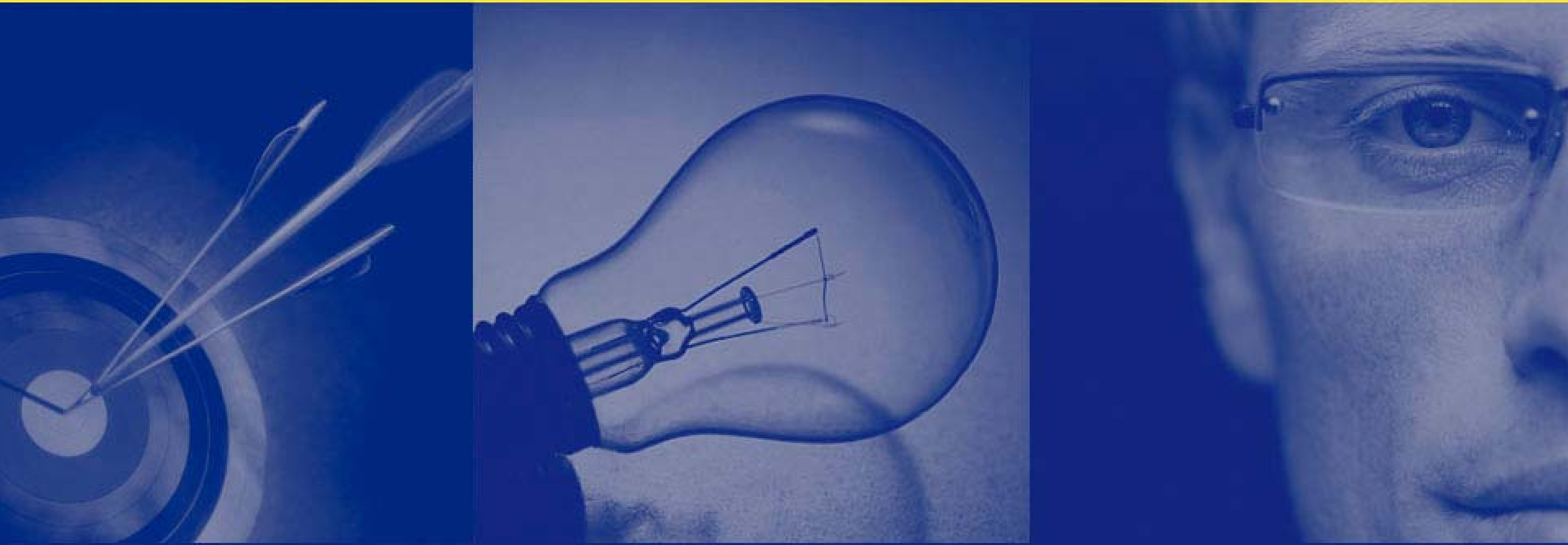
Automated Testing Options for PL/SQL



Steven Feuerstein
PL/SQL Evangelist, Quest Software
www.quest.com   steven.feuerstein@quest.com

How to benefit most from this session

- Concentrate on concepts, not details. Afterwards....
- Download and use any of my the training materials, available at my "cyber home" on Toad World, a portal for Toad Users and PL/SQL developers:

PL/SQL Obsession **http://www.ToadWorld.com/SF**

- Download and use any of my scripts (examples, performance scripts, reusable code) from the demo.zip, available from the same place.

filename_from_demo_zip.sql

- You have my permission to use *all* these materials to do internal trainings and build your own applications.
  - But they should not considered production ready.
  - You must test them and modify them to fit your needs.

What makes an application successful?

- **Seems like we should know and it should be obvious....but is it?**

- **It must be CORRECT.**
  - Meet user requirements, be as free of bugs as possible.
- **It must be FAST.**
  - If it is too slow, user frustration will doom the application.
- **It must be MAINTAINABLE.**
  - Must be easy to understand and maintain. Otherwise, ROI on the application is reduced.

How do we achieve CORRECT, FAST and MAINTAINABLE?

- **CORRECT**
  - Only one way to verify cor ctness test T s he backend code, st the UI, eve
- **FAST**
  - Ide le (s s nd the e
- **MAI**
  - Fo oding standards and review code.
  - Run *regression tests*, after every change, to verify that all programs still work.

Testing is key.

