Taking A Tour Through Test Country

A Guide to Tours to Take On Your Next Test Project

By Michael Kelly

"Exploratory software testing is a powerful and fun approach to testing," says James Bach. "In some situations, it can be orders of magnitude more productive than scripted testing."

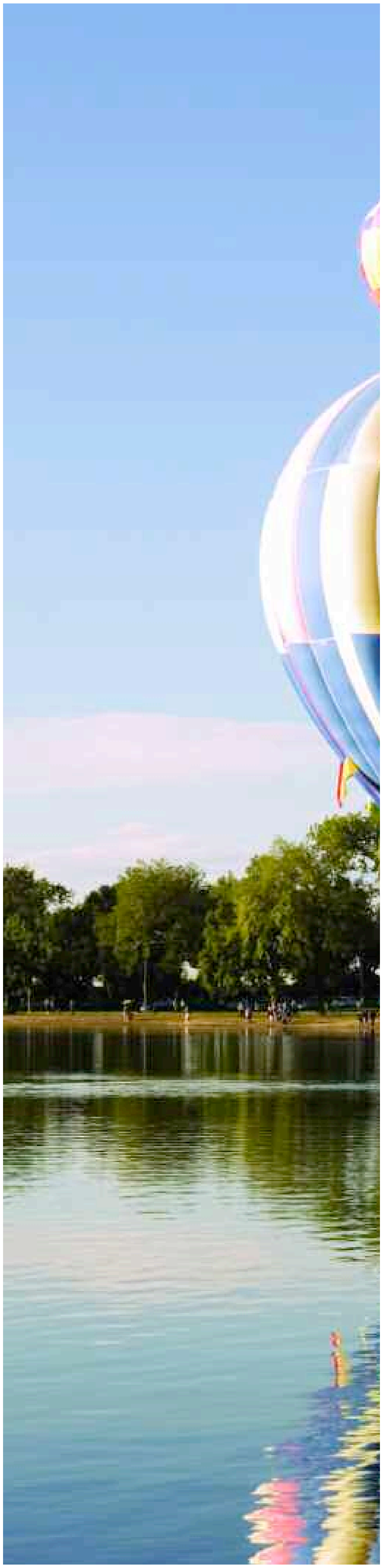
In his article "Exploratory Testing Explained," Bach defines exploratory testing as simultaneous learning, test design and test execution. I find this to be a useful definition, not only in practice but in explaining how I perform my exploratory testing. It divides my thinking into three parts: how I learn about the software I'm testing, how I generate test ideas and design my tests, and the mechanics of how I execute my testing. I find this division useful because it makes exploratory testing easier to explain, but also easier to practice and learn.

In this article, the focus is on the "simultaneous learning" part of the exploratory testing equation. I find that there are many articles and books available on test design and test execution, but few that focus on the problem of how we actually learn what the product does. This is an important step in the testing process, and in this article I will share an application touring heuristic I find useful when attempting to learn about the product. As you read this article, remember that the *touring* concept presented below is an approach to exploratory modeling, but it's not a bad thing if you suspend the tour and actually do some testing.

Getting Started With Application Touring

Earlier this year, while learning from Bach, I had a hard time getting started with a sample application he had given me to test. When he saw me struggling with where to start, he offered me some simple techniques for getting familiar with the product. He referred to the techniques as "tours" of the application.

Michael Kelly is an independent consultant who focuses on test automation and exploratory testing. He's a director at large for the Association for Software Testing and is the director of programs for the Indianapolis Quality Assurance Association. You can e-mail him at Mike@MichaelDKelly.com.



They are called tours because you are not necessarily looking for problems. You are simply learning the product. For a tour to become a test, you would have to have an oracle (the principle or mechanism by which we recognize a problem) of some sort.

The first tour he suggested was the **feature tour**. In the feature tour, you move through the application, getting familiar with all the controls and features you come across. You ask simple questions like, "What's this and what does it do?" and "How would I know if this feature is working?" You look for interactions, calculations, transformations, multimedia and error handling. When taking a feature tour, it can be helpful to look at one factor at a time.

