Black Box Software Testing

Spring 2005

REGRESSION TESTING

by
Cem Kaner, J.D., Ph.D.
Professor of Software Engineering
Florida Institute of Technology
and
James Bach
Principal, Satisfice Inc.

These notes are partially based on research that was supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Good regression testing gives clients confidence that they can change the product (or product environment).

Why use regression tests?

- Manage risks of change:
  (a) a bug fix didn't fix the bug or
  (b) the fix (or other change) had a side effect or
  (c) error in the build process
  (d) faulty localization
- **Conform to expectations of clients or regulators**
- **Conform to a misguided notion of "scientific" practice**
  - Replicability is an important attribute of experiments because it lets us:
    - cross-check each other's work
    - obtain a result as needed for application, and
    - re-examine an experiment in the face of a new theory
  - ***Replicability* is important.**
    - ***Repetition* may or may not have much value.**
- **Conform to a faulty analogy to manufacturing**
  - **Software quality is not about making many identical instances of the same widget, within the tightest achievable tolerances**

Managing risks of change

(a) a bug fix didn't fix the bug

- **Bug regression** (Show a bug was not fixed)

(b) the fix (or other change) had a side effect

- **Old fix regression** (Show an old bug fix was broken) *Programmers refactor their code to reduce this risk.*
- **General functional regression** (Show that a change caused a working area to break.)
- **Unit-level regression** (Discover changes *while fixing* the bugs)

(c) error in the build process

- **Build verification testing** (Smoke tests have little power but much breadth. The question is whether the build is fit to test). *Programmers use source and version control to reduce this risk.*

(d) faulty localization

- **Localization testing** (Determine whether a product has been properly adapted to the language and culture of a specific locale or market.)

Repeat testing after changes

- Which tests repeat?
- How similar to the previous ones?
- Why?

The essence of regression testing is repetition of tests after changes. But if you scratch the surface, you find several fundamentally different visions.

- ***Procedural***: Run the same tests again in the same ways
  - **Delegation of labor**: Welcome to the factory
  - **Reuse good tests**: Two visions of power (of tests)
  - **Acts of faith**: We do it because (… it's scientific…?)
- ***Economical***: Repeat tests because (when, if) they're cheap
- ***Support refactoring***: Help the programmer discover implications of her code changes.
- ***Risk-oriented***: Expose errors caused by change

Automating GUI regression

Automated GUI regression is classic procedural regression.

And the most commonly discussed approach to test automation :

1. create a test case
2. run it and inspect the output
3. if the program fails, report a bug and try again later
4. if the program passes the test, save the resulting outputs
5. in future tests, run the program and compare the output to the saved results. Report an exception whenever the current output and the saved output don't match.

Procedural regression: Reusing good tests

- It makes sense to reuse old tests that were carefully crafted.
- Does that mean that regression tests are repetitions of good tests?
  - Any test, *designed from any other perspective*, can be turned into a regression test.

    Just document and / or automate it

    And reuse it
- But what if you create good tests?
  - Should they become regression tests?

Procedural regression: Reusing good tests

- The key design issue for regression tests *as regression tests* is the expectation of reusability.
- Long term reusability requires maintainability
  - *Does that imply simplicity?*
    - If so, does that imply that we should prefer smoke tests to scenario tests? (Some people think so…)
    - But complex tests that use a lot of features of the program and enter a lot of data are the ones that might be the most beneficial to automate because this saves so much data entry time and spares use the nuisance of data entry mistakes accidentally made by the tester who is trying to perform the test by hand.
  - *I think the need for maintainability implies a need for architectural support for strong tests, not a push for weak ones.*

