



Six Simple Steps to Unit Testing Happiness



Steven Feuerstein
steven.feuerstein@questcom
www.unit-test.com
www.quest.com

Writing software is.....

FUN!

Testing software is.....

Hard Work

Buggy software is....

- Embarrassing
- Expensive
- Deadly

Buggy software is embarrassing

- There can be as many as 20 to 30 bugs per 1,000 lines of software code. —Sustainable Computing Consortium
- 32% of organizations say that they release software with too many defects.—Cutter Consortium
- 38% of organizations believe they lack an adequate software quality assurance program.—Cutter Consortium
- 27% of organizations do not conduct any formal quality reviews.—Cutter Consortium
- Developers spend about 80% of development costs on identifying and correcting defects.—The National Institute of Standards and Technology

Buggy software is expensive - \$60B per year in US alone!?


- **JUNE 25, 2002 (COMPUTERWORLD) - WASHINGTON -- Software bugs are costing the U.S. economy an estimated \$59.5 billion each year.** Of the total \$59.5 billion cost, users incurred 64% of the cost and developers 36%.
- **There are very few markets where "buyers are willing to accept products that they know are going to malfunction,"** said Gregory Tasse, the National Institute of Standards and Technology senior economist who headed the study. "But software is at the extreme end in terms of errors or bugs that are in the typical product when it is sold."
- **Oh, yes and Y2K: \$300B? \$600B?**

Buggy software is deadly

- **2003** Software failure contributes to power outage across the Northeastern U.S. and Canada, killing 3 people.
- **2001** Five Panamanian cancer patients die following overdoses of radiation, amounts of which were determined by faulty use of software.
- **2000** Crash of a Marine Corps Osprey tilt-rotor aircraft partially blamed on “software anomaly” kills four soldiers.
- **1997** Radar that could have prevented Korean jet crash (killing 225) hobbled by software problem.
- **1995** American Airlines jet, descending into Cali, Colombia, crashes into a mountain, killing 159. Jury holds maker of flight-management system 17% responsible. A report by the University of Bielefeld in Germany found that the software presented insufficient and conflicting information to the pilots, who got lost.

How do we avoid buggy software?

- Clear and accurate requirements
- Careful design
- Excellent tools
- Best practices, standards, guidelines (that is, follow them)
- Code review
- Thorough testing



**Uh oh...
the world is in
big trouble.**

Wouldn't it be great if...

- It was easy to construct tests
 - An agreed-upon and effective approach to test construction that everyone can understand and follow
- It was easy to run tests
 - And see the results, instantly and automatically.
- Testing were completely integrated into my development, QA, and maintenance processes
 - No program goes to QA until it has passed a battery of tests
 - Anyone can maintain with confidence, because my test suite automatically validates my changes

Different types of testing

- There are many types of testing: unit tests, functional/system tests, stress tests...
- A "unit test" is the test of a single unit of code.
- Unit tests are the responsibility of developers - that is, us, the people in this room.
 - **Not fundamentally a job for the QA department, which generally focuses on functional and system tests.**

- How do you (or your team) *unit test* your PL/SQL code today?
 - ? – We use automated testing software.
 - ? – We have a formal test process that we each follow, but otherwise a manual process.
 - ? – Everyone does their own thing and we hope for the best.
 - ? – Our users test our code.

Unit testing reality

- Let's face it: we PL/SQL developers don't spend nearly enough time unit testing our code.
 - For the most part, we run a script that displays output on the screen and then we stare at all until we decide if the test succeeded or failed.
- There are some understandable reasons:
 - Very few tools and utilities have been available, *to date*, for PL/SQL testing.
 - Managers don't give us enough time to prepare and execute tests.

- DBMS_OUTPUT.PUT_LINE - unit testing mechanism of choice?

```
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;
```

betwnstr.sf
betwnstr.tst

Problems with Typical Testing

- Almost entirely ad hoc
 - No comprehensive effort to compile test cases
 - No infrastructure to record cases and administer tests
- Difficult to verify correctness
 - Non-automated verification is slow and error-prone.
- Relies on the user community to test
 - Since we are never really sure we've tested properly, we rely on our users (or, we are lucky, the QA department) to finish our job

There has got to be a better way!

Moving towards a Better Way

- Change from within: your code will not test itself.
 - You must accept the responsibility and then be disciplined (sigh...that's not fun at all).
 - Commit to testing and watch the way you write your code change.
- Change from without: new possibilities are on the horizon!
 - utPLSQL and Ounit <http://utplssql.sourceforge.net/>
 - Quest Code Tester for Oracle <http://www.unit-test.com>
- Ah, but what about those six, simple steps?