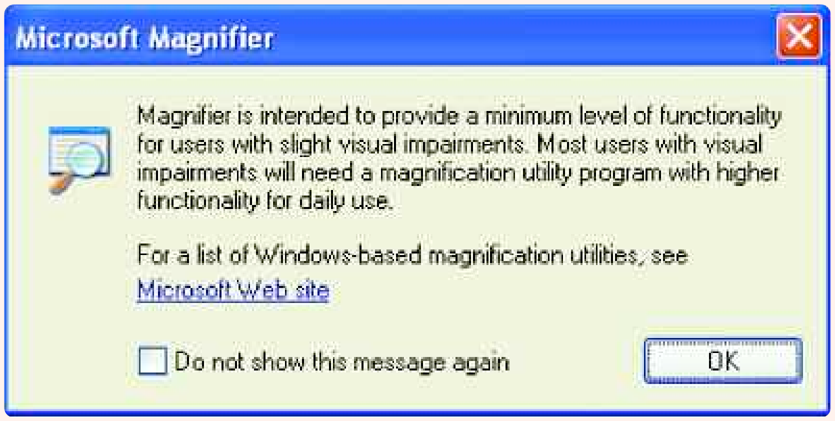
The second tour was the **variability tour**. In the variability tour, you look for things you can change in the product—and then you try to change them. Click buttons, select values, change settings and so forth. The goal is to try to get a feel for how things work and what possible values might be.

For excellent examples of applications that have a high degree of variability, check out James Lyndsay's black-box testing machines at www.workroom-productions.com. At one point in my training, Bach had me tour Puzzle 1 (one of the black-box testing machines made up of five buttons and three sliders). At the end of the tour, Lyndsay asked me to explain—in detail—the machine's behavior. If you can do that (and I struggled), you understand variability. Puzzle 5 seems especially diabolical, if you ask me.

The third tour Bach suggested was the **complexity tour**. In this tour, one attempts to find the five most complex things about the product. Complexity can exist around features or data. Complex features can be the most common features in an application (the algorithm behind Google search) or they might be rarely used—hidden away waiting to cause problems (end-of-year processing for an accounting system).

These three tours enabled me to jump into the product without becoming overwhelmed. As I toured the application I was able to learn about how it worked, I started identifying and prioritizing risks, and I was ultimately able to start identifying and designing tests.

FIG 1. MICROSOFT MAGNIFIER INITIAL DIALOG


Microsoft Magnifier

Magnifier is intended to provide a minimum level of functionality for users with slight visual impairments. Most users with visual impairments will need a magnification utility program with higher functionality for daily use.

For a list of Windows-based magnification utilities, see Microsoft Web site

Developing Additional Tours

Since the time I spent working with Bach, I've identified other types of tours that I've found helpful. All of these tours are based on the information in his "Heuristic Test Strategy Model." Each one reminds me to learn about a specific aspect of the product that I might otherwise ignore, and in the process of executing all the tours, I find that I visit the same functionality multiple times and look at it from different perspectives with a different concept of risk. That juxtaposition of different risks and features helps me build a better internal model of the product I'm testing.

The first tour I try to take is a **claims tour**. In this tour, you attempt to find all the information in the product that tells you what the product does. This is helpful in checking for product consistency. A product can be inconsistent with explicit or implicit product claims.

Explicit claims can often be found in help material, marketing material, training material, tool tips or even magazine articles (such as reviews of your product). An implicit claim can be found when your product uses industry terminology, has a common look-and-feel, executes common and well-understood functionality, or has a feature set that is considered to be well-understood.

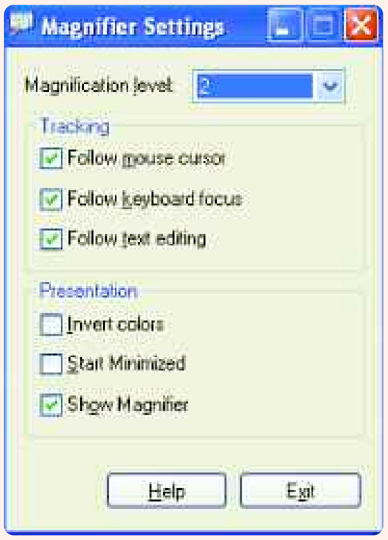
In the claims tour, you are looking not only to identify what the claims are, but also to attempt to identify which claims are vague, possibly incomplete or even inconsistent. While that's actually testing, don't forget that with exploratory testing we are learning, designing and executing our tests all at the same time. Make sure you do not have too much tunnel vision while you're touring.

