DON'T JUST BREAK SOFTWARE. MAKE SOFTWARE.

How storytest-driven development is changing the way QA, customers, and developers work.

by Tracy Reppert

STORYTEST-DRIVEN DEVELOPMENT (STDD) IS AN EXTREME PROGRAMMING PRACTICE. **The basic premise is that before any code is written, a team takes a story (or rough idea of a requirement) and fleshes out that story by producing an executable "storytest." Opponents say that STDD is "cowboy coding," or "snake oil," or**

STDD rates high with the development team at Nielsen Media Research. Standing (l to r): Thomas Hund, Praseed Thapparambil, and Pam Smoot. Seated (l to r): Cheryl Redding, Rhoda Foxworthy, and Merrilee Albright

"Creating the expected results was quite challenging, but it was rewarding when they ran."

—MERRILEE ALBRIGHT

just "a hindrance to real work." Proponents argue that STDD produces the simplest system possible and is the wave of the future, the new frontier.

EVERYONE GETS A SAY

With traditional acceptance testing, developers would create an entire system, declare it finished, and then submit it for acceptance testing. This mostly manual testing tended to get bogged down in paperwork and occurred so late in the development lifecycle that any mistakes were very expensive to fix. Even if the system met all of the stated requirements and passed through acceptance testing with flying colors, the customer, who hadn't seen the system since the requirements meeting, often was left feeling like he had not gotten what he wanted. STDD makes it possible to formalize the expectation of the customer into an executable and readable contract that programmers have to obey if they want to claim a working system.

"STDD gets the right people collaborating at the right time," says Joshua Kerievsky, XP coach and founder of Industrial Logic, Inc. STDD brings together the customer, developers, and testers before any code has been written. They collaborate to identify a specific piece of functionality, or "story," to be worked on. Customers and testers then specify criteria to validate that the story works and create an executable document that can be accessed by anyone on the team.

"When testing is done after the fact, customers simply say that the system is broken when it doesn't meet their requirements," says Somik Raha, XP coach for Industrial Logic. "With STDD, there is a specific contract for acceptance, so discrepancies are quickly resolved. There's a clear context for customers and developers to have a conversation and weed out misunderstandings. The risk of building the wrong system is much lower."

Ken Auer, president of RoleModel Software and one of the earliest practitioners of STDD, agrees. His team, along with Ward Cunningham, first used STDD to produce a software system in 1999. "STDD made it easier to nail down what our end-users wanted," recalls Auer. "By having executable requirements for each story, we were able to know when we had completed stories, which made project tracking a whole lot easier and better."

NO ONE HAS TO BE THE BAD GUY

Raha explains that not only does STDD relieve stress; all the involved parties love the process. "Customers love it because of the new level of confidence they get—instead of just hoping the development team has understood the requirements correctly. Developers love it because there's no way they can be accused of

A GLIMPSE OF STDD IN ACTION.

by Brian Marick

STORYTESTS AS PROPS FOR CONVERSATION

In order to estimate a story, programmers have to understand what the business expert wants. They learn this through face-to-face conversation. The business expert describes the feature, explains why it's useful, and gives examples, perhaps scribbling them on a whiteboard. The programmers ask questions, split the feature into constituent parts (smaller tasks), ask more questions about those tasks, and sometimes propose variations on the feature that might be simpler to implement.

EXAMPLES are particularly useful because they ground the conversation in something concrete and keep important details from being lost in abstract discussions and vague generalities. To illustrate, let's suppose a business has a rule that, for accounting purposes, all months are treated as having thirty days. (Some bonds are valued under a similar rule.) One of the stories might be "Implement the 30-day rule." The business expert might write something like this on the whiteboard:

1 April to 1 May is 30 (as normal)
1 Feb to 1 March is 30 (even though 28)
1 Jan to 1 Feb is 30 (really 31)

QUESTIONS should follow. A programmer or tester might ask if it's really true that the number of days from 1 January to *both* 1 February and 31 January is thirty. The business expert would answer that it is. Then someone else might ask, "So, it's twenty-seven days from 1 February to 11:59 PM on 28 February, but the next minute it's thirty days?" The business expert might start to answer "Yes," only to be interrupted by someone saying, "Not if it's a leap year," which would lead to a discussion of the relevance of leap years, and then, perhaps, to whether time zones matter.