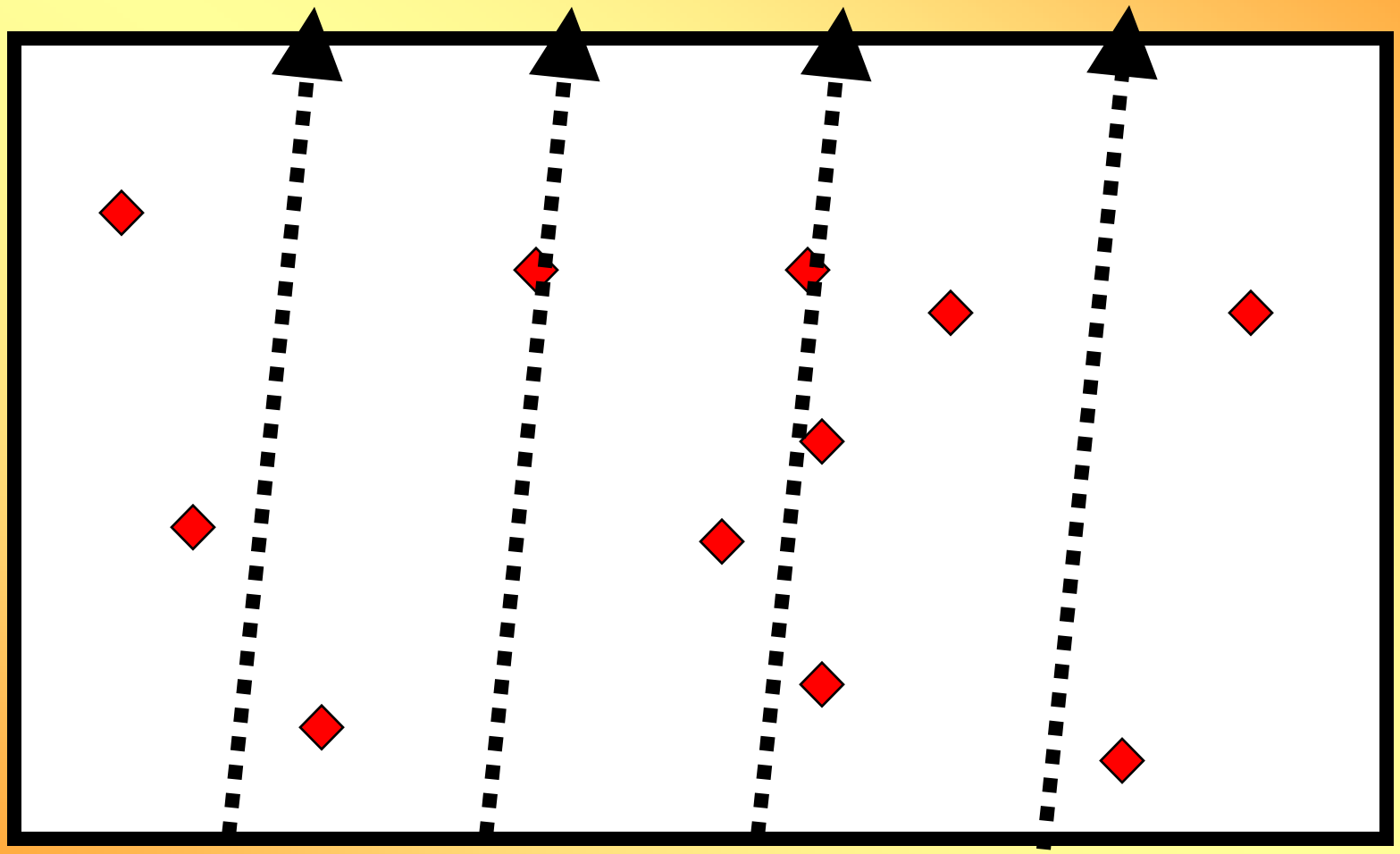
Reusing good tests

- If power refers to the ability of a test to detect a bug *if it is there*, then a test can be powerful even if the program has passed it so often that we are almost certain that bug will never appear.
- **On the other hand,**
  - The probability of finding a bug with a regression test that the program has already passed *is* very low
  - Estimates from LAWST:

    About 15% of the bugs found in a project are found by its regression tests. *(Obviously, there's a lot of variation.)*

    However, companies spend as much as 80% of their testing budget on regression testing (development and, especially, maintenance of the tests) and some of these have very low bug find percentages.
- A theoretically less powerful test can be more likely to detect bugs *in the same area as the theoretically more powerful test,* if you've already run the more powerful test (but not the weaker one) and the program consistently passes it

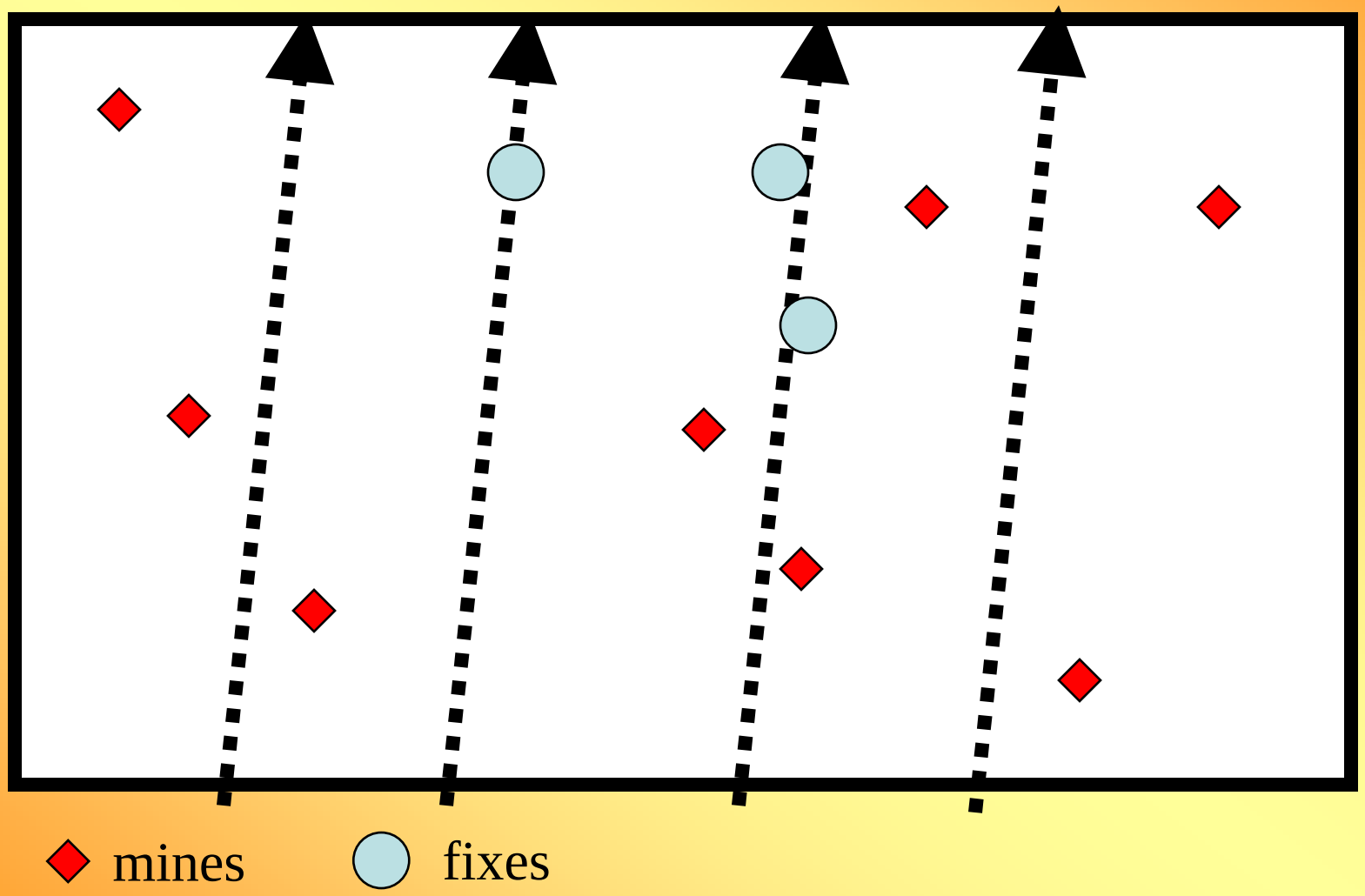
An analogy: Clearing mines



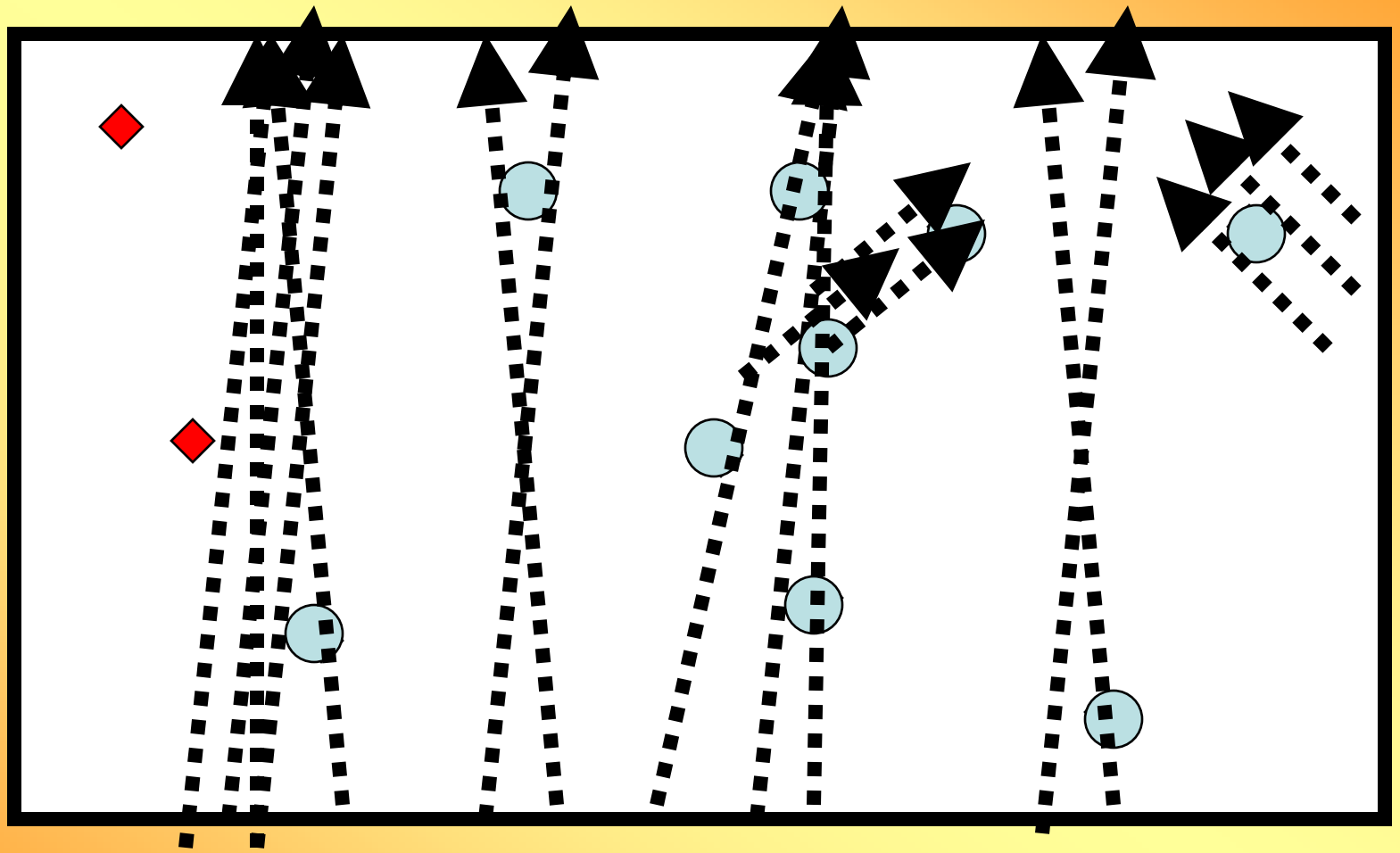
◆ mines

This analogy was first presented by Brian Marick. These slides are from James Bach..