The next tour I take is the **structure tour**. In this tour, you attempt to learn everything you can about what constitutes the physical product. That can include the code, hardware, multimedia files or databases. I often look at the source and unit tests for languages I'm familiar with. I also try to get a look at the database or the schema(s) if XML is involved. When performance testing, understanding the application architecture and the hardware being used and how it's configured are also priorities.

The next two tours focus on the product's users. The first is the **user tour**. In this tour, you attempt to imagine five users for the product and the information they would want from the product or the major features they would be interested in. The second tour is the **scenario tour**. Here, try to imagine five realistic scenarios for how the users identified in the user tour would use this product.

When I'm attempting to imagine users for the product, I find it helpful to think not only about what users will do, but what they value. I also look to see if

FIG 2 . MAGNIFIER SETTINGS DIALOG


Magnifier Settings

Magnification level: 2

Tracking
[✓] Follow mouse cursor
[✓] Follow keyboard focus
[✓] Follow text editing

Presentation
[ ] Invert colors
[ ] Start Minimized
[✓] Show Magnifier

there is real user data available in the application I'm testing. Test data is good, but real data offers a different insight into how an actual user will possibly use the system. As you're attempting to identify your users, don't forget about users you may not want, such as hackers and other malcontents; users who don't know what they really need to do or how to do it; or users who attempt to do too much with the product at one time.

It's helpful to think about the environments in which those users will operate, the access rights they will have, and the common patterns they will follow. Once you know who the users are, think about compelling stories of how they use the product. What inputs do they have, and what outputs do they care about? The scenarios you want to identify will be compelling stories of how someone who matters might do something that matters with the product.

Depending on the product, I might take a **data tour**. In this tour, you identify the major data elements of the application: what they are, where they are, where they go and where they come from. I recently worked on a Web service where the data tour was the most informative tour I took. Since there was no graphic user interface, little documentation on the service interface and only one defined user, the data became the main source for my test design.

When taking your data tour, don't just identify application inputs and outputs, look for default values, configuration files, options settings and temporal data (that is, data that has relationships with time). Note the size and structure of the data you find, especially if it's representative of production data. Look for possible ways to corrupt data or for ways data can get into the system that developers may not have anticipated. Attempt to identify how the data can be created, accessed, modified and deleted.

Directly related to the data tour is the **configuration tour**. This tour is similar to the variability tour and the data tour, but this time we are looking specifically for persistence and how it affects product features. Attempt to find all the ways you can change settings in the product in such a way that the application retains those settings.

Another important tour is the **interoperability (or compatibility) tour**. On this tour, you attempt to answer the following questions: "What does this application interact with?" "How well does it work with external components and configurations?" and "What does it depend on to function properly?" I recently worked on a project that experienced less than 75 percent uptime in its test environment for the entire test cycle due to all the environmental problems we had and all the external services the application relied on.

During this tour, determine which applications, operating systems, hardware and services the product needs to function. Does it need any special plug-ins, drivers, fonts or language runtimes (Java, Ruby, etc.) to operate correctly? A tip-off that you might be interfacing with an external service may be the appearance of data that you (the user) didn't enter. In addition, degradation in performance can sometimes be a clue that there is something happening in the background.

