Lexical Test Prioritization for **Faster Feedback**

Toni Mattis, Falco Dürsch, Robert Hirschfeld

Software Architecture Group

Hasso Plattner Institute, University of Potsdam, Germany

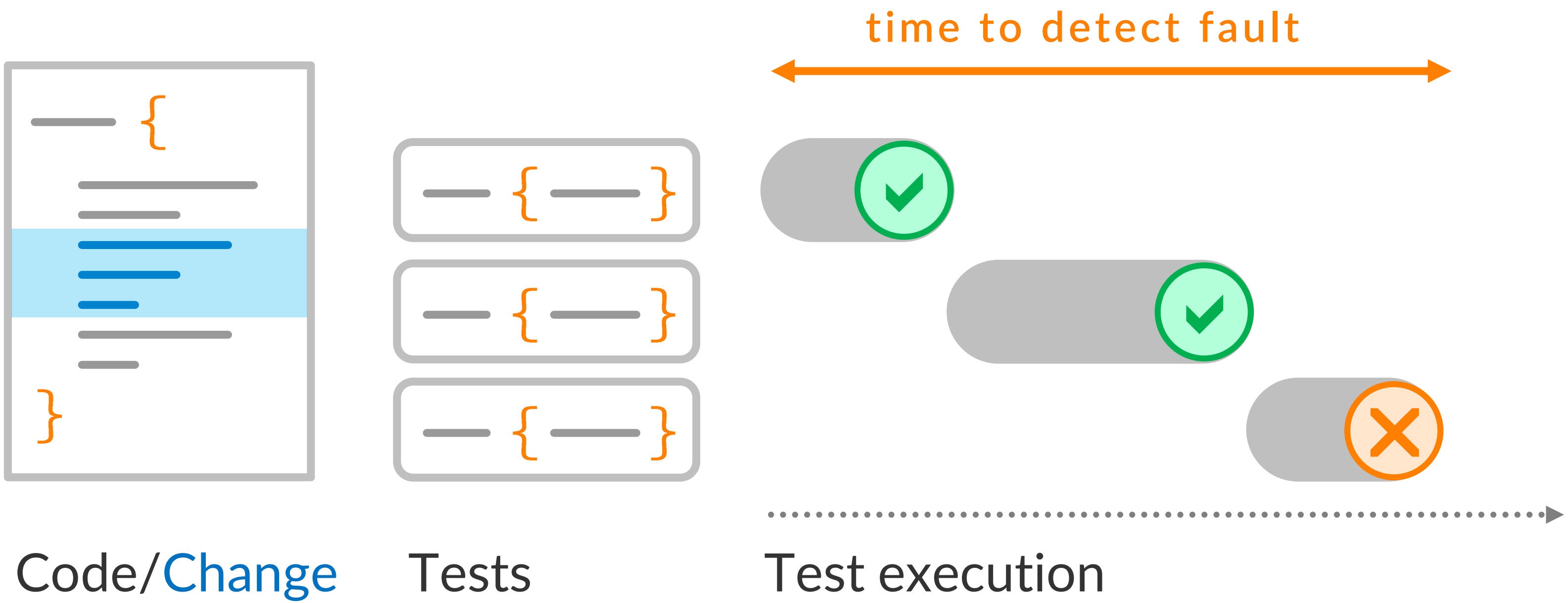
PX/19 1 Apr. 2019, Genoa, Italy

“Unit tests are not live!”