Totally repeatable tests won't clear the minefield



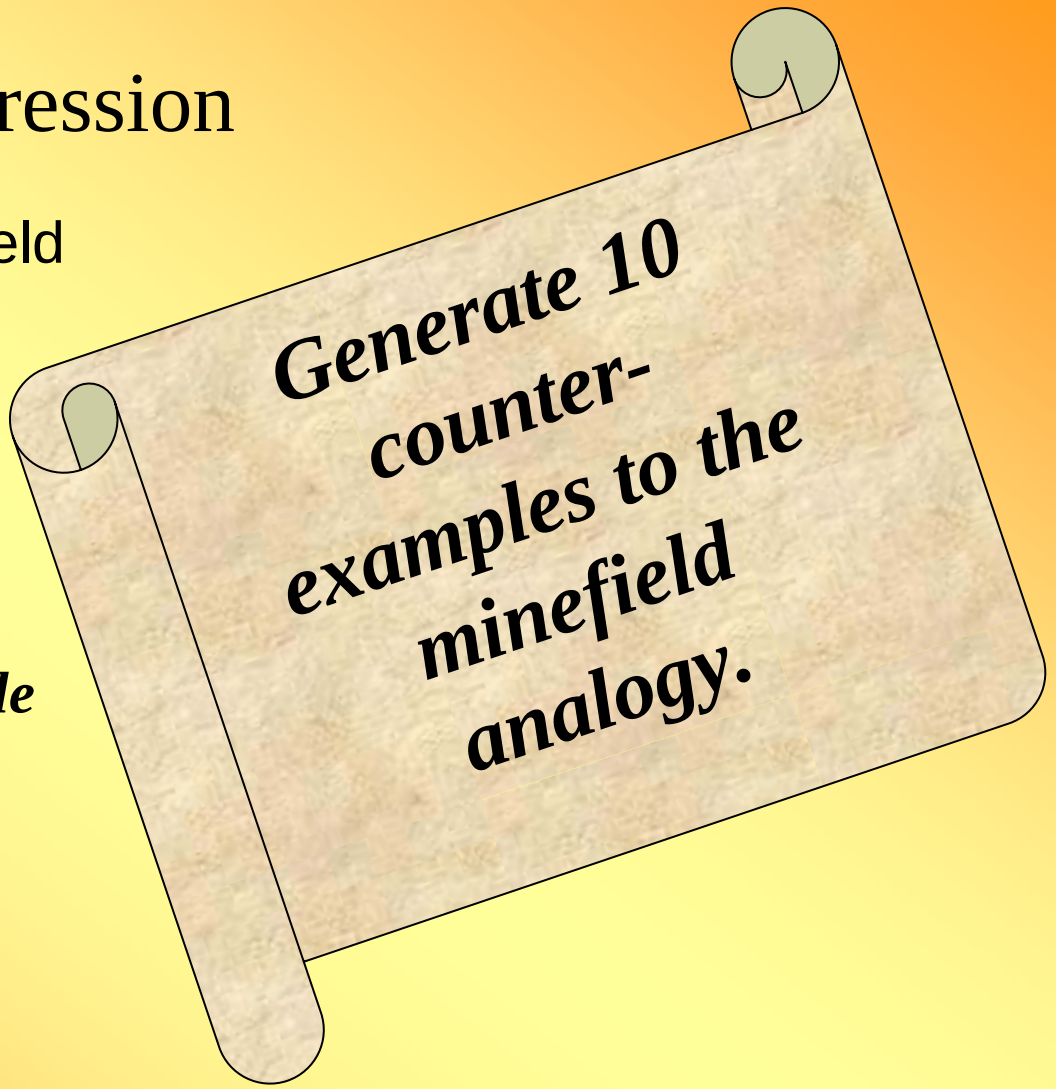
Variable Tests are Often More Effective



mines fixes

Automated GUI regression

- Look back at the minefield analogy
- Are you convinced that variable tests will find more bugs under *all* circumstances?
  - ***If so, why would people do repeated tests?***



Repeat testing after changes

- Which tests repeat?
- How similar to the previous ones?
- Why?

The essence of regression testing is repetition of tests after changes. But if you scratch the surface, you find several fundamentally different visions.

- ***Procedural***: Run the same tests again in the same ways
  - **Delegation of labor**: Welcome to the factory
  - **Reuse good tests**: Two visions of power (of tests)
  - **Acts of faith**: We do it because (… it's scientific…?)
- ***Economical***: Repeat tests because (when, if) they're cheap
- ***Support refactoring***: Help the programmer discover implications of her code changes.
- ***Risk-oriented***: Expose errors caused by change

The economic perspective

- Cost of reuse versus cost of generation of new tests
  - What is the maintenance cost compared to creation cost?
  - Will we have the resources available for new tests in the event of
    - Patch releases?
    - Localization releases?
- Benefit of reused test versus benefit of new tests
  - High power, saved data entry time, elimination of user error, performance benchmarking, etc.
- Opportunity cost of working on variants of old tests instead of new tests that
  - *Focus on new areas of the program* or
  - *Focus on vital areas of the program*.
    - We might reuse tests not because we think they're great
      - but because it might be better to use weak tests than none, when the cost of creating better tests exceeds the benefit.

Repeat testing after changes

- Which tests repeat?
- How similar to the previous ones?
- Why?

The essence of regression testing is repetition of tests after changes. But if you scratch the surface, you find several fundamentally different visions.

- ***Procedural***: Run the same tests again in the same ways
  - **Delegation of labor**: Welcome to the factory
  - **Reuse good tests**: Two visions of power (of tests)
