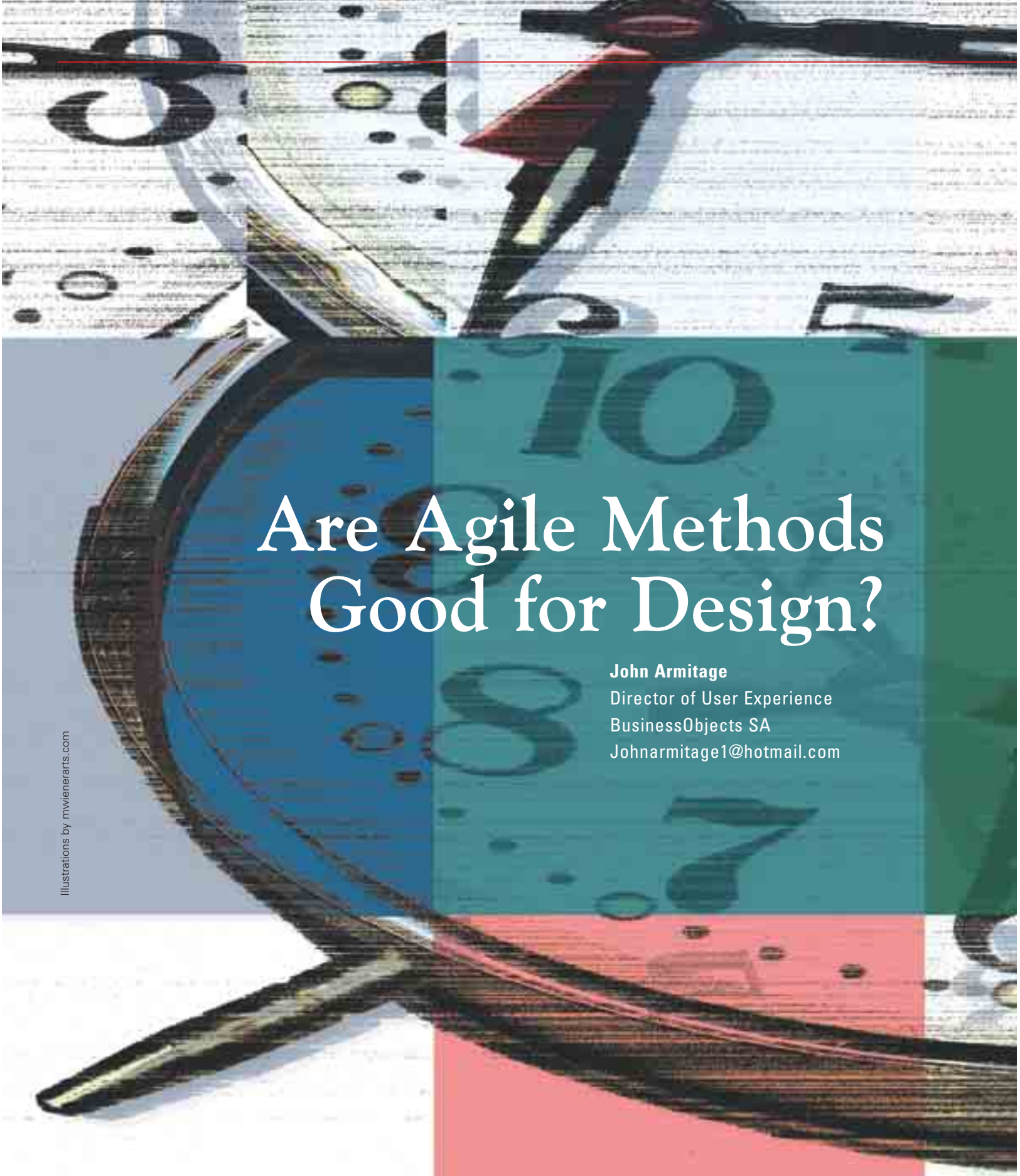

Are Agile Methods Good for Design?

John Armitage
Director of User Experience
BusinessObjects SA
Johnarmitage1@hotmail.com

Illustrations by mwienerarts.com

In a recent series of consulting projects, I served as the designer for a software development team working with agile development methods, widely known through the most severe of these methods, Extreme Programming, or XP. The most salient aspect of the agile methods movement [1] and of XP is emphasis on rapid, iterative releases of working products at the expense of sophisticated planning and design.

