Techniques For Improving Regression Testing In Continuous Integration Development Environments

Sebastian Elbaum*, Gregg Rothermel*, John Penix*

*University of Nebraska-Lincoln, *Google Inc.

Slides prepared by Farah Hariri
Outline

- Overview
- Testing At Google
- Problem and Existing Techniques
- Proposed Approach
- Evaluation
Outline

▪ Overview
▪ Testing At Google
▪ Problem and Existing Techniques
▪ Proposed Approach
▪ Evaluation
Main Question: How to conduct regression testing *cost effectively* in a *continuous integration system* at a large scale like Google?
Overview

- **Main Question**: How to conduct regression testing *cost effectively* in a *continuous integration system* at a large scale like Google?

- **Proposed technique**: revisit the old regression test selection and prioritization techniques from the literature to be *lightweight* and *effective*

---

**Proposed Technique**

\[ T' \subseteq T \]

- \( T \): set of regression tests to run
- History of runs for those tests (passing or failing)

\[ T' \] a prioritized permutation of \( T \) executing tests expected to fail first
Main Question: How to conduct regression testing *cost effectively* in a continuous integration system at a large scale like Google?

Proposed technique: revisit the old regression test selection and prioritization techniques from the literature to be *lightweight* and *effective*.

Evaluation: apply the new techniques over a dataset collected from Google simulating their testing process.
Outline

- Overview
- Testing At Google
- Problem and Existing Techniques
- Evaluation
Regression Testing At Google
Regression Testing At Google

Developers

new code

Testing Team
Regression Testing At Google

Developers

new code

test suites

Testing Team
Regression Testing At Google

Developers

new code

test suites

automated testing infrastructure

Testing Team
Testing At Google

- Does the new functionality work?
- Did we break existing functionality?
Testing At Google

- Does the new functionality work?
- Did we break existing functionality?

multiple submits

submit
Testing At Google

- Does the new functionality work?
- Did we break existing functionality?

Run regression tests

multiple submits

submit
Testing At Google

- Does the new functionality work?
- Did we break existing functionality?

multiple submits

Run regression tests

Test Failure

submit
Testing At Google

- Does the new functionality work?
- Did we break existing functionality?

Run regression tests

Test Failure

Detective work to track down the bad changes
Testing At Google

Pre-submit testing stage

Run regression tests

Post-submit testing stage
Testing At Google

Pre-submit testing stage

Run regression tests

Post-submit testing stage

submit
Testing At Google

Pre-submit testing stage

Run regression tests

Post-submit testing stage

submit

Run regression tests
Testing At Google

Pre-submit testing stage

Run regression tests

Post-submit testing stage

submit
Testing At Google

Pre-submit testing stage

Run regression tests

Post-submit testing stage

- Prevents excessive problems in post-submit
- Early detection of failures
- Reduces number of problems that slip into the codebase and affect future builds
Outline

- Overview
- Testing At Google
- Problem and Existing Techniques
- Proposed Approach
- Evaluation
Problem

How to make this process of pre-submit and post-submit testing efficient and cost effective in a CI environment like Google?

- The simplest regression testing strategy: retest all
  ■ prohibitively expensive
- Existing Regression techniques in the literature:
  ■ Regression Test Selection
  ■ Regression Test Prioritization
Existing Techniques: Regression Test Selection

- For a given test suite $T$ and a code change, select a subset $T'$ of $T$ that exercises the code changes to run.
Existing Techniques: Regression Test Selection

- For a given test suite $T$ and a code change, select a subset $T'$ of $T$ that exercises the code changes to run.

- The diagram illustrates tests $t_1$, $t_2$, and $t_3$ for version $v$, and how the tests are impacted by the code change for version $v+1$. Only $t_3$ is highlighted, indicating it is selected for testing in the new version.
Existing Techniques: Regression Test Selection

- For a given test suite T and a code change, select a subset T’ of T that exercises the code changes to run.
Existing Techniques: Regression Test Selection

- For a given test suite $T$ and a code change, select a subset $T'$ of $T$ that exercises the code changes to run.
Existing Techniques: Regression Test Prioritization

- For a given test suite $T$ and a code change, reorder the tests such that faults can be detected early in the test execution cycle.
Existing Techniques: Regression Test Prioritization

- For a given test suite T and a code change, reorder the tests such that faults can be detected early in the test execution cycle.
Existing Techniques: Regression Test Prioritization

- For a given test suite $T$ and a code change, reorder the tests such that faults can be detected early in the test execution cycle.
Existing Techniques: Why are they not suitable?