  - **Acts of faith**: We do it because (… it's scientific…?)
- ***Economical***: Repeat tests because (when, if) they're cheap
- ***Support refactoring***: Help the programmer discover implications of her code changes.
- ***Risk-oriented***: Expose errors caused by change

Refactoring support: Change detectors

- The programmer modifies working code in order to improve its quality (such as maintainability, performance, etc.)
- The program should pass the same behavioral tests before and after the change(s).
- Unit test libraries make refactoring much easier
  - Common case: test-driven development using glass-box testing tools like JUnit, httpUnit, and FIT to create the low-level regression tests.
  - 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.

Refactoring support: Change detectors

There are enormous practical differences between system-level black-box regression testing and unit-level (or integration-level) 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 tests focuses the programming, yielding better code in the first place.
- In the unit test case, when the programmer's change has a side-effect, (under XP) she immediately discovers and fixes it. There is no communication cost. You don't have (as in black box testing) a tester who discovers a bug, replicates it, reports it, and then a project manager who reads the report, maybe a triage team who study the bug and agree it should be fixed, a programmer who has to read the report, troubleshoot the problem, fix it, file the fix, a tester who has to retest the bug to determine that the fix really fixed the problem and then close the bug. All labor-hours considered, this can easily cost 4 hours of processing time, compared to a few minutes to fix a bug discovered at the unit level.
- In the black box case, test case maintenance costs are high, partially because the same broken area of code might be be involved in dozens of system level tests. And partially because 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.
- Fowler, *Refactoring;* Astels, *Test-Driven Development;* Link, *Unit Testing in Java;* Rainsberger *JUNIT Recipes*

Repeat testing after changes

- Which tests repeat?
- How similar to the previous ones?
- Why?