XP Guidelines [2,10]

- Complete product pieces sequentially, in relative isolation of one another.
- Refactor the product periodically to ensure that the pieces fit.
- Apply efforts to product construction versus representations of the product.
- Create little or no documentation or specifications.
- Keep code and design simple and easy to change.
- Break down complex features until manageable.

This seemingly anti-design approach is anathema to most designers, whose very existence depends on representing what is to be built. Although my experience was frustrating at times, I tried to keep an open mind and was curious to learn why agile methods are gaining popularity. I came away with insights about software development and the design process that spurred me to write this article.

I believe that agile methods, however threatening or ridiculous they may seem, can benefit design. To understand how, let's briefly go back to the drawing board...

A Product Development Metaphor

As a kid learning to draw, I would often indulgently render small parts of a drawing in rich detail, and pay less attention to how these parts related to each other. Lacking an overall spatial plan, I often ended up with gross inaccuracies that ruined the drawing.

My instructional books recommended starting with a light sketch to plan the composition, and then to gradually commit to areas of light and shade over the entire surface to evenly develop the composition. Maintaining such a balanced overview of the drawing would reveal planning flaws sooner and allow their correction. Initially I resisted drawing these lines that would later need to be erased, but the advice eventually worked; results became more predictable and the eraser became my friend. I still use this fundamental approach to design complex software systems, and in fact it is ideal in most formal design instruction.

For digital product development, these drawing stages roughly correspond to the product's migration through three basic forms or phases: requirements, specification, and builds resulting in the final product. Commercial development processes, however, are rarely ideal. Many approaches have been tried, and all involve representing the end product in these forms and in this order. Each form represents a "current version" of the product expressed in different media and detail.

Agile Methods

Agile methods are unique primarily in their approach to the quantity, scale, and frequency of these phases. Agile projects are implemented more or less as a series of small projects (called *stories* in the XP lexicon), strung together into a larger, ongoing project. Each story has its own requirements, specification, and product phases.

Acknowledging that large, long projects often change course during development, agile methods seek to produce finished, working, reliable code for what has been worked on. These results can be more valuable to a customer than an unfinished, poorly written, and unreliable shadow of an initial, overall design.

The agile movement was instigated and pioneered by software developers in reaction to a frustrating history of projects being delayed, going over budget, and eventually collapsing under their own weight [6]. Tired of working stressful jobs with long hours on doomed projects, the founders blamed volatile requirements and efforts to create sweeping, ambitious, improperly targeted product designs before any real code had been written.

Even though development models had matured from being primarily ad hoc (little or no structured process) to waterfall phases (sequenced handoffs from discipline to discipline) to more interdisciplinary blended phases (e.g., having designers advise on the requirements and developers advise on the design), project results were still disappointing. Ad hoc processes typically resulted in chaos. Waterfall processes, and to a lesser degree blended phases, experienced trouble and delays because disciplines made incorrect feasibility assumptions or acted in absence of a shared product vision, or both.

If large projects could be reduced to a series of mini-projects that are started, finished, tested, and delivered to the customer in quick succession, perhaps risk could be minimized. Customers could provide feedback on delivered work to inform the next mini-

project. Iterations would be built on one another over time, resulting in software systems that "grow" into existence while actually being used, versus being "determined" ahead of time. Iterations would be tested for reliability. In the end, customers would be able to get more of what they wanted faster, because they would have had more flexibility with the requirements.

The risk of this approach is the inability to model ahead of time what is to be built (to design) and ending up with a well-crafted and reliable final product that lacks a coherent structure and vision.

XP may appear to be merely an ad hoc process with a catchy name, particularly in its disregard for prescriptive design in favor of building what the customer says they want or need next. Many designers have experienced the ad hoc treadmill and work hard to instill planning and discipline in the completion of complex projects. Agile literature, however, purports highly structured, disciplined methods (not always extending to professional practice). XP seeks to break down complex features into the simplest possible forms and quickly build them. If the result works and the customer likes it, the feature is kept and perhaps enhanced. Below is a simple illustrative example:

A. In a standard scrolling list, you can:

1. Select one item
2. Select multiple items
3. Select discontiguous items

XP might implement a scrolling list allowing only item 1. Although items 2 and 3 are plausible functionality extensions, having item 1 released sooner and more reliably could provide more value.

Ad hoc development. Large structural errors are possible.

Staged progression from low to high fidelity.

The ideal XP result: a series of finished pieces assembled serially into a cohesive whole.

One risk of phased processes: Design is accurate but implementation is poorly crafted or unreliable.