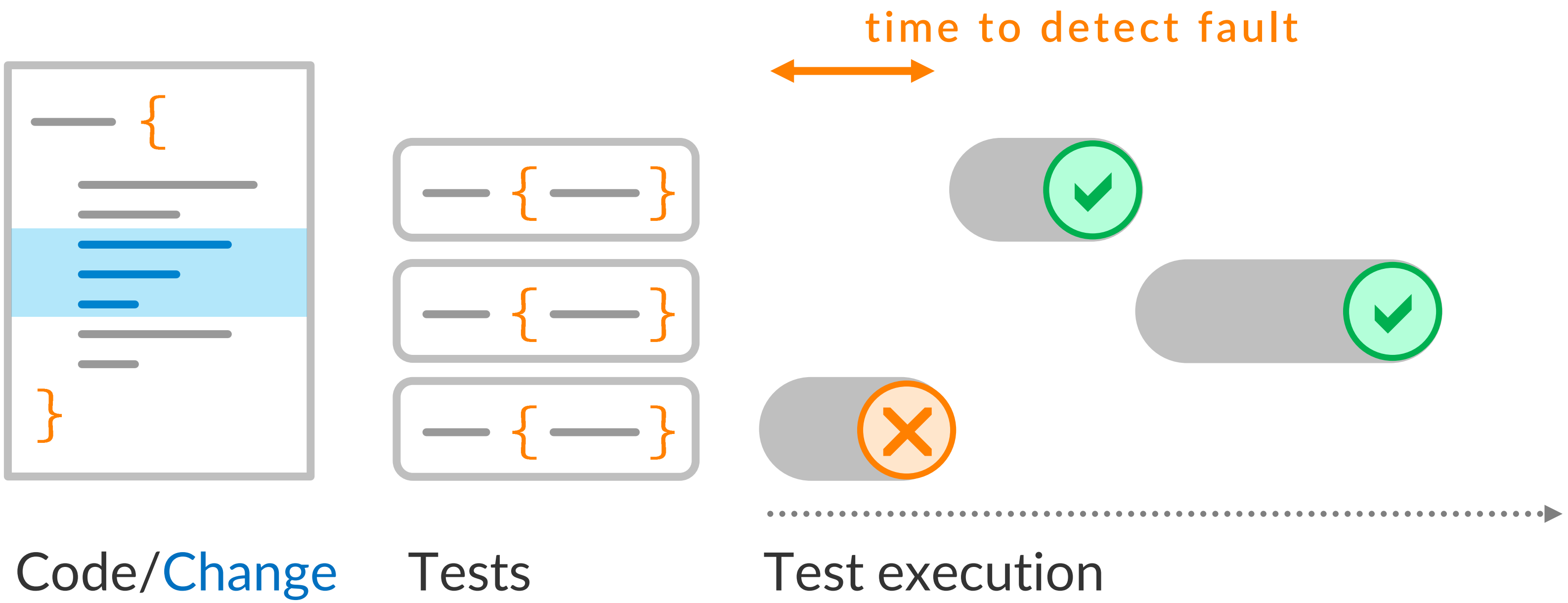
Liveness:
“Impression of changing a program while it’s running”

- Test results: observable property of the program
- **Immediate & continuous feedback**