That discussion might well be cut short, because the point here is for the programmers to have enough information to estimate accurately, not for them to know every last detail.

building a system that is not acceptable. And QA loves it because programmers aren't viewing them as the bad guys," says Raha.

STDD challenges the notion that QA has to play a rival role to a programming team if they expect to be able to spot defects. In practice, teams find that they are saving a lot of time by having QA help produce storytests from the beginning. "A tester is sometimes viewed as an adversary, when all they are trying to do is ensure that the delivered product functions well," says Merrilee Albright, lead SQA analyst at Nielsen Media Research and member of an STDD team. "I've been testing a long time, and have found that my role on this team definitely has more respect than it might in the normal software development process. I can utilize the expertise of other team members to help me create tests. Having the customer available to review the test results and make suggestions immediately has proved invaluable. Creating the expected results was quite challenging, but it was rewarding when they ran and ensured that the system was functioning as expected."

THE FIRST STEP CAN BE A BIG ONE

If STDD is so great, why do so many people distrust it? Perhaps some of the skeptics of STDD haven't experienced it since its inception. XP and STDD definitely had some growing pains. Joshua Kerievsky explains, "In the early days of XP, we did a lot of unit test-driven development and would automate storytests only after we'd written our code. The trouble with this approach was that it was often hard to get subject matter experts to define storytests, which meant that with successive iterations we would get further and further behind on the number of storytests we still needed. That produced higher defects, since the lack of sufficient story details led to code that failed to fully satisfy customer expectations." He adds that STDD now helps teams obtain the necessary amount of story details when they need it—before writing code.

Even now that most of the quirks have been worked out, people may still be wary of such a big change. "It was difficult to trust the process in the beginning," says Pam Smoot, senior project manager at Nielsen Media Research, which began implementing STDD for a major data-warehousing project, "but it's so much better than what we used to do. Having expected results created upfront by a business/QA person helped us find miscalculations, misinterpretations, and miscommunications right away rather than late in the project."

"It was difficult to trust the process in the beginning, but it's so much better than what we used to do."
—PAM SMOOT

Skeptics begin to see the value of STDD in just a couple of weeks, according to Smoot, though it can take a few

CONVERSATIONAL EXAMPLES BECOME STORYTESTS

Once the iteration is planned, it's the job of the programmers to produce features that satisfy the business expert, and it's the job of the business expert to support them—mainly by being available for further conversation. For example, a programmer might realize that she doesn't know what the program should do in some special case, so the business expert has to tell her. Or a programmer might realize that the code is so structured that it would be trivial to add a new wrinkle to the feature, so the business expert should tell her if the new wrinkle is a good idea.

The business expert should also help with the tests. The table at right shows a starting set of tests that came up in the imaginary planning meeting we mentioned earlier. (Ignore the first row for now.)

Who writes that table? It's entirely up to the team. Some business experts might be happy writing such tables themselves. In other cases, it might be dedicated testers or the programmers. What's important is that the tests get written efficiently and that the business expert believes that seeing them pass will increase his confidence in the program.

This set of tests needn't be complete. It need only be enough for the programmers to get started. As they work on getting those tests to pass, other people can continue writing tests. The test-driven style is to make tests pass one at a time; while working on one test, the programmer doesn't worry overmuch about what tests remain. So it doesn't matter if some of those tests are not yet known. These storytests do not necessarily completely replace conventional testing. They're designed to build quality into the product. Someone else still needs to create tests to detect where the storytests fell short of the goal of preventing bugs.