Another risk of phased processes: Design never ends and implementation is never finished, as in Ted Nelson's Xanadu [8].

The risk of the XP approach. Individual pieces have value, but the larger system is disjointed.

The answer: Split your time investment between overall design and detailed, iterative development.

A key piece of XP that allows it to make the pieces fit together and adapt is called refactoring. Refactoring is XP's eraser; it involves taking time out from adding new features in order to rework and integrate what has been done. It's a bit like cleaning up the mess after a long run of construction (or doing redesign after much has already been built and learned).

XP resembles iterative design, in that both processes use short cycles of output and feedback. A major difference, however, is that while iterative design typically seeks to model, assess, and revise larger systems at low and high fidelities, XP builds and releases smaller systems strictly at *extremely* high fidelities.

To professional **designers**, **agile methods** are likely to be **threatening**.

Implications for Design

To professional designers, agile methods are likely to be threatening. In agile proponent Martin Fowler's essay "Is Design Dead?" [4] his rhetorical answer is actually "no." However, his use of design refers to technical design versus user experience design [5]. In fact, the agile community rarely mentions users or user interfaces at all, which means that either they neglect the user experience or are focusing on projects with less need for sophistication in user experience (UE).

Whether it occurs in a large, speculative, upfront effort, or in small, steady steps that grow into a coherent product, design needs to occur nonetheless. Perhaps the main benefit provided by UE designers is the ability to imagine and represent how a disordered set of functions and meanings can be synthesized harmonically. Unfortunately for designers, XP relegates the design role to short-term efforts to design and redesign small, isolated elements. Less severe agile methods, however, can benefit from both the designer's vision and the practicalities of XP.

Agile Project Experience

In a series of three client-server software development projects, our consulting team (two designers, one project manager, one analyst, two to four developers) worked with a client-driven agile process that featured one- to two-week release cycles. The products were a touchscreen-based, media search-and-play system for general recreational use and two related

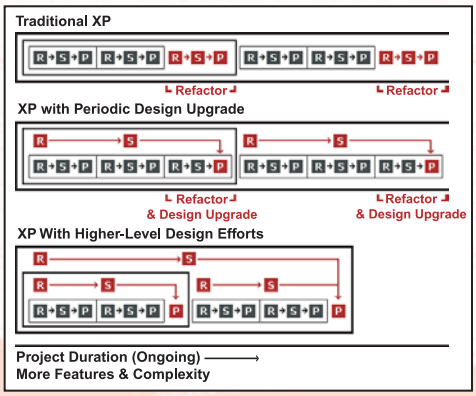

Figure 1. Multi-layered design efforts in a six-story agile project. Dedicated UE design efforts are in red.

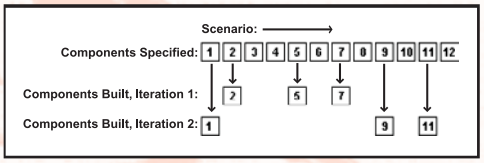

Figure 2. The component planning process.

cataloging applications for use by a small group of media librarians. All the products had high standards for usability and appeal, so we were reluctant to abandon our user-centered design practices.

We evolved a hybrid approach that required design work on three levels, done in parallel. The low-level effort (typical for XP) supported the short-term iterations by providing detailed component designs to drive construction. A second level presented low-fidelity redesigns of existing builds in order to inform refactoring effort. This split duty limited the designer's ability to optimize either effort with just "one pass," but because all design and product assets were flexibly built, it was easy to layer on improvements as the project progressed. In the later efforts, as detailed later, we were able to design on a third level to provide overall product vision (see Figure 1).

Usage scenarios drove our design effort, taking the form of linear, Shockwave-based storyboard depictions of users completing workflows. Storyboards showed how users would use a series of components, or groups of user-interface (UI) functionality (such as a browser tree, a form, or a dialog box), to complete a task. In the storyboards, the design would show typical (not comprehensive) depictions of these components, just enough to gather feedback from users and stakeholders on the design's overall merit. Concurrently, and upon design validation, the customer and engineering team would prioritize components for development and assign them to iterations (see Figure 2).

Components chosen for an iteration were then

designed in more detail—but not complete detail. These components were then assembled into a rough working version of the system for testing. Components were added in later iterations and changed as their designs changed.

