Faster Feedback through Concept-based Test Prioritization

Toni Mattis, Robert Hirschfeld
Research School for Service-Oriented Systems Engineering
Hasso Plattner Institute, University of Potsdam, Germany

HPI Nanjing Workshop 3 – 4 Sept. 2019, Nanjing, China
Testing in Software Engineering

Unit Test

1. Runs a small part of the code with example input
2. Checks if output is as expected

Code
(Production)

Tests
Testing in Software Engineering

Unit Test

1. Runs a small part of the code with example input
2. Checks if output is as expected

Code (Production) Tests
Goal: Immediate Feedback

Code/Change  Tests  Test execution

time to detect fault
Goal: Immediate Feedback

Code/Change  Tests  Test execution

time to detect fault
Lexical Test Prioritization

```
{  


}
```

- `return ...`
+ `if session.user.is_admin:`
+ `return ...`

```
assert get(...).status == 403
with login(admin_user):
    assert get(...) == ...
```

Lexical similarity
Hypothesis

Test cases that *share vocabulary* with the most recent *change* are more likely to fail

Approach

1. Seed faulty changes
2. Run tests
3. Re-order tests based on lexical similarity
4. Check how much earlier failures occur
Fault Seeding

**Version N**

```plaintext
{ ...
  ...
  ...
}
```

**Diff**

- 
+ 
+

**Version N + 1**

```plaintext
{ ...
  ...
  ...
}
```

**Faulty Diff**

- 
+ 
+

**Mutant N + 1**

```plaintext
{ ...
  ...
  ...
}
```
Fault Seeding

Version \( N + 1 \)

Control Run

Sampling Run

Mutant \( N + 1 \)
Fault Seeding

Negate condition

```python
if session.user.is_admin:
    ...
```

```python
if not session.user.is_admin:
    ...
```

Swap operator

```python
average = total / count
```

```python
average = total * count
```

Change number

```python
response.status = 404
```

```python
response.status = 405
```

Drop call

```python
user_profile.save()
return redirect(...)
```

```python
return redirect(...)
```
Feature Extraction

```python
if not session.user.is_admin:
    return ...

assert get(...).status == 403
with login(admin_user):
    assert get(...) == ...

{ admin, user, session ... }
{ admin, user, status, login, ... }
```
Prioritization

Mutant

{ admin, user, ... }

Sampling Run

{ login, user, password, ... }

{ render, template, ... }

{ admin, user, ... }

TF-IDF:

#Tests with word w

IDF weight

blueprint

html

this
Prioritization

Mutant

\{
\{ \text{admin, user, …} \}
\}

Sorted sampling run results

\{ login, \text{user}, password, … \}

\{ render, template, … \}

\{ admin, user, … \}
Case Study: Flask

Python web framework, 74 diffs, 413 seeded faults

Seeded faults detected vs. Execution time [seconds]

- 79% ranked faults
- 93% seeded faults detected
- 8% untreated faults
- 88% seeded faults detected
Case Study: Flask

Python web framework, 74 commits, 413 seeded faults

Test run profiles for Flask / UNT

Test run profiles for Flask / LDA

Test Executions (untreated)

Test Executions (ranked)
Case Study: Flask

Limitation: **Pull Requests**
- Largest type of “change”
Case Study: Flask

Limitations:

» Pull Requests

» Distinguishing names ("NoAppException") split into generic words ("no", "app", "exception")
Exploiting Topicality

Changed identifiers: 
user, password, check

Test to prioritize: 
login, email, password

Topic/Concept Model
Exploiting Topicality

Changed identifiers: 
user, password, check

Test to prioritize: 
login, email, password
Approximating the Call Graph

Method 1

Method 2

Method 3

Test transitively relates to Topic 2

Test

Change

Topic 1

Topic 1

Topic 1

Topic 2

Topic 2

Topic 2
Approximating the Call Graph

Method 1

Method 2

Method 3

Test

Markov Chain

0.4 0.6

0.1 0.9
Live Testing Tools

**AutoTDD** runs a selected *set of tests* whenever another selected *set of code locations* is changed.

---

**AutoTDD with Travis-CI Support**

**Installation**

1. Make sure you have `metacello-work` installed.
2. Load the project with:

```
Metacello new
baselines: ['AutoTDD'];
```
Conclusion

» **Change-based fault seeding** is an effective method to generate many failures distributed like actual changes

» **Lexical information** can be exploited to quickly guess which tests may fail

» There is more potential in exploring **topicality** and **Markov properties** of the vocabulary