DynaQFocus: Focusing test prioritization on builds with test failures

Emad Fallahzadeh and Peter C Rigby
Concordia University
CSER 2021

emad.fallahzadeh@concordia.ca

https://users.ensc.concordia.ca/~pcr/
peter.rigby@concordia.ca
Introduction

- Big companies like Google make lots of changes per minute
- They run thousands of tests to verify code changes
- They follow Continuous Integration (CI) process
  - Requires rerunning tests for each change
- It delays release in a rapid release environment
- Test prioritization can help
Related works

- Some studies focused on pre-submit test-case selection
- Others conducted test case prioritization after submitting the change
- Kim and Porter were pioneers in using historical test failures for test prioritization
- Elbaum et al. used a combination of pre-submit selection and post-submit prioritization
Our contribution

● Design a new test prioritization algorithm
  ○ Bases on the hypothesis that bugs might cluster
  ○ One failing test might be a clue for the other ones
● Analyze testing datasets features in big projects
Prioritization Algorithms

- BatchedFIFO (baseline)
- GoogleTCP
- DynaQFocus
- DynaQFocusFail
BatchedFifo Algorithm

Algorithm 1: BatchedFifo

Result: BatchedFifo get builds one after another and runs one test from each.

1 while There are more builds do
2 fill dispatchQueue with b builds;
3 while dispatchQueue is not empty do
4 | build = dispatchQueue.getNextBuild();
5 | run(build.getNextTest());
6 | end
7 end
BatchedFifo Simulation

1. C
   B
   A

2. D
   C
   B

3. D
   B
   A
BatchedFifo Simulation

![BatchedFifo Simulation Diagram]
BatchedFifo Simulation

C
B
A

D
C
B

D
B
A

1 2 3
BatchedFifo Simulation

1 2 3

C D D
B C B
A B A
BatchedFifo Simulation

![Diagram of BatchedFifo Simulation](image-url)
GoogleTCP Algorithm

Algorithm 2: GoogleTCP

Result: GoogleTCP prioritizes the test cases for the next run after running the test cases inside the dispatch queue each time

1 while There are more builds do
2    fill dispatchQueue with b builds and order the test cases in each build based on their previous failures;
3    while dispatchQueue is not empty do
4        build = dispatchQueue.getNextBuild();
5        run(build.getNextTest());
6    end
7 end
GoogleTCP Simulation
GoogleTCP Simulation

1 2 3

E
D
C

C
B
A

D
C
A
GoogleTCP Simulation

1

E
D
C

2

C
B
A

3

D
C
A

15
GoogleTCP Simulation

E  C  D
D  B  C
C  A  A

1  2  3
GoogleTCP Simulation

Diagram: A rectangle is divided into three columns labeled 1, 2, and 3. Each column contains a smaller rectangle with the letters A, B, C, D, E. The letters are ordered as follows:

- Column 1: E, D, C
- Column 2: B, C, A
- Column 3: D, C, A
GoogleTCP Simulation

E
D
C

1

B
C
A

2

D
C
A

3
GoogleTCP Simulation
GoogleTCP Simulation

1. E
2. D
3. C

1. B
2. C
3. A

1. D
2. C
3. A
Algorithm 3: DynaQFocus

Result: DynaQFocus focuses on code changes that have a failing test, prioritizing the tests of this build above other builds.

```plaintext
/* we run tests from b builds to avoid starvation */

while There are more builds do
  fill dispatchQueue with b builds;
  while dispatchQueue is not empty do
    build = dispatchQueue.getNextBuild();
    verdict = run(build.getNextTest());
    /* on pass, go to the next build */
    if verdict == failure then
      /* focus: run all tests for the current build */
      runAllTests(build);
    end
  end
end
```
DynaQFocus Simulation

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
DynaQFocus Simulation

D  C  D
C  B  C
B  A  A
1  2  3
DynaQFocus Simulation

D  C  D
C  B  C
B  A  A
1  2  3
DynaQFocus Simulation

```
  D  
  C  
  B  
1

  C  
  B  
  A  
2

  D  
  C  
  A  
3
```
DynaQFocus Simulation

![Diagram of DynaQFocus Simulation with nodes and connections]

Nodes:
- D
- C
- B
- A

Connections:
- D to C
- C to B
- B to A
- A to D

Steps:
1. D
2. C
3. D
4. C
5. A
6. A

Progression:
- Start at D
- Move to C
- Continue to B
- Then to A
- Finally, return to D
DynaQFocus Simulation

D
C
B
1

C
B
A
2

D
C
A
3
Algorithm 4: DynaQFocusFail

Result: DynaQFocusFail prioritizes the tests in builds based on their previous failures and also focuses on a buggy build.