A dialog box component provides a simple example of this process (see Figures 3-5). The initial design of a dialog box for creating and editing DVD chapter lists used a custom list display with hairline-rule dividers. Because the effort to build this design would not fit within an iteration, and our development tool (Visual Studio) made such custom work difficult, we used a more standard list control in the first programmed version. The customer (and end users) liked this so much that we abandoned the considerable extra work to build the original design and merely upgraded alignments and colors—and added the button icons. We then worked on other features with more potential value.

This process may sound similar to a standard iterative design process. However, it was different (from what I have experienced) because only one low-fidelity design was prepared, it was not shown to users before being coded, and except for integrating it with the current build, the developer worked in isolation from other parts of the system. The result was deemed "good enough" and the team moved on to the next task.

Although the projects were successful, the UE design effort in the first project was frustrating, partly because the designers joined the project late. Because the product featured complex navigation and interaction capabilities, designing comprehensive alternative system models within the weekly release schedule was impossible. Instead (in a microcosm of how larger products evolve over much longer periods), smaller features were added and tested one-by-one, allowing customer feedback on the design and performance. Feedback would accumulate until the overall system design needed rethinking. To do this, one designer would step out of the weekly release cycle.

After awhile, the designers became frustrated with fixing symptoms of larger problems that could not be solved within one iteration. We felt that we'd designed the project several times over, constantly editing, redoing, and combining elements, primarily because of requirements that were always in flux.

Changing Requirements

The effort to eliminate changes to requirements has always been a losing battle, and always will be. Requirements change because they can, and software's mutable nature (compared to other product types) makes its development more susceptible to (or accommodating of) change. Software has no raw materials or physical structure to scrap, has fewer standards for quality, has low manufacturing and tooling costs, and has very rapid rates of innovation and evolution.

Perhaps it is more important to realize that in many cases the project itself is the greatest influence on the project. The more work that is done on a project, the more the project context changes. Agile methods seek to benefit from the intelligence of experiencing the real product's existence, and the sooner the

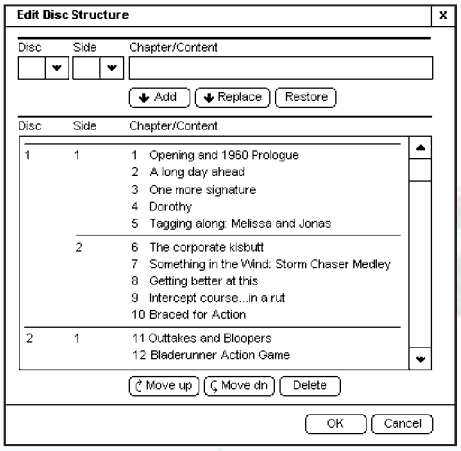

Figure 3. Initial design wireframe.

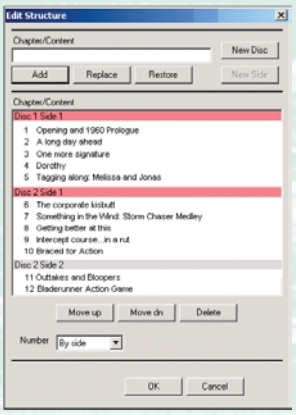

Figure 4. Initial build screenshot.

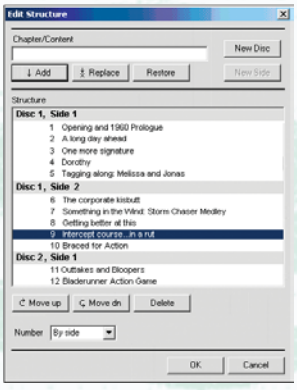

Figure 5. Revised design and build screenshot.

better. Design, conversely, aims to predict what the entire product will be *before* it exists. Proponents of heavy up-front design, such as Alan Cooper [3, 7], claim that adequate product intelligence should reside in a specification. This can be true, but in cases where technology is new or untried, requirements are volatile, the domain unfamiliar, or the complexity immense, it can be too risky to heavily invest in assumptions without adequate "reality checks."

In summary, agile methods exist to mitigate product development risk. They are more empirical than other methods, using trial and error to reduce the risk of building the wrong thing, with the expense of having more routine code rework and having to maintain all development code close to release-quality. Essentially a series of very-high-fidelity design experiments, they achieve low-level certainty by accepting high-level uncertainty.

Why Now?