Six Simple Steps to Unit Testing Happiness

- 1. Describe fully the **required functionality** of the program.
- 2. Elaborate the **test cases** for the program.
- 3. Define the **header of the program** (name, parameter list, return value).
- 4. Build **test code** that implements all test cases.
- 5. Write the **program unit**.
- 6. **Test, debug, fix**, test, debug, fix, test, debug....
- *Then...repeat* steps 3-6 for each **enhancement and bug report**.

Describe required functionality

- 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.

- *Before I write any code, I will come up with as many of the test cases as possible -- and write my test code.*
 - This is known as "test-driven development". TDD is a very hot topic among developers and is associated with Agile Software (<http://agilemanifesto.org/>) and Extreme Programming.
- Putting aside the fancy names and methodologies, TDD makes perfect sense -- when you stop to think about it.

If you write your program before you define your tests, how do you know you when you're done?

And if you write your tests *afterward*, you are likely to prejudice your tests to show "success."

Brainstorm the test cases

- Even a simple program will have many test cases!
 - You don't have to think of *every* one before you implement your program and start your testing.
 - You should aim at least for a "representative" sampling.
- But where do you store/define the test cases?
 - You can certainly put the information in and work from a document or spreadsheet.
 - Best of all, however, is to link the test case definitions as tightly as possible to the code.

Some of the test cases for BETWNSTR

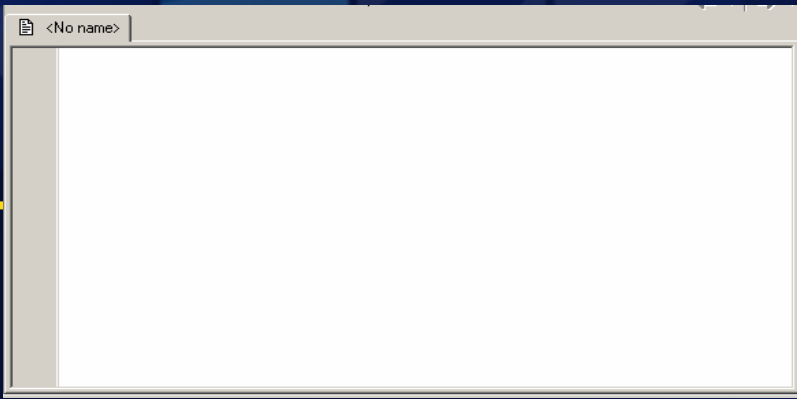
- Start and end within the string ("normal" usage)
- Start of 0
- End past end of string
- Null string, string of single character, 32767 len character
- Null start and/or end
- Negative start and end
- Start larger than end (positive and negative)
- Variations of the above with different inclusive values

Define the program specification

```
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!

Testcases and Test Code



- The challenge (terror?) of the blank screen....
 - How do I define the test cases?
 - How do I set up those tests?
 - How do I verify the results?
- Let's see how the Quest Code Tester for Oracle helps me tackle these challenges.
 - Define and maintain your test cases through a graphical interface, then let it do all the work.

Write the program.

- Now that I know I can test the program, I can start implementing betwnstr...

Finally!

betwnstr1.sf

First version of "between string"

```
CREATE OR REPLACE 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  
IS  
BEGIN  
    RETURN ( SUBSTR (  
        string_in  
        , start_in  
        , end_in - start_in + 1 )  
    );  
END;
```

Test, debug, fix, test, debug, fix, test, debug...

- With a test script in place, I can quickly and easily move back and forth between running my program, identifying errors, debugging and fixing the code, running the program again.
- I also then have my test process and regression test in place so that as I make enhancements or fix bugs, I can fall back on this foundation.
 - It is *critical* that you maintain your test case definitions and test code as your program evolves.
 - And update those *first* -- before you change the program!

Change Your Testing Ways

- Qute (and even utPLSQL) can make a dramatic difference in your ability to test and your confidence in the resulting code.
- Build a comprehensive "library" of unit tests as you build your application
 - These tests and all their test cases can be passed on to other developers
 - Anyone can now enhance or maintain the code with confidence. Make your changes and run the tests. If you get a green light, you're OK!

Testing: Baby steps better than paralysis.

- Unit testing is an intimidating process.
 - You are never really done.
 - You have to maintain your test code along with your application code.
- But every incremental improvement in testing yields immediate and long-term benefits.
 - Don't worry about 100% test coverage.
 - Download Quest Code Tester for Oracle and give it a try!
 - Special pre-release version free at least through January 2007!