> continued

CalendarFixture			
from	**to**	**elapsed()**	**actual()**
1 Apr 2003	1 May 2003	30	30
1 Feb 2003	1 Mar 2003	30	28
1 Jan 2003	1 Feb 2003	30	31
1 Jan 2003	31 Jan 2003	30	30
1 Feb 2003	28 Feb 2003	27	27
1 Feb 2004	29 Feb 2004	28	28
1 Feb 2004	1 Mar 2004	30	29

"There's really no situation in which I can see STDD not being the best process to use."

—THOMAS HUND

months to fully absorb it, explains her fellow team member, product manager Rhoda Foxworthy. "Now everyone on the team really sees its value and is excited about taking it outside the team, although we're not sure how," says Foxworthy. "Our next big challenge, when our platform team is retired, is continuing its spirit and use in a production environment where there are walls and naysayers."

CHANGE IS HARD—BUT WORTHWHILE

Many STDD practitioners encounter an ironic reaction to their success: resentment from the rest of the organization. "It's weird that that happens, but it does," says Ward Cunningham, whose FIT tool is typically used for STDD because team members with diverse backgrounds can understand it. Cunningham continues, "It's just one of those quirks of human organization. Attitude is based on experience, and eventually people shift their attitudes and realize this method is to their advantage."

Considering such human quirks is key to successfully integrating STDD. "People always resist paradigm shifts," says Raha. "So it's especially important to introduce STDD when a project has just begun and people are still motivated."

As with any XP practice, keeping the STDD process going can be challenging if it's considered to be merely an option and not a rule. "STDD should be followed as a policy from the top down," says Narkeeran Narasimhan, a testing specialist for Ansys Corp., which develops engineering simulation software and has been doing XP since 2002. "Over a period of time, these processes can get compromised if, for example, new developers don't want to learn a method, and their managers—not wanting to upset them—don't want to enforce it."

The key to making the change is just getting people to try things, explains Cunningham. Developers who do try STDD tend to love the influence it has on a software design. "When I started doing STDD, I was amazed at how much simpler my code became, versus the code I'd written using either upfront design or minimal design with unit test-driven development," says Kerievsky. "Following STDD closely enabled me to produce designs I simply would not have anticipated."

"There's really no situation in which I can see STDD not being the best process to use," says Thomas Hund, senior SQA analyst at Nielsen Media Research. "As well as it's worked for us, I'm a believer." **{end}**

Tracy Reppert is a freelance journalist who has written about technology for the New York Times, *the* San Francisco Chronicle, PC Magazine, *and* Wired. *She lives in Berkeley and can be reached at tracy@industriallogic.com.*

> *continued*

STORYTESTS BECOME CODE

Making a table of storytests pass requires two steps.

THE FIRST STEP is to write a *fixture* that translates the table into executions of the program. (In the case of the example in the sidebar on page 21, the fixture is named by the first row of the table: CalendarFixture.) Usually, the fixtures do not drive the graphical user interface. Instead, they go "below the GUI" and make calls to the business logic of the program in the same way that the GUI does. This makes tests more maintainable, and it also helps to align the language of the program with the language of the business. (If business people talk of buying bonds or negotiating leases, it makes sense for the code to use words like "bond" and "lease" and "buy". This style of programming is called "domain-driven design," after Eric Evans's book of the same name.)

THE SECOND STEP is to make the test pass. In some cases (like our calendar example), that's straightforward. The programmer runs the table, sees that a row fails, changes the code, runs the table again to see that the row now passes.

In other cases, the changes needed to make the tests run are too large for that. Fans of test-driven design like to run tests often. They don't like to make many changes before gaining the reassurance of seeing a new test pass and all previous tests continue to pass. So they will break their coding task into smaller pieces. They'll determine a chunk of code that needs to change, write a programmer test that describes the change (using a programmer testing tool like JUnit or NUnit), make the change, and check it. They'll proceed that way, test by test and chunk by chunk, until the larger storytest passes.