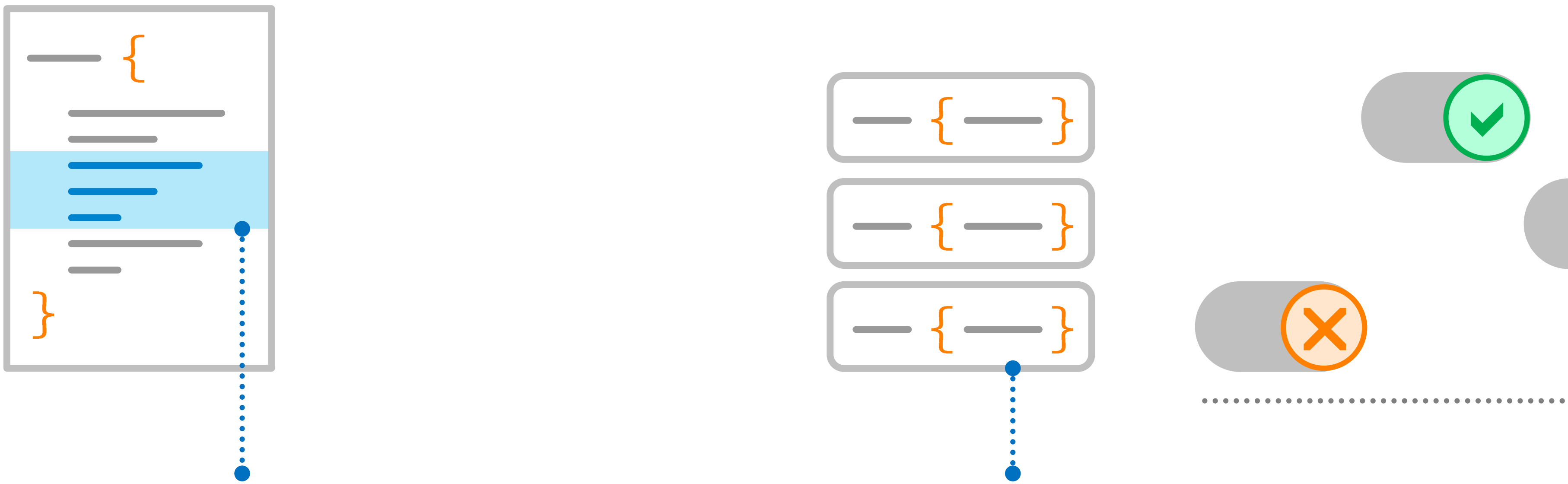
Goal: Immediate Feedback



Goal: Immediate Feedback



Lexical Test Prioritization



```
- return …

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

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

Lexical similarity

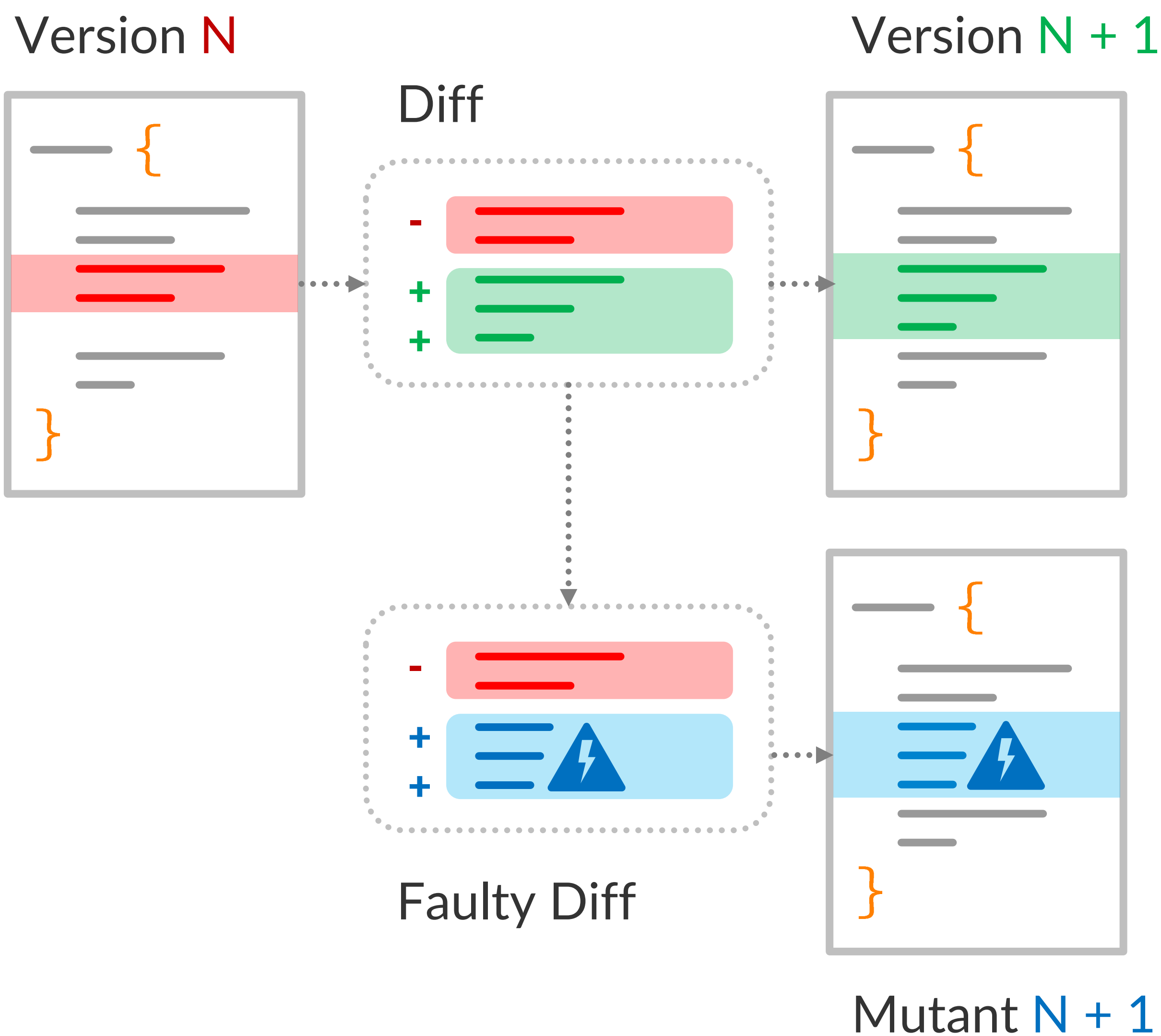
Hypothesis

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

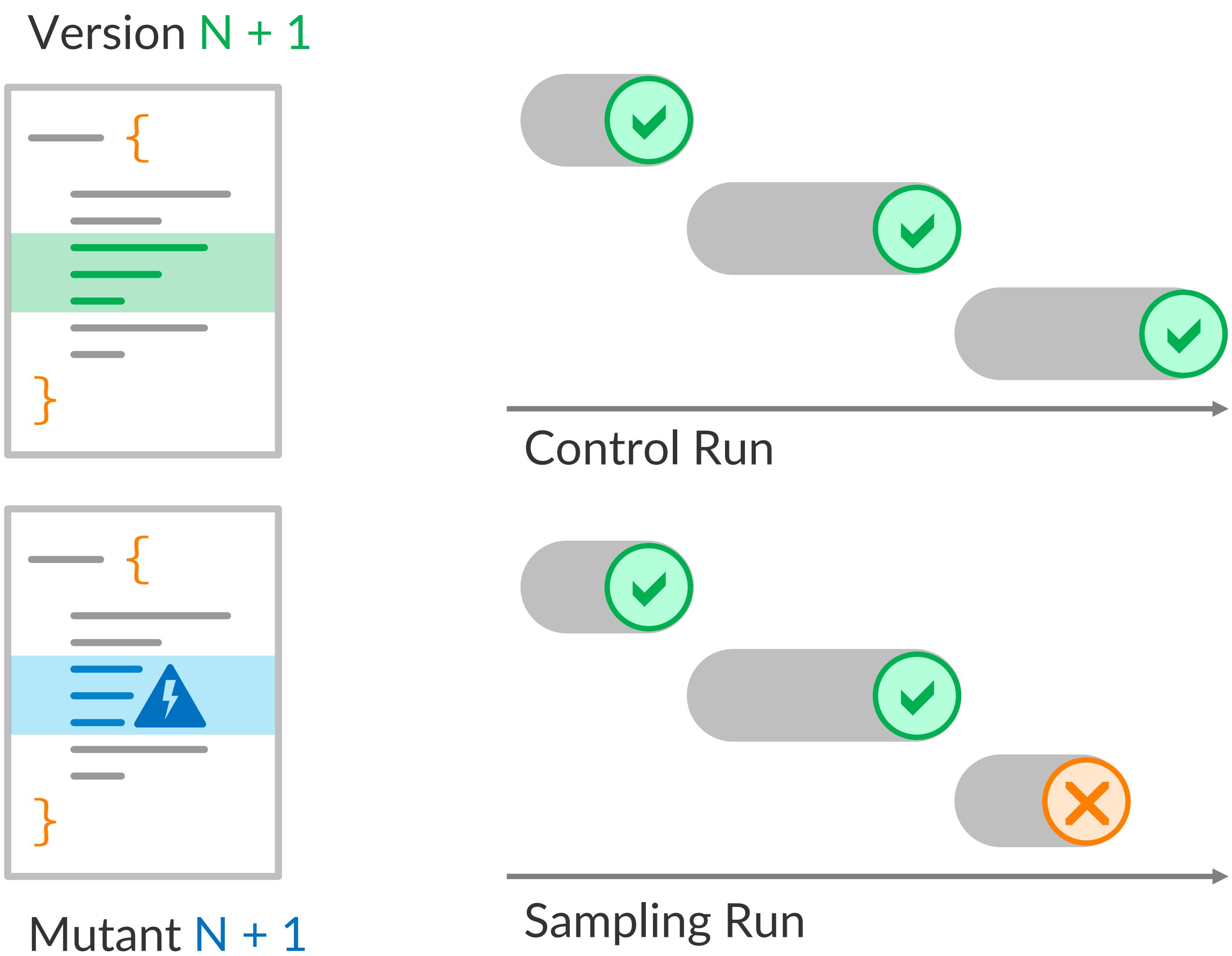
Approach

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

Fault Seeding



Fault Seeding



Fault Seeding

Negate condition