- most of them require code instrumentation
  - data gathered is rendered obsolete by code churns in CI systems

- require significant analysis time
  - overly expensive due to the high frequency of arrival of testing requests in CI systems
  Ex: Google’s codebase undergoes 10 changes per minute!
Outline

- Overview
- Testing At Google
- Problem and Existing Techniques
- Proposed Approach
- Evaluation
Proposed Approach

- Continuous Regression Test Selection (pre-submit phase)
- Continuous Regression Test Prioritization (post-submit phase)
Proposed Approach

- Continuous Regression Test Selection (pre-submit phase)
- Continuous Regression Test Prioritization (post-submit phase)
Continuous Regression Test Selection (pre-submit)

- **Key Idea:** in evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change
  - selecting test suites based on some “failure window $W_f$” might be cost effective
Continuous Regression Test Selection (pre-submit)

❖ **Key Idea:** in evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change

➢ selecting test suites based on some “failure window $W_f$” might be cost effective
Continuous Regression Test Selection (pre-submit)

- **Key Idea:** In evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change.
  - Selecting test suites based on some “failure window $W_f$” might be cost effective.
Continuous Regression Test Selection (pre-submit)

- **Key Idea:** in evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change
  - selecting test suites based on some “failure window $W_f$” might be cost effective
- However, this approach ignores the effect of test suites that did not reveal faults in recent builds
  - use an “execution window $W_e$”, where tests not executed within that window are selected
Continuous Regression Test Selection (pre-submit)

- **Key Idea:** in evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change
  - selecting test suites based on some “failure window $W_f$” might be cost effective
- However, this approach ignores the effect of test suites that did not reveal faults in recent builds
  - use an “execution window $W_e$”, where tests not executed within that window are selected
Continuous Regression Test Selection (pre-submit)

- **Key Idea:** in evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change
  - selecting test suites based on some “failure window $W_f$” might be cost effective
- However, this approach ignores the effect of test suites that did not reveal faults in recent builds
  - use an “execution window $W_e$”, where tests not executed within that window are selected
Continuous Regression Test Selection (pre-submit)

- **Key Idea:** in evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change
  - selecting test suites based on some “failure window $W_f$” might be cost effective

- However, this approach ignores the effect of test suites that did not reveal faults in recent builds
  - use an “execution window $W_e$”, where tests not executed within that window are selected

- This approach ignores newly added test suites, which are clear candidates for execution in the pre-submit stage
  - automatically add new test suites to execute
Continuous Regression Test Selection (pre-submit)

- **Key Idea:** in evolving systems, test suites that have failed in a recent version are in some ways “proxies” for code change
  - selecting test suites based on some “failure window $W_f$” might be cost effective
- However, this approach ignores the effect of test suites that did not reveal faults in recent builds
  - use an “execution window $W_e$”, where tests not executed within that window are selected
- This approach ignores newly added test suites, which are clear candidates for execution in the pre-submit stage
  - automatically add new test suites to execute
Algorithm 1 SelectPRETests

Parameters:
Test Suites $T$,
Failure window $W_f$,
Execution window $W_e$

for all $T_i \in T$ do
    if $\text{TimeSinceLastFailure}(T_i) \leq W_f$ or $\text{TimeSinceLastExecution}(T_i) > W_e$ or
        $T_i$ is new then
        $T' \leftarrow T' \cup T_i$
    end if
end for
return $T'$
Approach

- Continuous Regression Test Selection (pre-submit phase)

- Continuous Regression Test Prioritization (post-submit phase)
Continuous Regression Test Prioritization (post-submit)

- **SelectPRETests** can have a secondary effect of shifting the execution of failing test suites to the post-submit whenever those tests are not selected.
- Skipping test suites that would have failed in the pre-submit causes:
  - delays in failure finding
  - slow down development
- Solution: test suite prioritization in post-submit
  - reveals failures faster
  - developer may use this info to decide whether to continue the system build or halt it
Continuous Regression Test Prioritization (post-submit)

- **SelectPRETests** can have a secondary effect of shifting the execution of failing test suites to the post-submit whenever those tests are not selected.
- Skipping test suites that would have failed in the pre-submit causes:
  - delays in failure finding
  - slow down development
- Solution: test suite prioritization in post-submit
  - reveals failures faster
  - developer may use this info to decide whether to continue the system build or halt it
- Prioritization algorithm similar to the selection algorithm in the pre-submit

![Diagram showing execution history with weights $w_e$ and $w_f$]
Continuous Regression Test Prioritization (post-submit)

- However there are issues to consider here:
  a. continuous arrival of test suites in small batches or bursts related to code submits
Continuous Regression Test Prioritization (post-submit)