The essence of regression testing is repetition of tests after changes. But if you scratch the surface, you find several fundamentally different visions.

- ***Procedural***: Run the same tests again in the same ways
  - **Delegation of labor**: Welcome to the factory
  - **Reuse good tests**: Two visions of power (of tests)
  - **Acts of faith**: We do it because (… it's scientific…?)
- ***Economical***: Repeat tests because (when, if) they're cheap
- ***Support refactoring***: Help the programmer discover implications of her code changes.
- ***Risk-oriented***: Expose errors caused by change

Risk-oriented regression

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.

Here are examples of a few common ways to test a program more harshly while retesting in the same area.

- **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).
- **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).
- **Look for specific errors** (such as similar products' problems) that involve 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).
  - » Thanks to Doug Hoffman for a draft of this list

Summing up

- Regression testing as a technique:
  - **It primarily specifies an activity**
    - Retest after changes
  - **It only weakly specifies a risk**
    - Changes break things
    - Regulators might reject things that don't look tightly controlled
  - **It specifies a weak oracle**
    - Whatever happened last time
      - That won't always be correct
      - And it doesn't help when you first design and run the test
  - **It doesn't address coverage or the characteristics of the tester.**
- *The details of test design come from some other underlying technique.*

Summing up

- Regression tests have mixed costs and benefits
  - Many regulators and process inspectors like and expect this approach
  - The tests exist already (no need for new design, or new implementation, but the maintenance cost can be enormous)
  - Because we are investing in re-use, sometimes we can afford to craft each test carefully, making it more likely to be valuable later.
  - This is the dominant paradigm for automated testing, so it's relatively easy to justify and there are lots of commercial tools.
  - These tests are often overrated. They lose their power to find bugs over time (the bug find curve for regression has been compared to an immunization curve—Boris Beizer nicknamed it the *Pesticide Paradox)*
  - Repeating ***the same tests*** means ***not looking*** for the bugs that can be found by ***other tests***.
- You might get the same benefits, with more creativity, in a less structured approach to regression. But that might cost more.
- Much regression testing can / should be done, more efficiently and effectively, as unit tests ***rather than*** black box system tests.

Readings

REGRESSION TESTING IN GENERAL

- Marick, How Many Bugs Do Regression Tests Find?
- Beck, Test-Driven Development By Example

GUI-LEVEL AUTOMATED REGRESSION TESTING

- Chris Agruss, Automating Software Installation Testing
- James Bach, Test Automation Snake Oil
- Hans Buwalda, Testing Using Action Words
- Hans Buwalda, Automated testing with Action Words: Abandoning Record & Playback
- Elisabeth Hendrickson, The Difference between Test Automation Failure and Success
- Doug Hoffman, Test Automation course notes
- Cem Kaner, "Improving the maintainability of automated test suites." *Software QA*, Volume 4, #4, 1997. (Also published in Proceedings of the *10th International Software Quality Conference (Quality Week)*, San Francisco, CA, May 28, 1997.)
- Cem Kaner, "Avoiding shelfware: A manager's view of automated GUI testing." (Keynote address) *Software Testing Analysis & Review Conference (STAR) East*, Orlando, FL, May 6, 1998.
- Cem Kaner, "Architectures of test automation." *Software Testing, Analysis & Review Conference (Star) West*, San Jose, CA, October, 2000.
- John Kent, Advanced Automated Testing Architectures
- Bret Pettichord, Success with Test Automation
- Bret Pettichord, Seven Steps to Test Automation Success
- Keith Zambelich, Totally Data-Driven Automated Testing