The Path to Code Testing Happiness
aka Options for automated testing

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

How to benefit most from this session

- **Watch, listen, _ask questions_. Then 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.**

Testing PL/SQL Programs - Key points

- **It's the _only_ way to know that your programs really and truly work.**
  - And that _is_ the whole point.

- **Testing is mostly viewed as something that happens _after_ development.**
  - In other words, we put it off as long as possible, usually until it is too late.

- **Instead, testing should be seen as a _part_ of the development process.**

- **So why _don't_ we test our code better?**

Challenges of testing PL/SQL programs

- **Testing is hard in any and every language.**
  - You should expect to have to write *at least* 10 lines of test code for every line of application code that needs testing!

- **And in PL/SQL we usually must test the contents of database tables.**
  - That's TOUGH!

- **Other data structures are also hard to test.**
  - Collections, cursor variables, records, XML documents....

Problems with Typical Testing: an example

```
CREATE OR REPLACE FUNCTION betwnstr (
    string_in IN VARCHAR2
  , start_in IN INTEGER
  , end_in IN INTEGER
) RETURN VARCHAR2
IS BEGIN
    RETURN ( SUBSTR (string_in
  , start_in, end_in - start_in + 1 ));
END;
```

- **BETWNSTR is a variation on SUBSTR.**
  - Get string between two start and end points.

- **Can DBMS_OUTPUT.PUT_LINE really be the unit testing mechanism of choice?**

betwnstr.sf
betwnstr_crude.tst
betwnstr.tst

```
BEGIN
    DBMS_OUTPUT.PUT_LINE (betwnstr (NULL, 3, 5, true));
    DBMS_OUTPUT.PUT_LINE (betwnstr ('abcdefgh', 0, 5, true));
    DBMS_OUTPUT.PUT_LINE (betwnstr ('abcdefgh', 3, 5, true));
    DBMS_OUTPUT.PUT_LINE (betwnstr ('abcdefgh', -3, -5, true));
    DBMS_OUTPUT.PUT_LINE (betwnstr ('abcdefgh', NULL, 5, true));
    DBMS_OUTPUT.PUT_LINE (betwnstr ('abcdefgh', 3, NULL, true));
    DBMS_OUTPUT.PUT_LINE (betwnstr ('abcdefgh', 3, 100, true));
END;
```

What's wrong with my testing?

- **I usually just "try" a few things.**
  - Testing is incomplete; mostly we are reassuring ourselves that the program is not *obviously* broken.

- **I can't repeat my tests.**
  - We all too often do "throw away" testing, with the silent assumption that we will only have to do this once.

- **I manually verify results.**
  - Takes way too much time and I can easily get it wrong.

- **I start testing too late in the process.**
  - If I wait till I am "done" writing my program, I will run out of time.

Testing as an *integral part* of development

- **Here's an idea: think about testing your program *before* you implement that program.**
  - When you test, you treat your program as a black box.

- **So just write the header, then stop and ask yourself:**
  - How will I know when I am done, when the program is working?

- **Then take the answers to that question and implement them as test cases!**

**Let's try out these ideas on betwnstr.
Let's start over and "do it right."**

Step 1. What are the requirements?

- **I need a variation of SUBSTR that will return the portion of a string between specified start and end locations.**
- **Some specific requirements:**
  - It should work like SUBSTR as much as makes sense (treat a start location of 0 as 1, for example; if the end location is past the end of the string, the treat it as the end of the string).
  - Negative start and end should return a substring at the *end* of the string.
  - Allow the user to specify whether or not the endpoints should be included.

Step 2. How do I call this program?

```
FUNCTION betwnstr (
    string_in      IN    VARCHAR2
  , start_in       IN    PLS_INTEGER
  , end_in         IN    PLS_INTEGER
  , inclusive_in   IN    BOOLEAN DEFAULT TRUE
)
    RETURN VARCHAR2 DETERMINISTIC
```

- **My specification or header should be compatible with all requirements.**
  - I also self-document that the function is deterministic: no side effects.
- **I can (and will) now create a compile-able stub for the program. Why do that?**
  - Because I can then fully define and *implement* my test code!

betwnstr0*.sf

Steps 3. How will I know when it is working?

- **Let's consider some options that are better than writing hit-or-miss test scripts.**
- **utPLSQL**
  - Open-source framework, part of the xUnit family
  - You must write the test code yourself.
  - PL/Unit: a light version of utPLSQL
- **dbFit**
  - Based on the FitNesse testing framework, you use tabular text to specify your tests
- **Quest Code Tester for Oracle**
  - Robust, integrated test environment.
  - *Generates* test code from repository.

utPLSQL and PL/Unit

- **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

dbFit

- **Sorry, only just learned of it and have not been able to get it installed.**

- **Basic idea: build text tables that define your tests and are then run through the FitNesse engine.**

```
|import|
|dbfit.fixture|

|Query|SELECT ActiveUserCount FROM ActiveUsersCount|
|ActiveUserCount?|
|>>countbeforetransaction|

|Query|SELECT ActiveUserCount FROM ActiveUsersCount|
|ActiveUserCount?|
|>>countaftertransaction|

|Query|SELECT (:countaftertransaction - :countbeforetransaction) As
DiffCount|
|DiffCount?|
|1|
```

Real test automation with Quest Code Tester for Oracle

- *Describe* the tests you need through a graphical interface -

  It's just like SQL vs. programming.

- 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.

Now let's test BETWNSTR using Quest Code Tester....

Change Your Testing (and Development) Ways

- **Stop separating development from testing.**
  - They are two sides of the same coin.
- **Don't give up on testing because of the amount of effort involved.**
- **Find a way to *automate your testing*.**
  - Without automation, it is all but impossible to test well.
- **Every testing framework will help you....**
  - Focus on describing and building features.
  - Greatly reduce the number of bugs.
  - Produce regression tests that makes confident code evolution and maintenance possible.