```
if session.user.is_admin:
  …
```

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

Swap operator

```
average = total / count
```

```
average = total * count
```

Change number

```
response.status = 404
```

```
response.status = 405
```

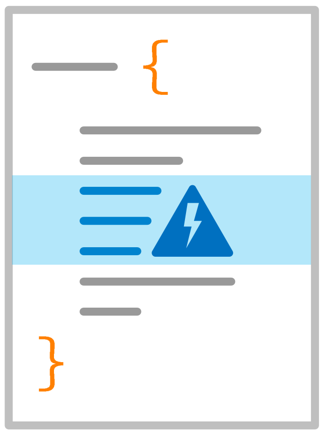
Drop call

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

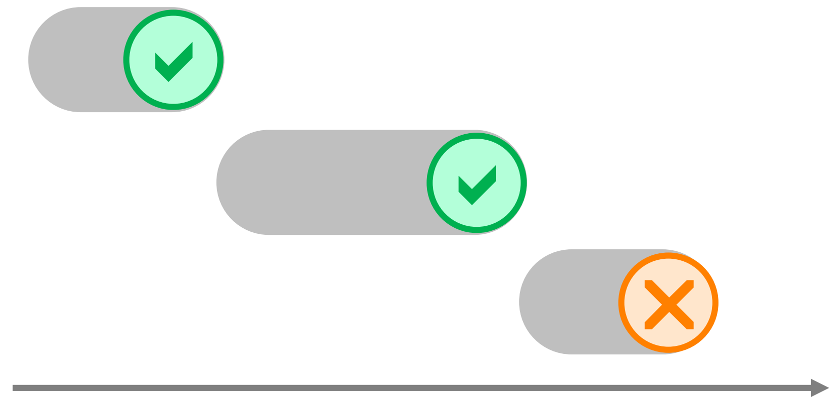
```

