Black Box Software Testing

2004 Academic Edition

PART 11 -- 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.

Regression Testing: 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

Regression testing

- Tag line: *"Repeat testing after changes."*
  - But scratch the surface and you find three fundamentally 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).

Risk-oriented regression testing

- Tag line
  - *"Test after changes."*
- 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.)

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.

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

Refactoring support: Change detectors

- Tag line
  - *"Change detectors"*
- 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, httpunit, and fit.***

NOTES:

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

Refactoring support: Change detectors

This is out of scope for Testing 1, but a substantial part of the Testing 2 course.

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 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. You don't have (as you would 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.

- Beck, *Test-Driven Development By Example*
- Astels, *Test-Driven Development*
- Link, *Unit Testing in Java*
- Fowler, *Refactoring*

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)

Procedural regression testing

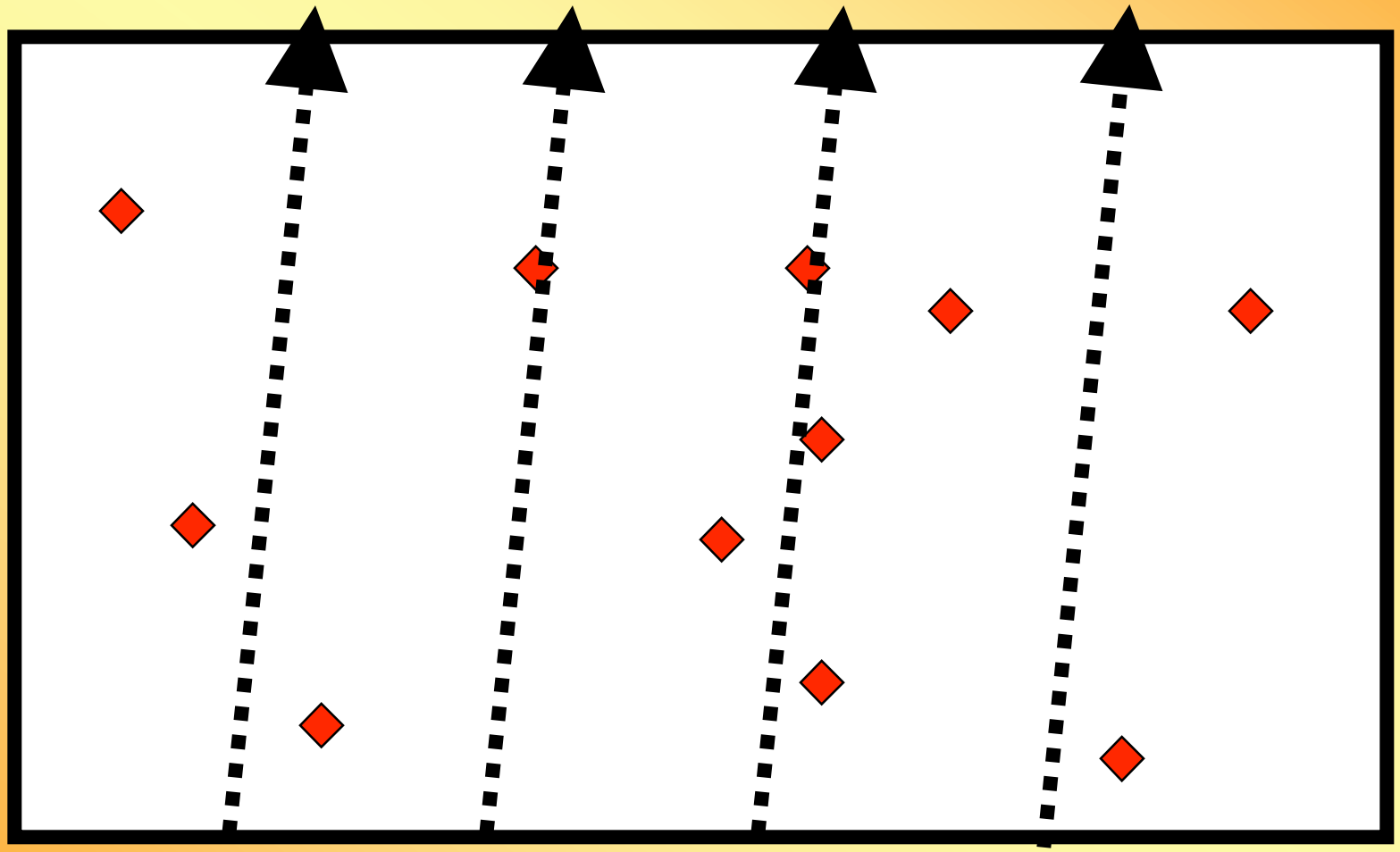
- Benefits
  - The tests exist already (no need for new design, or new implementation, but there will be maintenance cost)
  - Many regulators and process inspectors like them
  - Because we are investing in re-use, we can afford to take the time to craft each test carefully, making it more likely to be powerful in future use
  - This is the dominant paradigm for automated testing, so it is relatively easy to justify and there are lots of commercial tools
  - Implementation of automated tests is often relatively quick and easy (though maintenance can be a nightmare)