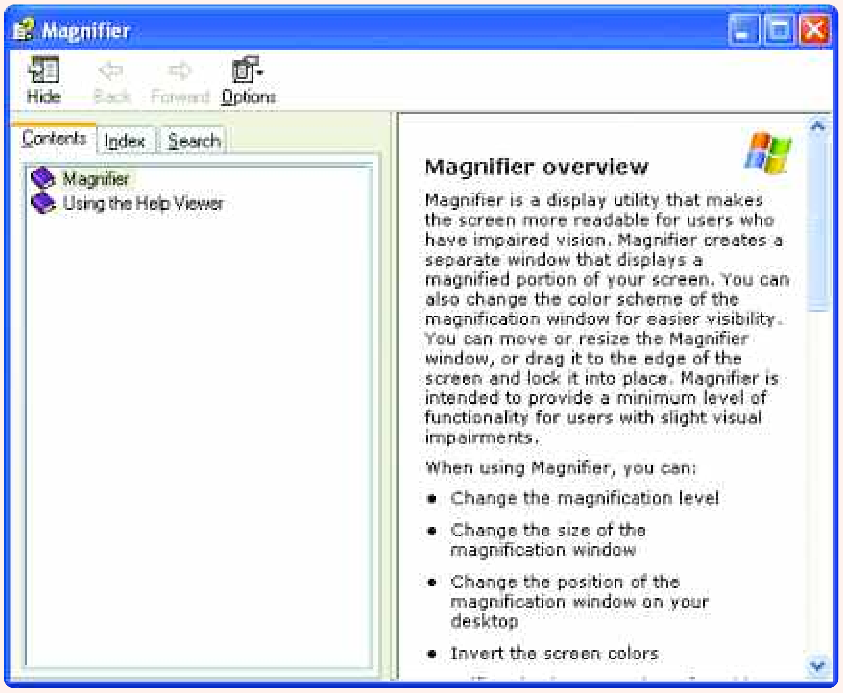
Finally, take a **testability tour**. Try to find all the features you can use as testability features and identify the tools you have available that you can use to help in your testing. Take a look at James Bach's "Heuristics of Software Testability." Thanks to my automation background, this is my favorite tour. A testability feature is a feature of the product that gives us more visibility into the inner works of the product or more control over the state of the product.

The main testability features I look for when taking this tour are scriptable interfaces, test harness and stubs, log files and diagnostic utilities. For example, I often look for WSDL interfaces for Web services, JUnit test harnesses for Java applications and log files for Web application servers. An excellent example of leveraging a scriptable interface can be found in Brian Marick's article "Bypassing the GUI." In addition, attempt to identify modules or layers of the application that can be tested independently. Many systems I've tested have had some level of diagnostic utilities associated with them (especially those that used external Web services). The easiest way to find those is to ask developers, but you can sometimes find them lying around in admin consoles if you are lucky enough to know where to find them.

When looking for test tools, I look for both open-source tools and the commercial tools that I have available. Depending on the software I'm testing, I'll look for any of the following:

Scripting languages
Disk imaging tools
File scanners
Network analyzers
Macro tools
Debuggers
Data generators
Runtime analysis tools
Static and dynamic analyzers

FIG 3. MAGNIFIER HELP DIALOG



Magnifier overview

Magnifier is a display utility that makes the screen more readable for users who have impaired vision. Magnifier creates a separate window that displays a magnified portion of your screen. You can also change the color scheme of the magnification window for easier visibility. You can move or resize the Magnifier window, or drag it to the edge of the screen and lock it into place. Magnifier is intended to provide a minimum level of functionality for users with slight visual impairments.

When using Magnifier, you can:

- Change the magnification level
- Change the size of the magnification window
- Change the position of the magnification window on your desktop
- Invert the screen colors

COMING UP WITH A HEURISTIC THAT WORKS

Now that I have all these tours, I need a way to remember to use them. For me, the easiest way to do that is with a mnemonic.

James Bach suggests these general guidelines to use when developing a heuristic:

- Attempt to solve a problem.
- Conceive of a need or desire to add structure to that attempt OR notice a pattern.
- Look for a pattern in the problem you are solving.
- Try to understand the pattern as best you can.
  - What's the essence of this pattern?
  - How can I simplify this pattern?
- Label it.
- Try it (experiment with it).
  - Be a skeptic.
  - Vary your label.
  - See if you actually remember your heuristic when you need it.