return redirect(…)
```

Feature Extraction



Mutant



Sampling Run

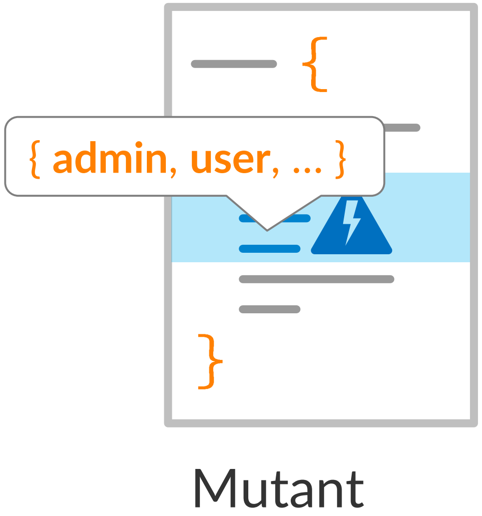
```
if not session.user.is_admin:
  return …
```

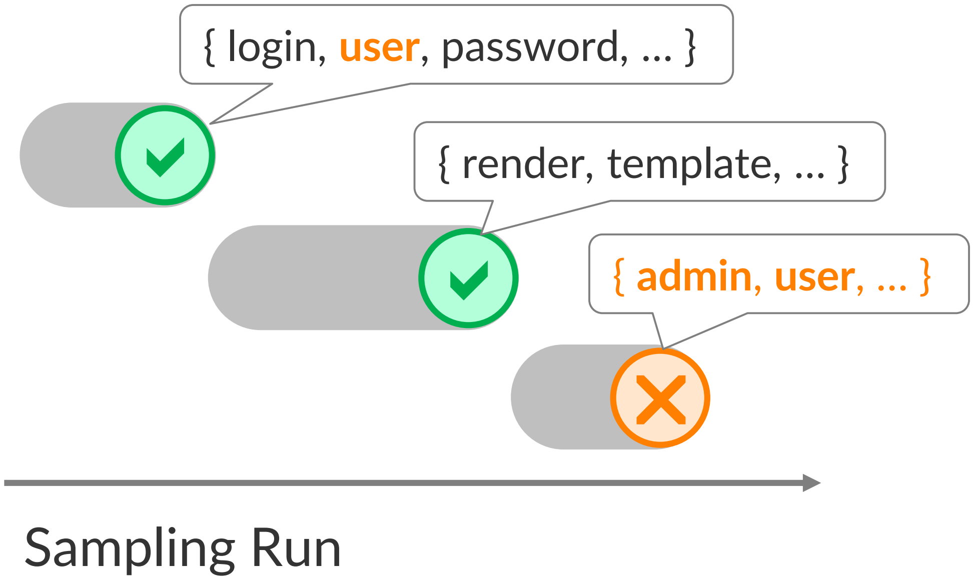
{ admin, user, session … }

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

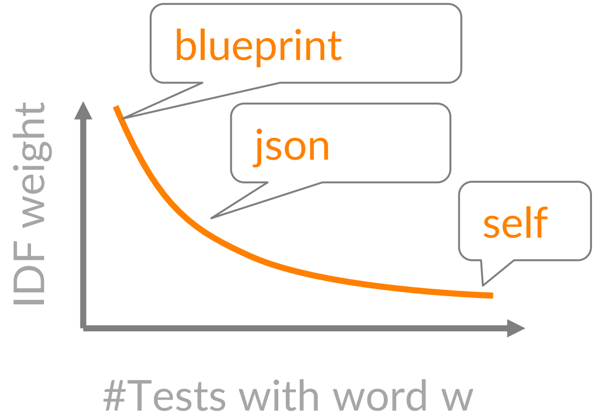
{ admin, user, status, login, … }

Prioritization

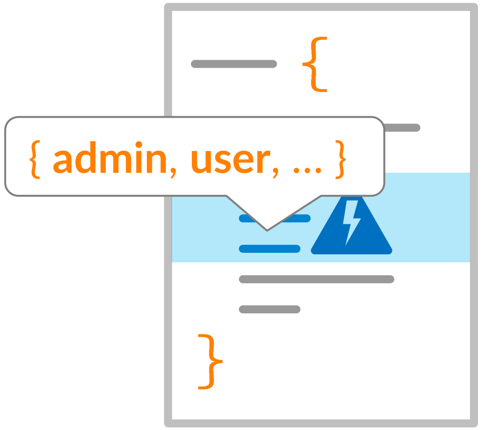