As a result of lowered software distribution costs made possible by the Internet, it is economically feasible to release new software versions of incrementally greater value at a high frequency. Lower transaction costs, for both development and distribution, are now allowing trial and error to replace some of design's predictive expertise. Also, business relationships in general are becoming more transactional, meaning more deals are done, but at a smaller scale, similarly to the lean sourcing and manufacturing movement [9]. Agile methods also reflect more of a service model versus a product model, with value

delivered in streams versus in bulk. In fact, commercial Web site evolution is a good example of agile methods at work *en masse*. Most sites have evolved as a series of live experiments versus being driven by monolithic plans.

Another point of view, and my personal suspicion, is that XP is a reaction to a shortage of good software application designers, architects, and strategists. Software is difficult to model, and in a way, XP is just a brute-force way for engineers to get work accepted without design and designers.

Guidance for Designers

Despite the identified shortcomings, agile methods can improve the design process. Iterative releases being used by customers, even those having had little design input, can serve as ongoing usability tests. This arrangement also allows automated usage tracking and testing tools to be brought into play sooner and in a more relevant context. Lower-risk release cycles can also encourage design experiments (albeit at low levels), and gone are the ominous specification documents to write. The fast release pace gives an ongoing sense of accomplishment, and because there are closer team interaction, shared goals, and less solitary time invested in elaborate design or engineering schemes, there are less defensiveness and territoriality about individual designs.

Here are some tips for designers working in an agile environment:

- Embrace a larger context and judge success by the success of the team or project (which is perhaps the essence of being interdisciplinary).
- Appreciate that providing partial solutions earlier can be more valuable than providing full solutions later on.
- Design solutions and work products that can easily be changed.
- Learn to design the simplest possible version of your idea, and add to it later.
- Think hard about what to design first and what to leave for later.
- Be willing to throw out what's done if it's not working or if it was the wrong thing to do in the first place.
- Learn to quickly jump from low-level to high-level design tasks.
- Branch out from the build iterations to sketch and model an overall vision, yet still respect the learnings from early technical trials.

I believe that all projects and circumstances are different and require flexibility in process. Perhaps agile methods are best used in cases of exotic technology, volatile requirements, lack of high-level architecture expertise, or lack of high UE standards. XP-like projects could be a good experience for entry-level designers. The fast pace, simple scope, and ability to see designs built quickly could be a good training ground for the basics of interaction and of working with developers.

In a way, designing software is actually not too different from drawing. Advanced drawing instruction, in fact, has students repeatedly create figure

drawings one after another, rendered in impossibly short times such as 15 seconds or less. The goal is to eliminate perfectionism, loosen the gesture, and force the artist to keep their eye on the subject. Having a plan is good, but the best artists have a dialog with the drawing. Agile methods can benefit design when they allow the system, and customers, to talk back to the designer with live experience, and afford the opportunity for the designer to respond.

ACKNOWLEDGMENTS

The author wishes to thank Jack Hakim, Tom Spitzer, David Thureson, and David Rowley for their input to this article and the work it describes.

REFERENCES

1 Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. The Agile Manifesto. Available at http://agilemanifesto.org

2. Beck, K. *Extreme Programming Explained*. Reading, MA: Addison-Wesley, 1999.

3 . Cooper, A. *The Inmates are Running the Asylum*. Indianapolis, IN: SAMS, 1999

4. Fowler, M. Is Design Dead? Available at www.martinfowler.com/articles/designDead.html

5. Fowler, M. and Taber, C. Planning an XP Iteration. Available at www.martinfowler.com/articles/planningXpIteration.html, 4-6

6. Gibbs, W. W. Software's Chronic Crisis. *Scientific American* (Sept. 1994), pp. 86-95. www.cis.gsu.edu/~mmoore/CIS3300/handouts/SciAmSept1994.html

7. Nelson, E. Extreme Programming vs. Interaction Design. FTP Online. Available at www.fawcette.com/interviews/beck_cooper/default.asp

8. Nelson, T. Project Xanadu (http://xanadu.com/)

9. Wheelright, S.C., Clark, K.B. *Revolutionizing Product Development*. New York: The Free Press, 1992.

10. Xprogramming.com: An Extreme Resource. Available at www.xprogramming.com/xpmag/whatisxp.htm

EDITORS

Kate Ehrlich
Collaboration User Experience Group
IBM Research
One Rigers Street, Cambridge, MA 02142
617-693-1170 katee@us.ibm.com

Austin Henderson, Director,
Systems Laboratory Advanced Concepts & Design Pitney Bowes
35 Waterview Drive MS 26-21, Shelton, CT 06484
203-924-3932 austin.henderson@pb.com