1 while There are more builds do
2 fill dispatchQueue with b builds with tests prioritized in each build based on their previous failures. The more a test fails previously, the higher it will be in order;
3 while dispatchQueue is not empty do
4 build = dispatchQueue.getNextBuild();
5 verdict = run(build.getNextBuild());
6 /* on pass, go to the next build */
7 if verdict == failure then
8 /* focus: run all tests for the current build */
9     runAllTests(build);
10 end
11 end
12 end
DynaQFocusFail Simulation

\[
\begin{array}{ccc}
D & C & C \\
C & B & D \\
B & A & A \\
\end{array}
\]
DynaQFocusFail Simulation
DynaQFocusFail Simulation
DynaQFocusFail Simulation

D  C  C
C  B  D
B  A  A
1  2  3
DynaQFocusFail Simulation

1

D

C

B

2

C

B

A

3

C

D

A
DynaQFocusFail Simulation

D
C
B

C
B
A

C
D
A

1
2
3
DynaQFocusFail Simulation

D D C

C C B

B B A

A A

1 2 3
DynaQFocusFail Simulation
DynaQFocusFail Simulation
DynaQFocusFail Simulation

1. D
   C
   B

2. C
   B
   A

3. D
   C
   A

arrow from 2 to 3
Datasets overviews

- **Google**
  - 3.5 million tests
  - 8,952 failing tests
    - Test failure ratio = 0.25%
  - Large builds
    - Consisting of up to 65,000 tests

- **Chrome**
  - 5.2 million tests
  - 810,514 failing tests
    - Test failure ratio = 15.4%
  - Small builds
    - Consisting of up to 139 tests
Evaluation Metric

- GainedRunOrder
- PercentageGain
\textbf{GainedRunOrder}

\texttt{GAINEDRUNORDER} (A) = RunOrderFail(FIFO) − RunOrderFail(A)
PercentageGain

\[
\text{PercentageGain}(A1, A2) = 1 - \frac{\text{GAINEDRunOrder}(A1)}{\text{GAINEDRunOrder}(A2)}
\]
Experimental Results
## Median GainedRunOrder Results

<table>
<thead>
<tr>
<th></th>
<th>Google</th>
<th>Chrome</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoogleTCP</td>
<td>8927</td>
<td>57</td>
</tr>
<tr>
<td>DynaQFocus</td>
<td>310</td>
<td>113</td>
</tr>
<tr>
<td>DynaQFocusFail</td>
<td>9407</td>
<td>221</td>
</tr>
</tbody>
</table>
Percentage Gain Against GoogleTCP

<table>
<thead>
<tr>
<th></th>
<th>Google</th>
<th>Chrome</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynaQFocus</td>
<td>-96.52%</td>
<td>98.24%</td>
</tr>
<tr>
<td>DynaQFocusFail</td>
<td>5.37%</td>
<td>287.71%</td>
</tr>
</tbody>
</table>
Discussion
Build Level Failure Distribution

- Ratio of failures in each build
  - If the majority of tests are failures => prioritization does not help
  - If there are a few failures per build => focusing idea does not help
- How many builds we have for different number of test failures?
Build-level Failure Distribution
Build-level Failure Distribution of Google
Build-level Failure Distribution of Chrome
Conclusion

- We hypothesized that test failures cluster and the focusing idea might help
- DynaQFocusFail performs the best
- Results on Chrome are better than Google because of the failure ratio
- We have to consider failure distribution for the future designs

emad.fallahzadeh@concordia.ca  peter.rigby@concordia.ca
Google test failures time series
Chrome test failures time series
Google Dataset Failures to Passes
Chrome Dataset Failures to Passes