- However there are issues to consider here:
  a. continuous arrival of test suites in small batches or bursts related to code submits
Continuous Regression Test Prioritization (post-submit)

- However there are issues to consider here:
  a. continuous arrival of test suites in small batches or bursts related to code submits => *Lower priority test suites remain unexecuted for longer than desired*
Continuous Regression Test Prioritization (post-submit)

- However there are issues to prioritizing suites as they arrive (similar to traditional test suite prioritization):
  a. continuous arrival of test suites in small batches or bursts related to code submits => *Lower priority test suites remain unexecuted for longer than desired*
  b. May cause test suites that have relatively similar behavior to all be assigned high priority => suites with dissimilar behavior get executed later, lowering prioritization effectiveness
  c. focuses on test suites related to single submissions => most of them won’t fail because of the pre-submit validation => scheduling many likely-to-pass test suites ahead of more-likely-to-fail test suites
Continuous Regression Test Prioritization (post-submit)

- To address all of these issues: Introduce a prioritization window $W_p$ over the suites submitted for execution.
- When $W_p$ is exceeded, we prioritize the non prioritized test suites.

![Diagram showing test prioritization]

- Suits submitted for execution
- Suites execution history

Legend:
- Red: high priority
- Pink: low priority
To address all of these issues: Introduce a prioritization window $W_p$ over the suites submitted for execution.

When $W_p$ is exceeded, we prioritize the non prioritized test suites.

Continuous Regression Test Prioritization (post-submit)

- suites submitted for execution
- suites execution history
- window size exceeded!

$W_f$ is the window size exceeded!
Continuous Regression Test Prioritization (post-submit)

- To address all of these issues: Introduce a prioritization window $W_p$ over the suites submitted for execution.
- When $W_p$ is exceeded, we prioritize the non prioritized test suites.

NOTE: the history of runs used for prioritization is ONLY from the post-submit failures.
Continuous Regression Test Prioritization (post-submit)

Algorithm 2 PrioritizePOSTests

Parameters:
- POSTQueue,
- Failure window \( W_f \),
- Execution window \( W_e \),
- Starting point \( P_0 \) in POSTQueue

for all \( T_i \) ∈ POSTQueue after \( P_0 \) to lastEntry.POSTQueue do
  if \( \text{TimeSinceLastFailure}(T_i) \leq W_f \) or \( \text{TimeSinceLastExecution}(T_i) > W_e \) or \( T_i \) is new then
    \( T_i\).Priority ← 1
  else
    \( T_i\).Priority ← 2
  end if
end for

sortByPriority(POSTQueue, \( P_0 \), lastEntry.POSTQueue)

\( P_0 = \) lastEntry.POSTQueue
Outline

- Overview
- Testing At Google
- Problem and Existing Techniques
- Proposed Approach
- Evaluation
Evaluation

- Evaluate to answer the following two questions:
  - How cost-effective is the RTS technique during pre-submit testing and how does the cost vary with different settings of $W_f$ and $W_e$?
    - Compare against baseline of retest all and random selection
  - How cost-effective is the TCP technique during post-submit testing and how does the cost vary with different settings of $W_p$?
    - Compare against no prioritization
    - Evaluates based on time to detect first failure instead of APFD
Evaluation

Contains over 3.5M records of test suites executions

<table>
<thead>
<tr>
<th>Phase</th>
<th>Size</th>
<th>% Total</th>
<th>% Failing</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-submit</td>
<td>Small</td>
<td>26</td>
<td>0.02</td>
</tr>
<tr>
<td>pre-submit</td>
<td>Medium</td>
<td>10</td>
<td>0.13</td>
</tr>
<tr>
<td>pre-submit</td>
<td>Large</td>
<td>6</td>
<td>0.27</td>
</tr>
<tr>
<td>post-submit</td>
<td>Small</td>
<td>35</td>
<td>0.01</td>
</tr>
<tr>
<td>post-submit</td>
<td>Medium</td>
<td>13</td>
<td>0.32</td>
</tr>
<tr>
<td>post-submit</td>
<td>Large</td>
<td>10</td>
<td>0.65</td>
</tr>
</tbody>
</table>
Evaluation

Figure 3: Test Suite Selection: $W_e = 1$

Figure 4: Test Suite Selection: $W_e = 24$
Evaluation

Figure 6: Sample of prioritization results for 25 failures

Figure 7: Boxplots comparing prioritization techniques with $W_p = \{0.1, 0.5, 1, 2, 4, 8, 12\}$ against no-prioritization