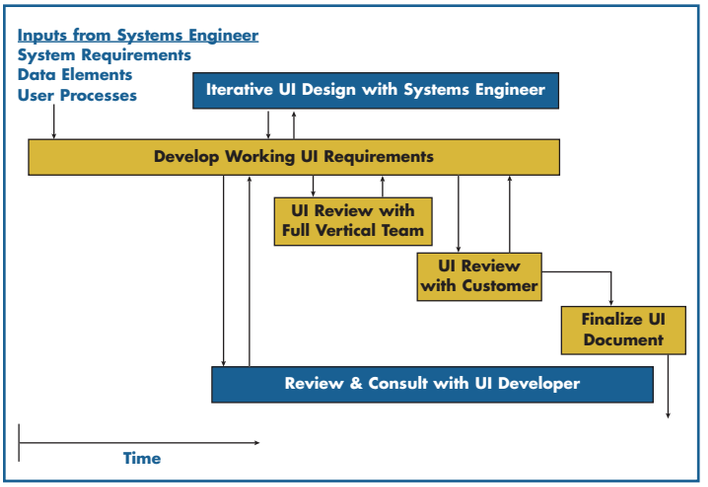


We have been on a team that has used this approach for the past several years and have had to figure out how to make usability contributions to several projects with very short cycle times simultaneously. In this article, we will describe how we have successfully integrated usability practices into agile development.

Our Approach

We created an iterative design process in which users and development team members used a progressive interactive schedule. There were five independent application teams that each contained a project manager, systems engineer, back-end developers, a development lead, and the customer for the project. There were two support teams who worked across all projects. One support team consisted of user interface (UI) engineers and UI developers, and the other consisted of system administrators

Essentially, the user interface systems engineer was the GUI architect. The UI systems engineer received system requirements and a completed users' environment questionnaire from the systems engineer and then started the framework for the applica-



UI systems engineering design process

tion. Once the main functionality was defined, there was a review with the UI systems engineer and UI developer to determine high-level development feasibility. The results were used to refine the requirements, communicate coding limitations to the systems engineer and users, and, as necessary, change the design.

Because agile development does not require complete or comprehensive requirements before development starts, the UI team can begin building the new application before the requirements are finalized. Once the first draft of requirements was complete, there was a UI review with the full application team (systems engineer, project manager, developers) and UI developers. This review was held at the same time as the systems engineer's detailed backend requirements review, which included the logic for the high-level software design and data specifications. In the joint review meeting, the backend and frontend developers could communicate their plans face-to-face and eventually be able to integrate their code more easily.

The application team and customers reviewed the revised documents together at a customer review meeting. The document was then finalized and the application developed.

After an application was completed, the development team held a meeting to identify areas for communication and process improvement (for example, if the backend developers don't provide an architecture with tables that the frontend developers can use, it impacts UI development and may alter the GUI design). These lessons learned were incorporated into the next application cycle. During down times, the UI team focused on areas that could be improved modularly (for example, updating style sheets to reduce wasted space).

Keys to Successful Integration

Our goal was to create the best possible user experience given the constraints of the time and resources available. We used several user experience tools to achieve this goal under agile development constraints.

For example, to deliver requirements quickly but still communicate clearly, we used a template for requirements. The template was built on our existing reusable user interface elements (for instance, menu layout and widget behavior), and was based on a design system following basic standards that had been reviewed and approved earlier by the project team.

Developers and repeat customers knew what to expect when they reviewed the requirements. The template included PowerPoint-based paper prototypes; minimal requirements were presented as bulleted features associated with a picture of each screen. Since pictures are easier to understand than words, except when defining actions, the users and development team could quickly understand the scope of the application.

In addition, every requirements document included information on the user environment and user goals; a task flowchart; the login screen; the main application menu screen; and each additional screen. Each screen also had a page describing the functional and behavioral requirements (widget logic, messaging, text entry parameters, drop-down defaults, and so on).