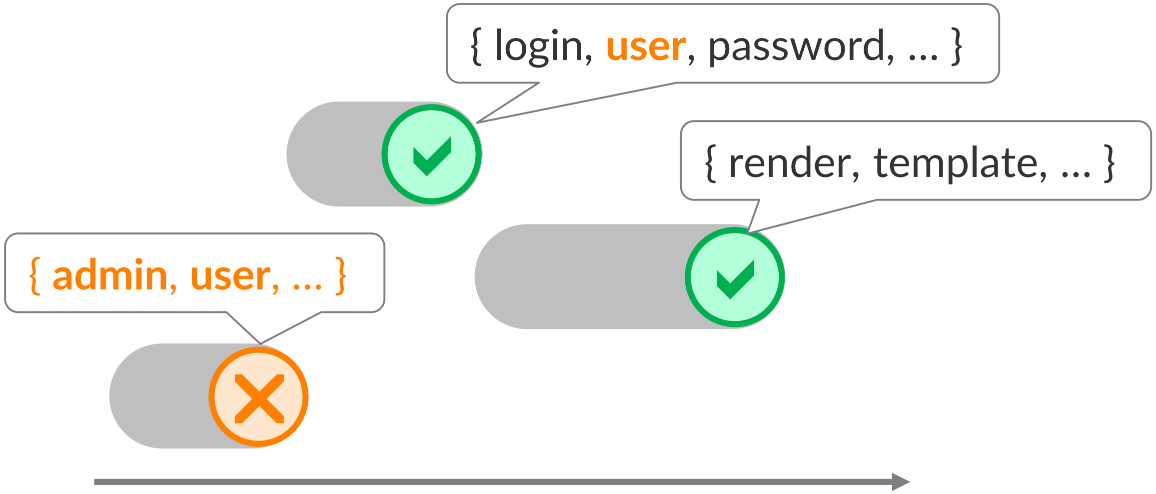
TF-IDF:



Prioritization

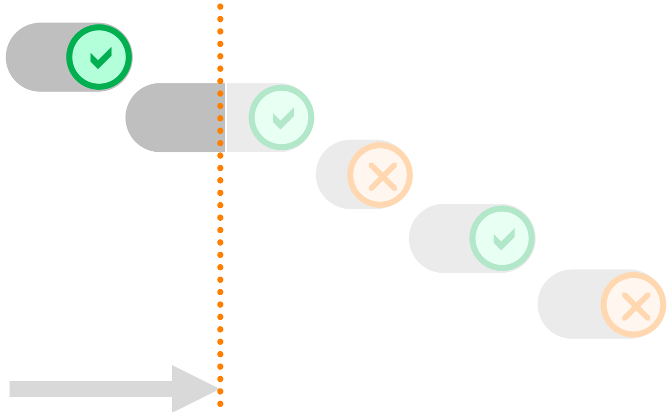


Mutant

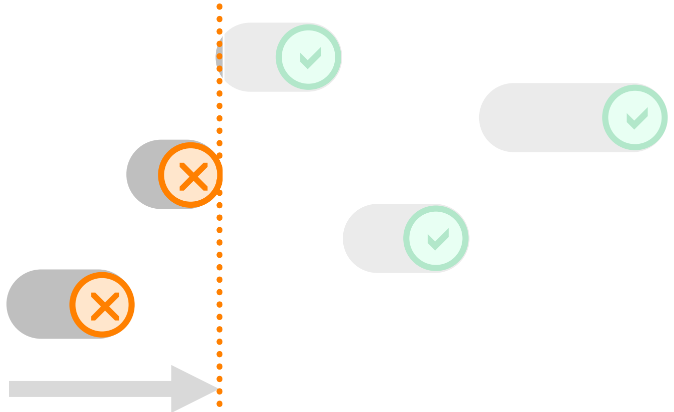


Sorted sampling run results

Comparison: Immediacy



0.1 seconds/
1.0 second

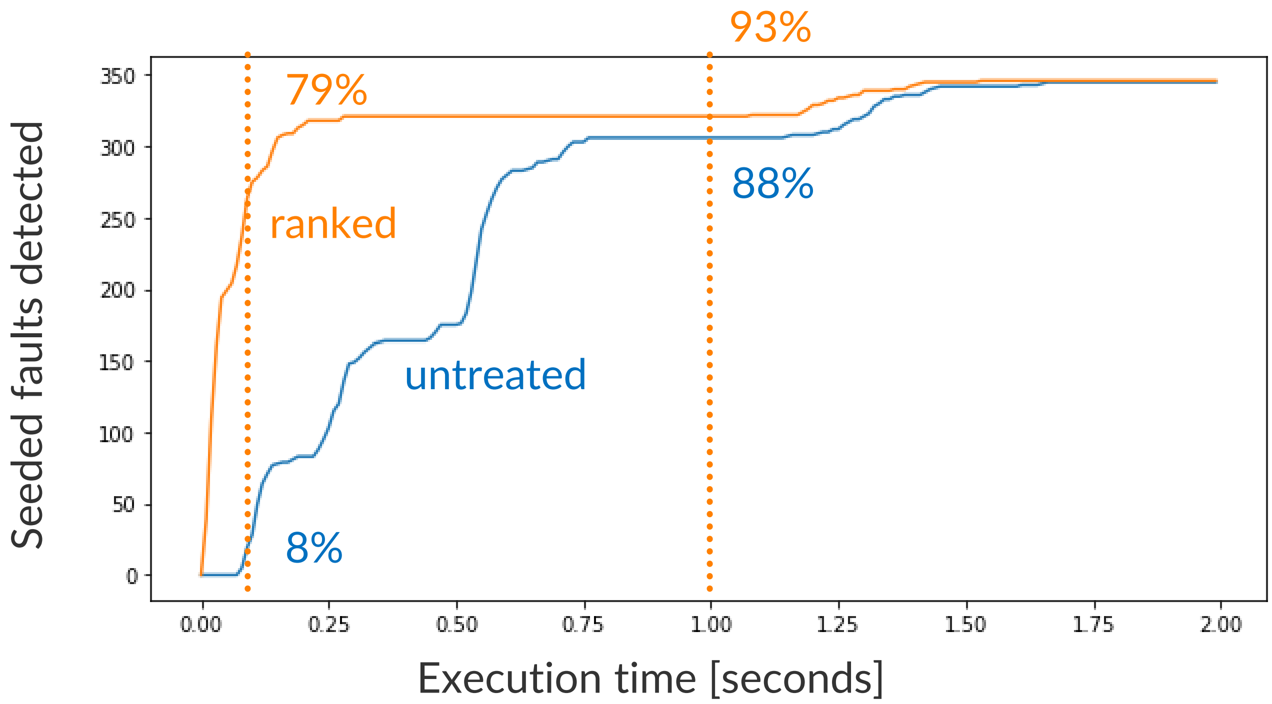


0.1 seconds/
1.0 second

Probability of detecting a fault immediately
(after 0.1/1.0 seconds)

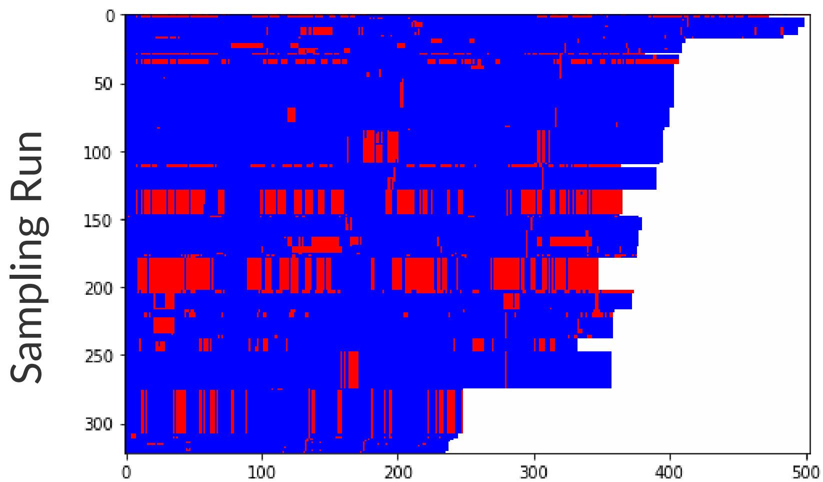