  - *We'll look more closely at implementation details in the discussion of automation. For now, let's look at the conceptual strengths and weaknesses of this approach.*

Automating GUI regression testing

- This is the most commonly discussed automation approach:
  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.

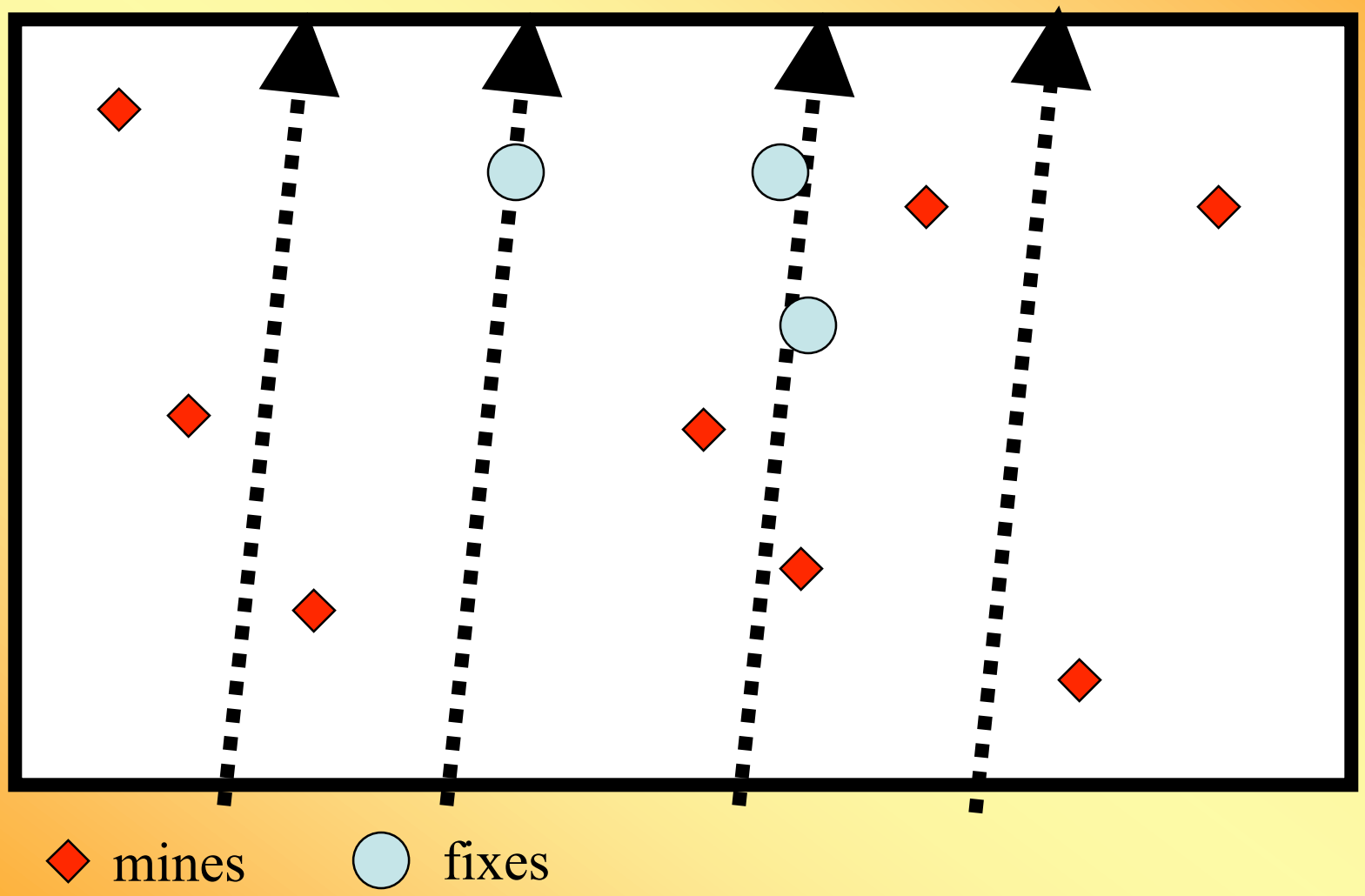
An analogy: Clearing mines