Lots of different kinds of tests

- **Functional or application level testing**
  - Usually performed by QA and users
- **Stress and performance testing**
  - Mostly a DBA job, but also for developers
- **Unit (code) testing**
  - The responsibility of developers.
  - Before we can say a program is finished, we (are supposed to) test it to prove that it works.
- **But how responsible are we?**
- **Let's face it: very few of us adequately test our software.**

Why don't we test more thoroughly?

- **Testing is hard, in any and every language.**
  - For thorough testing, you should expect to have to write *at least* 10 lines of test code for every line of application code that needs testing!
- **Testing is boring.**
  - You are not creating code or writing interesting algorithms.
- **Testing finds bugs (!).**
  - We don't really want to find bugs in our code. We have "gotten by" for years doing what we do.

What's wrong with the way we test?

- **We usually just "try" a few things.**
  - Testing is incomplete; mostly we are reassuring ourselves that the program is not *obviously* broken.
- **We can't repeat our tests.**
  - We all too often do "throw away" testing, with the silent assumption that we will only have to do this once.
- **We manually verify results.**
  - Takes way too much time and I can easily get it wrong.
- **We start testing too late in the process.**
  - If I wait till I am "done" writing my program, I will run out of time.

betwnstr.sf
betwnstr.tst

How can we improve our testing?

- **Manual testing is a dead end.**
  - It will never offer more than "band-aid" testing.
- **There is really only one practical solution: to *automate* code testing as fully as possible.**
- **Automation is key to....**
  - Practical, effective regression testing
  - Giving us the time to test
  - Increasing the coverage of tests
  - Integrating testing *into* the development process.

Options for automated testing of PL/SQL

- **utPLSQL and its variants**
  - Open-source framework, part of the xUnit family
  - You must write the test code yourself.
  - PL/Unit: a light version of utPLSQL
  - PLUTO: an object type-based version of utPLSQL
- **dbFit**
  - Based on the Fit platform, a tabular scripting approach, implemented in Java.
- **Quest Code Tester for Oracle**
  - Robust, integrated test environment
  - Commercial product

About utPLSQL and its variants

- **I built the original utPLSQL back in 1999 or so. I discovered Extreme Programming and its unit testing principle:**
  - "If testing is good, then everyone should test all the time." From there, I learned about Junit.
- **It is a "cooperative paradigm."**
  - You "cooperate" by calling utAssert programs to verify test results. utPLSQL "pays you back" by automatically running your test package and displaying the results.
- **Unfortunately, you still must write the test code yourself.**

ut_betwnstr.pks
ut_betwnstr.pkb

More complete test automation with Quest Code Tester for Oracle

- *Describe* the tests you need through a graphical interface.
- Save your descriptions in a test repository, available for reporting and analysis.
- Generate the test code (a PL/SQL package) based on your descriptions.
- Run the test and view the red light, green light results.

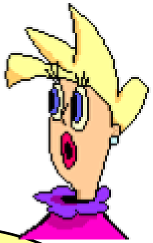
Let's build a test definition for the betwnstr function using Quest Code Tester.

Integrating testing into development

- **As long as we see testing as something we do *after* we are "done" writing our code, we are in serious trouble.**
- **We write our best code if we test *as* we proceed through development.**
  1. Make a change.
  2. Run your test.
  3. Verify that no bugs have been introduced.
  4. Confirm that the new feature works as desired.
- **Yeah, well, how can you do that?**

You *prepare* for each new program

- ***Hold off*** **on writing the cool and challenging algorithms.**
  - First take four steps:



1 **Validate Requirements** → 2 **Construct Header** → 3 **Define Expected Behavior** → 4 **Build Test Code**

Confirm what you are supposed to build, then identify assumptions

Think about the interface, not the implementation How will it be used?

In other words, how will you know when you have finished the program?

Establish a baseline *before* you write the code, then compile it against stub.

Yes! Think about testing before coding.

Crafting successful applications through automated testing

- **Stop separating development from testing.**
  - They are two sides of the same coin.
- **Rely on a predefined, standard testing *framework* that automates as much of the work as possible.**
- **Automated testing with a framework allows you to...**
  - Help you stay focused on critical, required functionality.
  - Greatly reduce the number of bugs.
  - Produce a regression test suite that makes safe evolution and maintenance possible.