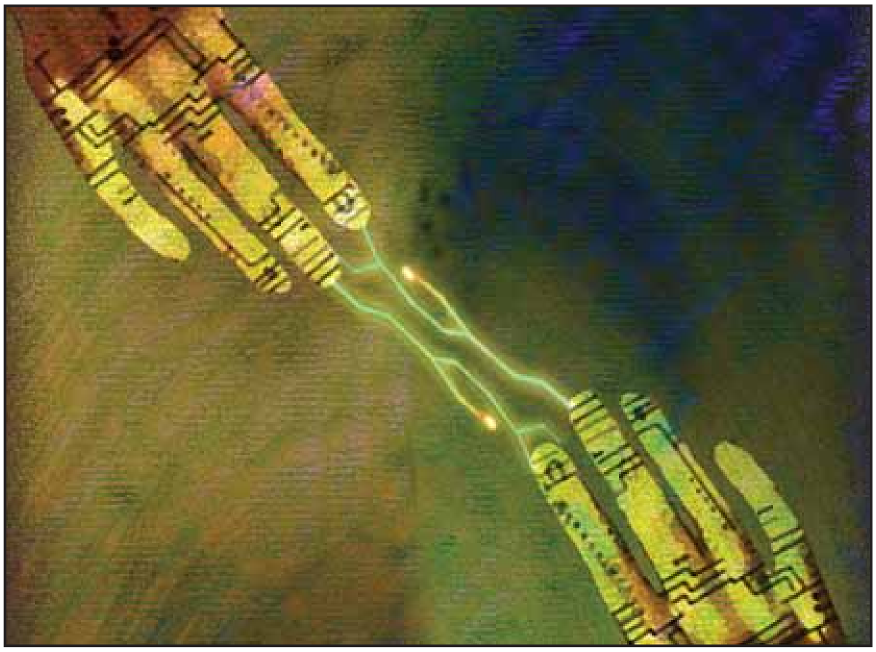
Bridging the Gap Between Disciplines

by Jeff Patton and Brian Marick

Collaboration is hard. Two people from different disciplines will often find history getting in the way. Their two disciplines coalesced from different experiences, drew different lessons, emphasized different values, and focused on different goals. All those things hamper coordinated work.

The two of us recently had a Eureka! moment after Jeff realized how a bridge could be built between two disciplinary worlds. This article isn't about the particulars of that bridge (but you can find out about it in the StickyNotes). Rather, it's about what factors led to the creative moment. Although this is a story about two consultants at a meeting in the mountains, we believe you can take advantage of the same factors in your workplace. We emphasize the factors by putting them in bold font and italics.


GETTY IMAGES

The Characters

Both of us have ***common software development preferences.*** Rather than planning carefully for the future, we prefer to remain nimble enough to be unfazed when the future brings surprising change. We believe that most businesses can't know what they'll need in a year, and we *know* we can't predict what we can deliver in a year, so we arrange to give the business a working piece of the product every few weeks. We don't know anyone who is very good at writing or reading precise, unambiguous, clear, and complete documents, so we try to get everyone together in the same room so that no one has to. Unsurprisingly, we're advocates of the Agile methods.

We also have a ***discipline in common.*** We're both competent programmers, and we're abreast of what's new in that field.

Nevertheless, our ***primary disciplines differ.*** Jeff is mainly known for User Experience (UX) design, the craft of shaping a product according to users' goals, contexts, tasks, and quirks. Brian is mainly known for software testing.

The Setting

At the point our story starts, we were ***isolated from normal work*** in the Canadian Rockies, leading a Canadian Agile Network ***workshop devoted to the question of what could emerge if two disciplines were bashed together.***

We faced some ***tight deadlines.*** We both were in the unusual (honest!) position of being fairly unprepared for the workshop. So we spent the afternoon and evening before the first day—and scattered hours throughout the workshop—obsessing over what should be delivered and how best to deliver it. That resulted in ***high energy focused on the topic:*** We didn't have time to be distracted by email, a good night's sleep, and like irrelevancies. The energy and deadline pressure continued past the workshop, as we'd promised to report on our discussions at a user group meeting the next evening.

The Plot

The story opens with ***an obsession*** of Jeff's: How should UX people fit into Agile projects? They seem to be needed. At the end of six months, delivering new features every few weeks, it's too easy to end up with a pile of features that don't hang together in any useful way. And yet the usual UX solution would break the rhythm of the project. That solution has the UX designers disappear from sight of the project proper, collect a great deal of data on the users, iterate through many ideas, and then deliver a large document describing in detail a tailored interaction design. (See the StickyNotes for more on interaction design.) Jeff came to the workshop wanting to hone his ideas on how UX designers can work *with* the project, *throughout* the project, helping a good interaction design emerge in the same way that Agile projects work and rework the internal code design.