Case Study: Flask

Python web framework, 74 commits, 413 seeded faults

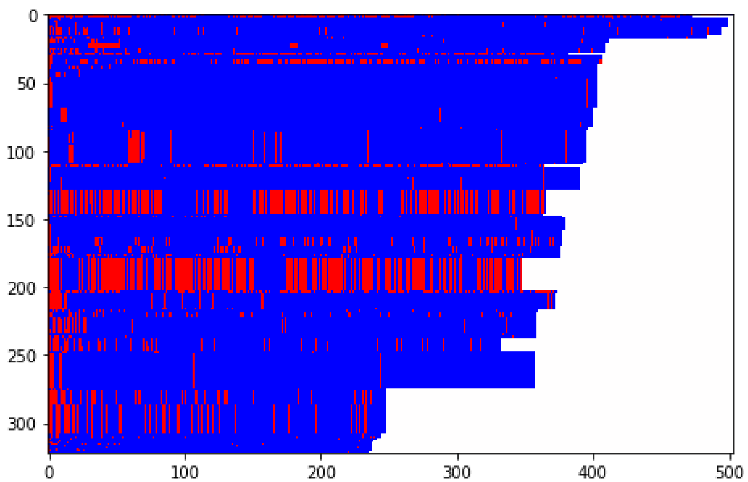


Case Study: Flask

Python web framework, 74 commits, 413 seeded faults



Test Executions (untreated)

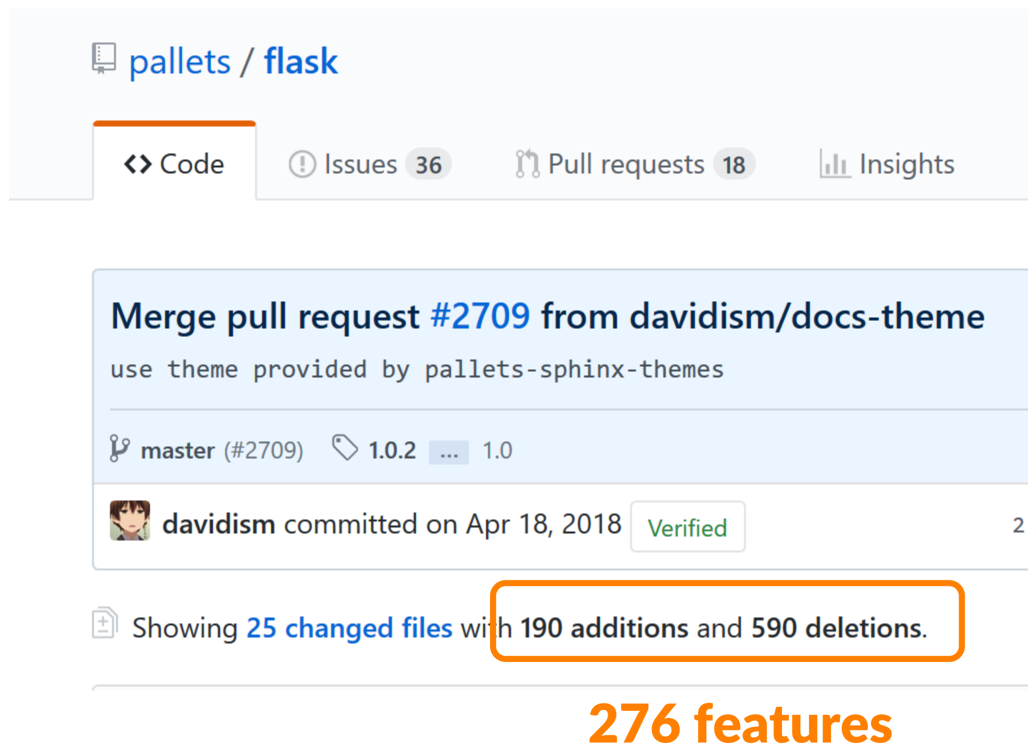


Test Executions (ranked)

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”)

https://github.com/pallets/flask/commit/5fba092c22bee738ad2818d587d75ae17

2 flask/cli.py