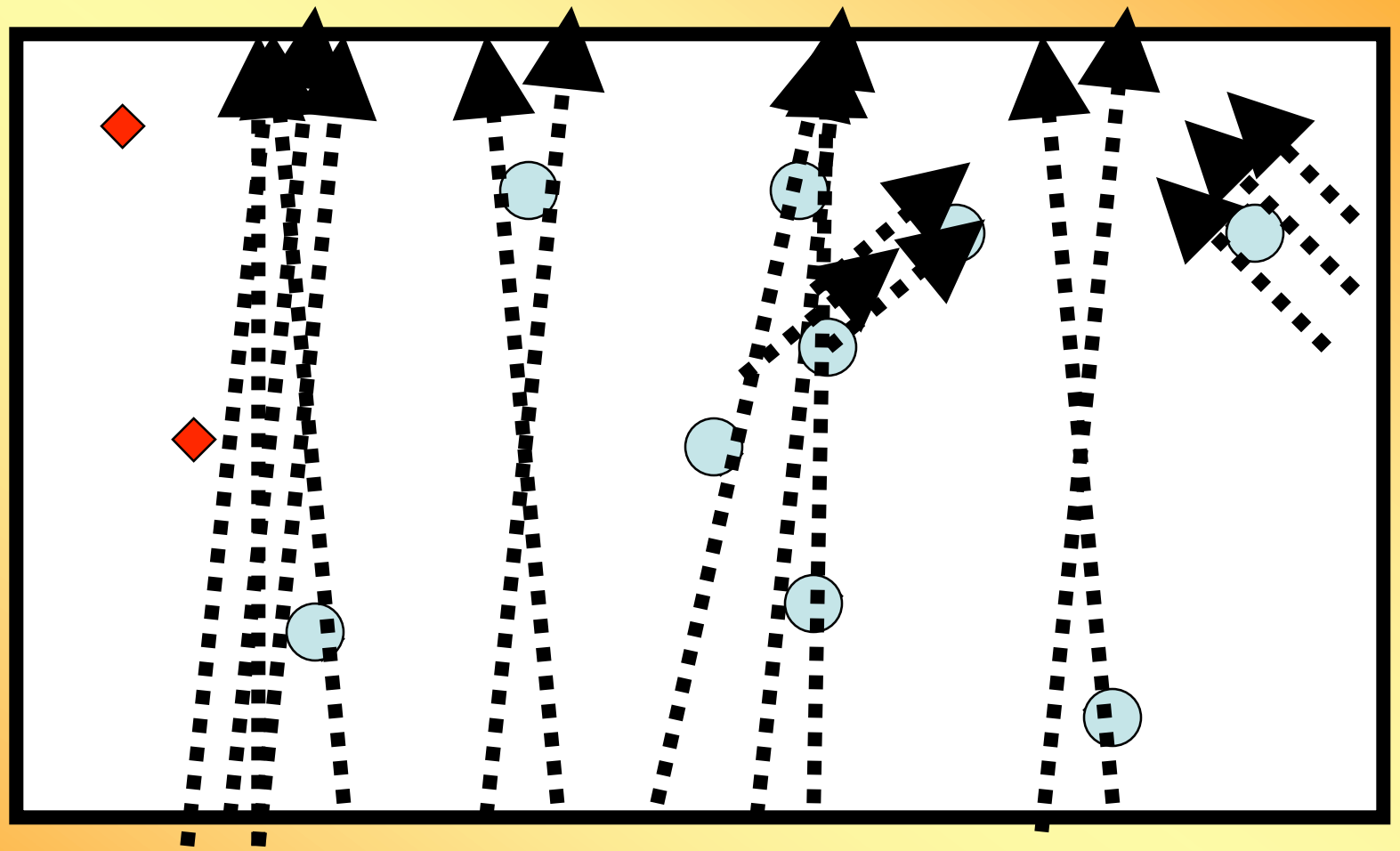
mines

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 testing

- 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?***

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.
  - Pesticide paradox
  - Low yield from automated regression tests
  - Maintenance of this standard list can be costly and distracting from the search for defects.