The heart of our days involved ***demonstrations of sequences of well-understood techniques.*** Jeff had groups of users walk through the creation of user goals and metrics, sketchy user profiles, and a rough task model used for release planning. (See his article in the January 2004 issue of *Better Software* for more information.) Brian gave a detailed demonstration of a technique he learned from Rob Mee. In it, tests written in business language are implemented in the Fit testing tool, the tests are made to pass one bit at a time (as with conventional test-driven design, just at a "higher" level), and business objects are extracted from the resulting test-support code. In essence, the tests drive the large-scale structure of the application. (See the StickyNotes for more on test-driven design and Fit.)

Watching this, Jeff saw three conceptual strands making a possible bridge between business-facing test-driven design and UX design:

1. The *terminology strand* links what UX people point to when they say "interaction context" to what a programmer points to when she says "the Presenter object in the Model-View-Presenter design pattern."
2. The *artifact strand* links the script a UX designer might use to explain workflow and navigation to the test a Fit-first programmer would use to create program objects (specifically, Presenter objects). They could in fact be the same chunk of words.
3. The *activity strand* lets the UX designer who has created an interaction context walk over to a programmer, show an executable example (test) or two, show a sketch of visual experience, have a little conversation, and get the programmer well started. Suddenly, the UX designer is part of the normal rhythm and smooth pace of an Agile project.

Jeff got there by *putting terminology in the background*. Consider: If you ask a UX designer what an "interaction context" is, her definition will be in terms of words interesting to UX designers. Perhaps she'll say, "An interaction context is a 'place' in the software where similar user roles perform similar tasks at similar times." Such a definition would be more likely to prompt you to ask questions about "user roles" and "tasks" than questions such as, "What's the closest thing to an interaction context in my world?" (After all, the UX designer isn't *in* your world.) So your questions will lead you to understand the self-contained definitional world of UX designers. It helps you know where to build a bridge *to*—but not how to build it.

For Jeff, the idea about how to build it came from watching Brian *perform specific detailed actions with a specific artifact*. That kind of observation encourages learning by analogy: "What in my world looks like that test script?" and "If I were working through that list, step by step, what would I hope to accomplish?" Those analogies build the bridge: "I'd work through a list like that to understand or explain an interaction context, while he'd use it to understand and create an object. What kind of object would match an interaction context? A Presenter . . . So how could an interaction designer write a script like that to force a programmer to create a Presenter?" From such observations flow *questions that draw out the representative from the other world:* "This is what I think I saw . . . does that sound reasonable to you?" And so the bridge is built—or at least sketched.

A sketch has to be tested. We spent the first part of that evening taking up space at the Saigon Y2K restaurant in Calgary. We worked through a simplified example of user experience design for a nagging problem Brian's wife has at work. After they kicked us out of the restaurant, we went back to the hotel and worked on writing a script/test and coding from it until they ejected us from the hotel restaurant. *Working through a realistic example helped us refine the insight.* We actually worked through it three times, once in private and twice more the next day in two talks. Each time, we learned more about the idea.

What Can You Do?

The word "Eureka!" is in the English language because good ideas often come as a surprise. But they don't come from nowhere. They depend on people, setting, and action. Although what we did is surely not the only way to surprise yourself, consider trying the following when you have a knotty problem:

- Gather together people from different disciplines—people who nevertheless have enough in common that they are both willing and able to understand each other.
- Make sure that the people have obsessed over the problem long enough that their brains are primed to make use of any stray fact.
- Put them somewhere without distraction. Give them enough time to think—but not so much time that they lose urgency.
- Have them show each other what they do. Ask them to seize on specific similarities in actions, artifacts, and definitions—and then to explore them deeply.
- When they have some piece of a solution, they should put it through its paces. If it survives, bring it home to try out in a real project (as Jeff will be doing). **{end}**

Jeff Patton leads teams of Agile developers to build the best software possible. He proudly works at ThoughtWorks. Jeff's series of columns on software design and pre-design tips appears each quarter on the StickyMinds.com homepage.

Brian Marick has worked in testing since 1981. He concentrates on developer testing, the interface between developers and independent testers, the criteria for objective test evaluation, and helping teams and projects understand and manage the tradeoffs inherent in software assurance. Brian founded Testing Foundations: Consulting in Software Testing in 1992. He is the author of The Craft of Software Testing *(1995) and is a technical editor for Better Software magazine. Contact Brian at marick@testing.com.*

Sticky Notes

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

- The Eureka! idea explained
- Places to learn about interaction design
- Places to learn about test-driven design
- Further reading on Fit