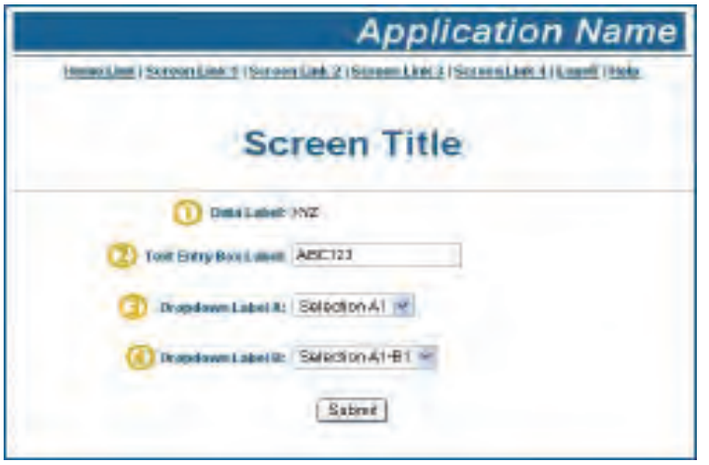
The UI engineers maintained a project website as the central requirements repository. The developers supported the design system by developing style sheets to jump-start development.

When we reviewed the requirements as a team, members either attended in person or used virtual-conference tools such as NetMeeting. During development, we were always available to explain feature behaviors to coders and we reviewed code progress frequently.

User-Centered Design in Agile Processes

Because users were on the teams, we were able to employ user-centered design techniques very efficiently. For example, we conducted work environment assessments that showed whether they used PCs or Suns, for instance, and what browsers they were using. The results of

➲ Continued on pg. 18



1. Data field: Brief identification of the data displayed in this field.
2. Text-entry field: Brief description of data entered by user. Can be required or not required.
3. Drop-down selection: Contains the following options in the following order—
   - Selection A1
   - Selection A2
4. Drop-down selection: Contains the following options in the following order—
   - Selection A1-B1
   - Selection A1-B2
   - Selection A2-B1
   - Selection A2-B2

Brief description of the interaction between the two drop-down selection lists: The values in drop-down # 4 are dependent on the value of drop-down # 3 and change dynamically as the drop-down list #3 selection is modified.

Example of the brief written screen requirements used in agile development.

Continued from pg.5

this assessment told the UI developers which browsers and platforms had to be supported.

To collect changing requirements and handle development time limits, the customers and developers met regularly during the user acceptance testing period to address usability issues. We had a system for handling requests that were in-scope versus enhancements that were considered out-of-scope. During acceptance testing, the team considered new features and enhancement requests as out-of-scope, while adjustments or refinements were in-scope changes. The development leads evaluated the in-scope changes to quickly determine feasibility and explain the limitations to the customers. If a change did not immediately add value, it was worked on post-release or shelved. Out-of-scope changes were kept for consideration during the next release cycle. Developers also quickly flagged potential schedule busters as soon as they appeared.

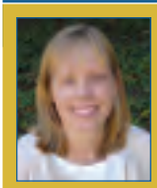
End Results

There are many benefits of integrating UI work into rapid development cycles. First, as we worked on multiple applications with similar functionality, we could re-use designs that had already been proven to be easy to use. Building on this foundation of user-tested designs and a standards document, there was less churn for developers and users during requirements development. Developers could begin coding for common navigation and functionality before the requirements were finalized. The standards let the team and users know what features to expect, where to find them, and how they would work. The team didn't have to worry about novel design elements, non-standard features, or unexpected functionality. The standards gave the suite of applications a consistent look and feel. Thus, when a user group received a new application, they needed little or no training.

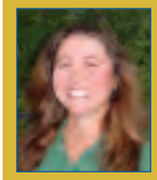
However, there were some limitations to this design approach. First, because of the templates, it was hard to incorporate new functionality or design into our short development cycles. Additionally, it was not easy to swap out engineers, as there was a long ramp-up time for new UI systems engineers. The long initiation period was required because each of the teams had grown its own communal project knowledge (mostly undocumented) over months and years of working closely together. In addition, as UI systems engineers working with the systems engineers and developers, we learned a considerable amount about the strengths and limitations of the systems and code design for each of the applications. This knowledge informed our UI designs and conversations with customers and team members. New team members needed to learn this information through many conversations and on-the-job training.

Nevertheless, the benefits of working within the constraints of agile development far outweighed the limitations. We recommend our approach to other teams working with short cycle times. UX

ABOUT THE AUTHORS



Gwen Pearson is a senior technical specialist at AT&T Labs in New Jersey. She works on applications and websites to provide life-cycle human-factors support. Gwen's favorite hobbies are reading, watching movies, and cooking.



Susan Pearsall has enjoyed a variety of user experience projects from public interactive sites to internal applications. Outside of work, Susan loves spending time with her family and playing at the beach. For more about agile development, see the UPA website at http://usabilityprofessionals.org/upa_publications/user_experience/current_issue/index.html.