Theo: Test Effectiveness Optimization from History

Kim Herzig£, Michaela Greiler$, Jacek Czerwonka$, Brendan Murphy£

£Microsoft Research, Cambridge
$Microsoft Corporation, Tools for Software Engineers (TSE)

Data extracted by the CODEMINE process managed by the TSE group (Redmond)
Improving Development Processes

Microsoft aims for shorter release cycles
- Speed up development processes (e.g. code velocity)
- Maintaining / increasing product quality

Empirical data to support & drive decisions

Joint effort by MSR & product teams
- **MSR Cambridge**: Brendan Murphy, Kim Herzig
- **MSR Redmond**: Tom Zimmermann, Chris Bird, Nachi Nagappan
- **TSE Redmond**: Jacek Czerwonka, Michaela Greiler
- **Windows, Windows Phone, Office, Dynamics product teams**

Technology changes
Legacy changes
New product features
Development Environment (should be well balanced)

Product / Service
Cost
Speed

Quality / Risk
Why Software Testing?

Software testing is very expensive
- Thousands of gates executed, millions of test cases executed
- Different branches, architectures, languages, etc.

Current process aims for maximal protection
- Aims to find code issues as early as possible
- Slows down product development

Actual problem
- We tend to repeat the same tests over and over again
- Too many false alarms (failures due to test and infrastructure issues)
- Each test failure further slows down product development
Project Goal

Reduce the number of test executions...

- **Identify tests** not failing at all or that are **more likely to produce false alarms**

...without sacrificing code quality

- **Run** every test at least once before integrating code change into **trunk branch** (winmain).
- We **eventually find all code issues** but we **take the risk of finding them later**.

Dynamic, self-adaptive optimization model
Solution

Dynamically reducing execution frequency of quality gate tests

• Analyze historic events on branch
  • Builds
  • Quality gate executions

• Analyze past code changes
  • From where do these changes come from?

• Analyze past test results
  • Passing tests, false alarms, detected code issues
Solution

Cost function decides when to skip tests.

\[ \text{Cost}_{\text{Execution}} > \text{Cost}_{\text{Skip}} \quad ? \quad \text{suspend} : \text{execute test} \]

\[
\text{Cost}_{\text{Execution}} = \text{Cost}_{\text{Machine/Time}} \times \text{Time}_{\text{Execution}} + \text{"Cost of potential false alarm"} \\
= \text{Cost}_{\text{Machine/Time}} \times \text{Time}_{\text{Execution}} + (\text{Prob}_{\text{FP}} \times \text{Cost}_{\text{Developer/Time}} \times \text{Time}_{\text{Triage}} )
\]

\[
\text{Cost}_{\text{Skip}} = \text{"Potential cost of elapsing a bug to next higher branch level"} \\
= \text{Prob}_{\text{TP}} \times \text{Cost}_{\text{Developer/Time}} \times \text{Time}_{\text{Freeze branch}} \times \#\text{Developers}_{\text{Branch}}
\]
Current Results

Simulated on Windows 8.1 development period (BVT only)
Dynamic, Self-Adaptive

Decision points are connected to each other

- **Skipping tests influences the risk factors** of higher level branches
- We **re-enable tests** if code quality drops (e.g. different milestone)

![Graph showing relative test reduction rate over time](image)

- **Training period**
- **automatically enable tests again**
Project Goal

Reduce the number of test executions ...

- Identify tests not failing at all or that are more likely to produce false failures

... without sacrificing code quality

- Run every test at least once before integrating changes into trunk branch (or main)
- We eventually find all code issues but take less time finding them later (on higher level branches)

Dynamic, self-adaptive optimization model

Solution

Skipping tests is risky. Using cost function to model risk

Comparing expected costs of running and not running test

\[ Cost_{\text{Execution}} > Cost_{\text{Skip Test}} \rightarrow \text{suspend} : \text{execute test} \]

\[ Cost_{\text{Execution}} = Cost_{\text{Machine/Time}} \times Time_{\text{Execution}} + \text{"Cost of potential false failure"} \]

\[ = Cost_{\text{Machine/Time}} \times Time_{\text{Execution}} + (Prob_{TP} \times Cost_{\text{Developer/Time}} \times Time_{\text{Stage}}) \]

\[ Cost_{\text{Skip Test}} = \text{"Potential cost of elapsing a bug to next higher branch level"} \]

\[ = Prob_{TP} \times Cost_{\text{Developer/Time}} \times Time_{\text{Freeze Branch}} \times \#\text{Developers/branch} \]

Dynamic, Self-Adaptive

Decision points are connected to each other

- Skipping tests influences the risk factors of higher level branches
- We re-enable tests if code quality drops (e.g., different milestone)

Current Results

Simulated on Windows 8.1 development period (IT only)