Lexical Test Prioritization for Faster Feedback

**Toni Mattis**, Falco Dürsch, Robert Hirschfeld

Software Architecture Group
Hasso Plattner Institute, University of Potsdam, Germany

**PX/19** 1 Apr. 2019, Genoa, Italy
“Unit tests are not live!”

**Liveness:**
“Impression of changing a program while it’s running”

» Test results: observable property of the program

» Immediate & continuous feedback
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 $N$

Diff

Version $N + 1$

Faulty Diff

Mutant $N + 1$
Fault Seeding

Version N + 1

Mutant N + 1

Control Run

Sampling Run
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())
```

```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

Sampling Run

TF-IDF:

IDF weight

#Tests with word w

{ admin, user, ... }

{ login, user, password, ... }

{ render, template, ... }

{ admin, user, ... }

blueprint

json

self
Prioritization

Mutant

Sorted sampling run results

{ admin, user, ... }

{ login, user, password, ... }

{ render, template, ... }

{ admin, user, ... }
Comparison: Immediacy

Probability of detecting a fault immediately (after 0.1/1.0 seconds)
Case Study: Flask

Python web framework, 74 commits, 413 seeded faults

Seeded faults detected vs. Execution time [seconds]

- Ranked: 79%
- Untreated: 8%
- 93% detected
- 88% ranked
Case Study: Flask

Python web framework, 74 commits, 413 seeded faults
Case Study: Flask

Limitation: **Pull Requests**

» Largest type of “change”

276 features
Case Study: Flask

Limitations:

» Pull Requests

» Distinguishing names (“NoAppException”) split into generic words (“no”, “app”, “exception”)

```
def locate_app(script_info, module_name, app_name, raise_if_not_found):
    raise NoAppException('Could not import "{name}"'.format(name=module_name))
```

```
def test_locate_app_raises(test_apps, iname, aname):
    info = ScriptInfo()
    with pytest.raises(NoAppException):
        locate_app(info, iname, aname)
```
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
baseline: 'AutoTDD';
```
Future Work

» Combination with coverage-based prioritization
» Tradeoffs (where does vocabulary mislead?)

Real-world Projects:
The most recently failed test is most likely to fail again!
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 combinations with related work and the actual failure history.