The following is the heuristic I came up with for application touring: FCC CUTS VIDS. The mnemonic stands for the following:

Feature tour
Complexity tour
Claims tour
Configuration tour
User tour
Testability tour
Scenario tour
Variability tour
Interoperability tour
Data tour
Structure tour

Performance test tools
Functional automation tools
File comparison utilities
Test-case management tools
Version-control tools
Diff utilities
Capture reply tools
Video recorders

Application Tours in Action

Let's look at an example of application touring. For this example, we'll look at the Microsoft Magnifier. I've never even started this application before, and all I know about its functionality is what its name tells me. Typically, when I tour an application for the first time, I have a notepad with me. As I tour the application and develop test ideas, I list them out on the notepad. I will do my best to duplicate that behavior for you here. Everything in italics is a note I would make on my notepad. Remember that these are heuristics, which means they are fallible. Different people may find different things while doing the same tour(s).

To get things started, I'll just launch the application. I won't yet actively think of my heuristic; I'll just click around until I run out of ideas. I also will limit my touring to half an hour. That includes both my interaction with the application and my analysis following that interaction where I review my heuristics.

As soon as I start the application *I'm given a claim about the application's functionality.*

Figure 1. I also see three pieces of functionality *a link to Microsoft's Web site,* a checkbox (which also shows *persistent data—it should save my settings* if I click the box), and the OK and "X" buttons to close the dialog. From this dialog, I also can identify a user for this application—*someone with slight visual impairments.* If I were actually doing exploratory testing, I would go ahead and execute some tests at this point for the simple pieces of functionality like the "X" button and the OK button, but for right now I'll just keep touring.

If I click OK, the dialog closes and I'm left with the Magnifier Settings dialog.

Figure 2. I see *11 pieces of functionality:* two buttons, six checkboxes, one select box and two buttons on the window title bar. After identifying those, I also noticed that the *application accepts hotkeys.* I ask myself if it *saves my settings* when I exit. If I click Help, I get the help screen.

Figure 3. I close this right away thinking that *I'll look through the claims in more detail later* and I'll find out if I need to test the functionality of the *default windows help dialog.*

I play around with the magnification level a bit, and notice that the displayed magnification changes. *This seems to be complex.* I will need to take time to think about tests for this feature.

Figure 4. If I move the mouse into the magnification area, the view goes gray and I get a message telling me that I float the magnification area in a window. *This also seems very complex,* so I note it so I can think of meaningful tests later.

I click each of the checkboxes, one at a time. I note their functionality and any behavior I notice as I do so. Finally, I click Exit.

You are not necessarily looking for problems. You are simply learning the product.

Using My Heuristic

It's at this point that I start to run out of ideas. Now I recall my touring mnemonic (see sidebar "Coming Up With a Heuristic That Works") and work through each type of tour.

Feature tour: I think I've noted all the features, but I might start up the application again and double check to see if I missed anything. I think I understood everything. If I was unsure about what a feature did, I would make a note of it and be sure to talk to a developer about it.

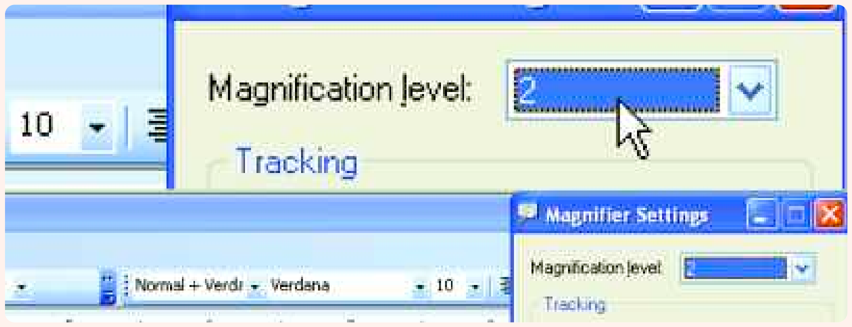
Complexity tour: I noted two features I think are complex. As I think about other types of complexity, it occurs to me that screen resolution might add another dimension to the complexity of my testing. I will need test cases that account for that. I'll also need to think about dual monitors, different video cards (possibly) and different modes of output, like output to a projector.

