Standard Regression Testing Does Not Work

Daniel Hansson, CEO Verifyter
Standard Regression Testing

- Check out the latest from the version control system
- Kick off a script that runs a lot of tests and present the test results
- Email the engineer(s) who have committed new changes to the revision control system since the last time the test(s) passed
Blame – 3 Standard Options

1. Blame all commits since last pass
2. Blame the first commit that caused the test to fail
3. Don’t blame anyone. Manually debug the test failure
The Standard Approach

“If we just test often enough then we will know who caused each regression failure”

Hey, it’s this guy!

Looks simple, but is it true?
We Checked

Is that the reason the test fails on the latest?

If so, undoing the change should make the test pass
The Result

We checked 916 bug reports in ASIC projects

In 41% of the cases the bug report was wrong!

Why is this simple approach so unreliable?
Accurate/Inaccurate Blame

Accurate Blame

“This commit was the reason this test started to fail”

Inaccurate Blame

“The test must still fail for the same reason here”

A lot of things happen after the first failure
Inaccurate Blame

41%

Many Scenarios
Two Solutions

1. Prevent Complexity
   – Test one commit at a time
   – Strict Continuous Integration

2. Handle Complexity
   – You must debug correctly all complex scenarios
   – Automatic debugger of regression failures
Strict Continuous Integration

Test each commit. Only let it in if the tests pass

Change → Test 

Gate Keeper Test

RCS

Test passed
Change is integrated into the Revision Control System (RCS)

Test failed
Bug report sent back to engineer

Perfect if complete test suite takes minutes and does not contain random tests. Very popular in the software industry.
Automatic Debugger

PinDown TestHub

Version Control System

Test Executor (LSF Farm)

PinDown Results Database

Email Bug Reports
Rerun on Old Revisions

Rerunning tests on older revisions to find bad commit

- Only takes 1 – 4 reruns of the failing test to find culprit
- Combines revisions in order to analyze complex scenarios
- This is only done for one test per bug
Farm Usage

- 200 tests (200h run time) - 10 failures
- Debug (10 tests, parallel)
- 200 tests - all pass

- Bug 1 after 2h
- Bug 2 after 5h
- Bug 1 fixed
- Bug 2 fixed

• Typical cost for re-running tests: 10 jobs on the farm
• Good investment to avoid next test run to show same state
Benefits

400% Faster Fixes, 5x Less Discussion, to 11% Shorter Projects
**Summary**

### Bad Commit Known

<table>
<thead>
<tr>
<th>Standard Regression Testing</th>
<th>Strict Continuous Integration</th>
<th>PinDown</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Benefits

<table>
<thead>
<tr>
<th></th>
<th>Standard Regression Testing</th>
<th>Strict Continuous Integration</th>
<th>PinDown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Triage</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bugs Fixed Faster</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Less Debate</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Shorter Project</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Support

<table>
<thead>
<tr>
<th></th>
<th>Standard Regression Testing</th>
<th>Strict Continuous Integration</th>
<th>PinDown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Test Suites</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Random Tests</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Cost

<table>
<thead>
<tr>
<th></th>
<th>Standard Regression Testing</th>
<th>Strict Continuous Integration</th>
<th>PinDown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug Testing</td>
<td>Yes (manual)</td>
<td>No</td>
<td>Yes (automatic)</td>
</tr>
</tbody>
</table>

**Only works for very small test suites. Random not supported**

**Can debug complete test suite, incl random tests**

**With continuous integration all large test suites still needs to be debugged either manually or by PinDown**