Regression Testing-I

- Regression testing is the execution of a set of test cases on a program in order to ensure that its revision does not produce unintended faults, does not regress, i.e. become less effective than it has been in the past
- Need to balance
  - Cost of selecting and executing test cases
  - The fault detection ability of the regression test suite

Regression Testing-II

Given program P, its modified version P', and a test set T that was used to previously test P, find a way to utilize T to gain sufficient confidence in the correctness of P'

Is regression testing important?

- Many examples show its importance
  - Ariane 5
    - Previously tested code was assumed to work correctly when re-used
  - Mars Polar Lander
    - Landing process not fully re-tested after the first landing failed and a wiring problem was fixed
    - Crashed on Mars surface as a result (most likely scenario)
Winamp example
- Winamp 2.76 contained a bug related to HTML playlists
  - It would allow malicious users to insert dangerous code in an ID3 tag
- Version 2.77 corrected this problem
- However, it re-appeared in version 2.79
  - Corrected again in 2.80

Regression testing
- Three different visions:
  - Procedural: Run the same tests again
  - Risk-oriented: Expose errors caused by change
  - Refactoring support: Help the programmer discover implications of her code changes.
- Fundamental question or goal
  - Good regression testing gives clients confidence that they can change the product (or product environment).

Procedural regression testing
- Tag line
  - "Run the same tests again"
- Paradigmatic cases
  - Manual, scripted regression testing
  - Automated GUI regression testing
  - Smoke testing (manual or automated)

Benefits
- The tests exist already (no need for new design, or new implementation, but there will be maintenance cost)
- Many regulators and process inspectors like them
- Because we are investing in re-use, we can afford to take the time to craft each test carefully, making it more likely to be powerful in future use
- Relatively quick and easy
- Many commercial tools available
An analogy: Clearing mines

Totally repeatable tests won’t clear the minefield

Variable Tests are Often More Effective

Procedural regression testing

- Blind spots / weaknesses
  - Anything not covered in the regression series
  - Repeating the same tests means not looking for the bugs that can be found by other tests
  - Low yield from automated regression tests
  - Maintenance of this standard list can be costly and distracting from the search for defects
Risk-oriented regression testing

- Fundamental question or goal
  - Manage the risks that
    - (a) a bug fix didn’t fix the bug or
    - (b) the fix (or other change) had a side effect.
- Paradigmatic case(s)
  - Bug regression (Show that a bug was not fixed)
  - Old fix regression (Show that an old bug fix was broken)
  - General functional regression (Show that a change caused a working area to break)

Examples of increasing power

- Do more iterations (one test hits the same function many times).
- Do more combinations (interactions among variables, including the function under test’s variables).
- Do more things (sequences of functions that include the function under test).
- Look for specific errors (such as similar products’ problems) that involve the function under test.

Risk-oriented regression testing

- In this approach, we might re-use old tests or create new ones
- Often, we retest an area or function with tests of increasing power (perhaps by combining them with other functions)
- The focus of the technique is on testing for side effects of change, not the inventory of old tests.

Examples of increasing power

- Methodically cover the code (all N-length sequences that include the function under test; all N-wide combinations that include the function under test’s variables and variables that are expected to interact with or constrain or be constrained by the function under test).
- Try other types of tests, such as scenarios, that involve the function under test.
- Try to break it (take a perverse view, get creative).
Refactoring support

- Fundamental question or goal
  - Support refactoring: Help the programmer discover implications of her code changes.
- Paradigmatic case(s)
  - Test-driven development using glass-box testing tools like JUnit.
  - The programmer creates these tests and runs them every time she compiles the code.

If a test suite takes more than 2 minutes, the programmer might split tests into 2 groups (tests run at every compile and tests run every few hours or overnight).

The intent of the tests is to exercise every function in interesting ways, so that when the programmer refactors code, she can quickly see

- what would break if she made a change to a given variable, data item or function or
- what she did break by making the change.

Glass-box regression testing

- In the unit test situation, the programmer (not an independent tester) writes the tests, typically before she writes the code. The testing focuses the programming, yielding better code in the first place.
  - In the unit test case, when the programmer makes a change that has a side-effect, she immediately discovers the break and fixes it. There is no communication cost.

Black-box regression testing

- Test case maintenance costs are high
  - The same broken area of code might be involved in dozens of system level tests
  - It takes a while for the black box tester to understand the implications of the code changes (which he doesn’t see)
  - The programmer fixes tests that are directly tied to the changes she makes, and she sees the tests break as soon as she makes the change, which helps her reappraise whether the change she is making is reasonable.