Claims tour: I noticed the claims made on the dialog and in the help file, but I didn't look at any claims that might be made on the Microsoft Web site, in the Windows XP documentation or elsewhere. I should look into finding more claims made about the application.

Configuration tour: I noticed that the initial dialog might save my settings. I also noticed that some of the checkboxes are application settings that should be persisted. I don't know yet if the magnification level is a persisted configuration setting.

User tour: I identified one type of user, the one in the initial claim. What other users might there be? Well, I might be a user. As I think about it, this might be an excellent testing tool. I could have used this when testing the precision of one of James Lyndsay's machines. I had been using Paint to

FIG 4. EXAMPLE OF MAGNIFIER IN ACTION



magnify the machines in order to count pixels. Now I don't need to do that anymore. So now I have testers and people with slight visual impairments.

I need one more user for my heuristic. I guess I might also use this feature while giving a presentation at a conference. Sometimes it can be hard to see things in the back.

Testability tour: Off the top of my head I would really like a script or a tool that lets me test/emulate many different resolutions and graphics card settings. It also might be handy to have a tool that lets me record my desktop as I test so I can replay crashes as videos.

Scenario tour: I already identified my scenarios when I identified my users. Given (what I believe to be) the narrow scope the application, I needed scenarios to help me think of users. That's OK—just recognize that any of the users identified might have multiple scenarios that may be meaningful.

Variability tour: The most variable feature is the magnification level and the actual magnification display. I can also vary the settings.

Interoperability tour: I know there is a tie into the default windows help dialog, I don't know that it talks to any other applications. I might ask around and look closer at the claims to ensure things are as simple as I think they are.

Data tour: I know it should save some settings when I exit, and I know it has temporal data when I use it (the area of the screen it's magnifying). I don't know where this data is stored (see structure tour). I have not yet thought to test the preset data. I already messed up the settings that we selected the first time I launched the application. I was lucky that I took a screen shot when I first opened it. Otherwise I would have to reinstall.

Structure tour: I have not yet looked at the structure of the application at all. I need to find where all the application files are stored, what DLLs are used (that may help me when thinking about interoperability), and I have not seen the source code. Since I know I can't see the source code, I would at least like to know the language, because that may tip me off to common implementation problems when using that language.

At this point, I would review my notes and start to go around and ask developers questions about what I have so far. I would then organize my ideas into exploratory test charters and begin my first round of testing.

Next Steps

First, identify how you tour the applications you test. Most likely you do it without even thinking about it. Open an application you have never used before and just write down what you notice. When you are done, you can come back and identify patterns in what you look for. Once you know what you can already find, you can develop a heuristic to help you identify the areas of the application you normally overlook.

Once you have a heuristic (I like mnemonics since they allow me to quickly write out more detail in my notes), label it in a way that allows you to remember it. Then try using it. Be skeptical of its effectiveness. Try to identify holes in it. And most important, see if you actually remember your heuristic when you need it—when you test a new application. ☒

REFERENCES

- Bach, James, "Exploratory Testing Explained," www.satisfice.com/articles/et-article.pdf, 2003.
- Bach, James, "Heuristic Test Strategy Model," www.satisfice.com/tools/satisfice-tsm-4p.pdf, 2003.
- Bach, James, "Heuristics of Software Testability," www.satisfice.com/tools/testable.pdf, 2003.
- Bach, Jonathan, "Session-Based Test Management," www.satisfice.com/articles/sbtm.pdf, 2000.
- Marick, Brian, "Bypassing the GUI," STQE Magazine, September/October 2002.

Can Agile Development Really Scale Beyond a Small Team?

Rally has helped thousands of developers, testers, analysts and their managers improve their responsiveness and velocity, gain real-time project visibility, and increase team collaboration for distributed, multi-team projects.

Scale Software Agility from Single Team Projects to Multi-Team Programs

Agile Software Development Management On Demand

- Program and Project Management
- Agile Requirements Management
- Test Management and Defect Tracking

Test drive the *New Rally Team Edition* at www.rallydev.com/stp

Visit our SD West 2006 Booth!

RALLY software development

scaling software agility

rallydev.com/stp