```
@@ -243,7 +243,7 @@ def locate_app(script_info, module_name, app_name, raise_if_not_
                    )
                elif raise_if_not_found:
                    raise NoAppException(
-                       'Could not import "{name}"."'.format(name=module_name)
+                       'Could not import "{name}".'.format(name=module_name)
                    )
                else:
                    return
```

of 276...

‘no’, 'app’, 'exception’,

```
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

Future Work

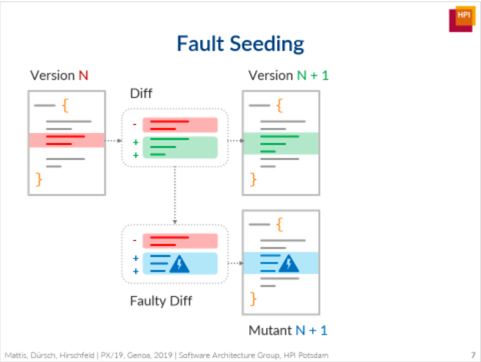
- Combination with coverage-based prioritization
- Tradeoffs (where does vocabulary mislead?)

Real-world Projects:

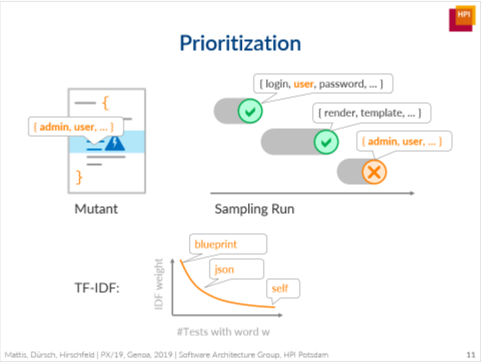
The **most recently failed test** is most likely to fail again!

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**