This process continues until all the tests for the story pass. Notice that these tests may change at any time before the end. Any one of the business experts, testers, or programmers may add or change a test if it helps complete the story more surely, quickly, or smoothly. However, the business expert is the final arbiter over whether a test is a correct example of the feature. When all the tests pass, the story is done, and a new story can be started.

Brian Marick has worked in testing since 1981. He is the author of The Craft of Software Testing, *and is the technical editor for* Better Software *magazine. Contact Brian at marick@testing.com. Read other writings at testing.com. (Author's note: Cheryl Redding and Praseed Thapparambil of Nielsen Media Research, and Chris Wheeler of Sciex contributed to this series of sidebars.)*

GETTING STARTED WITH STDD.

So how does a team get started doing STDD? For Nielsen Media Research, changing to STDD was part of the shift to XP. First, the main players and decision makers got together and held a two-day overview boot camp. Once they had senior management's approval, they formed the group and moved into an open workspace soon afterwards. Next, they bought books on XP for all team members and held a five-day intensive XP boot camp. They retained XP experts as trainers and coaches for the first four months. Their QA manager continuously found and fed them pertinent articles. Finally, they held a two-day intensive retrospective that delved into the pluses and minuses of their first thirteen weeks.

XP coach Joshua Kerievsky offers these specific steps for getting started with STDD:

1 Define Stories

Work with subject matter experts and analysts to define stories: roughly defined chunks of functionality. We typically use index cards for stories. Good story headings are no longer than five words and use the active voice. For example: "Book a Room," "Calculate Capital for Term Loan," or "Export Risk Report into Excel" are good story headings. The details of a story give enough information for programmers to make decent time estimates for implementing stories. For the story "Book a Room," the details might be "Allow user to make a reservation for a single room, assuming no special rate discounts or late checkouts." For more details on how to write good stories, see the new book *User Stories Applied* by Mike Cohn.

2 Define Storytests

Once stories have been defined, you can begin writing storytests. This is best done through the collaboration of subject matter experts, testers or QA persons, and programmers. A storytest helps verify that a story works correctly by documenting what inputs are supplied to a system and what outputs are expected. A document that contains a storytest can include other useful information, such as sentences to describe a story or a description of a formula that is being exercised by a storytest.

It takes some time to learn how to best model a storytest in a tabular form using Ward Cunningham's FIT framework (http://fit.c2.com). "You need a day or two, unless you've never done Java," says Cunningham. Ward's site offers a number of open source sample applications that let you practice without getting confused with your issues at work. Cunningham recommends trying to understand the music player, use the tables there to make it do something new, and then apply it directly to your work.

Once you have enough experience with that, decide whether to use a Column Fixture, Row Fixture, or Action Fixture. In most cases, whatever tabular form allows you to present the storytest in the most readable format is probably best, because the main audience for a storytest is nontechnical people. In practice, the easier it is to read a storytest, the more work programmers will have to do behind the scenes to map the storytest to the classes in the production code. That's okay because once programmers begin to write such mapping code, it will become increasingly easier to do so for future storytests.

3 Check In Storytests

It is best to check in storytests to your version control software as soon as they've been created. We do this as soon as a storytest has been defined. When developers integrate their code, they ought to see that new storytests have been checked in. This informs them that they can begin working on the story(ies) associated with the checked in storytest(s). In general, programmers should not begin programming a story until they have a storytest that is failing.

4 Program the Storytests

Once developers have a storytest, they will begin writing test code to show that the storytest is failing. If the team is using FIT, this involves defining the appropriate subclass of FIT fixture class and then delivering production code to make the storytest pass. During this time, it is typical and optimal to do unit test-driven development in conjunction with STDD. This helps you to evolve production code that has sufficient test coverage. The combination of these testing techniques leads to fine-grained unit tests for classes and medium- to large-grained storytests for the features of the system. Throughout this process, programmers check in code, unit tests, and storytests when they are passing.