Report on Automatic Test Generation

Diffblue vs. SquareTest

The Source Code for each testing tool can be found branches `diffblue`, and `squaretest`.

Similarities

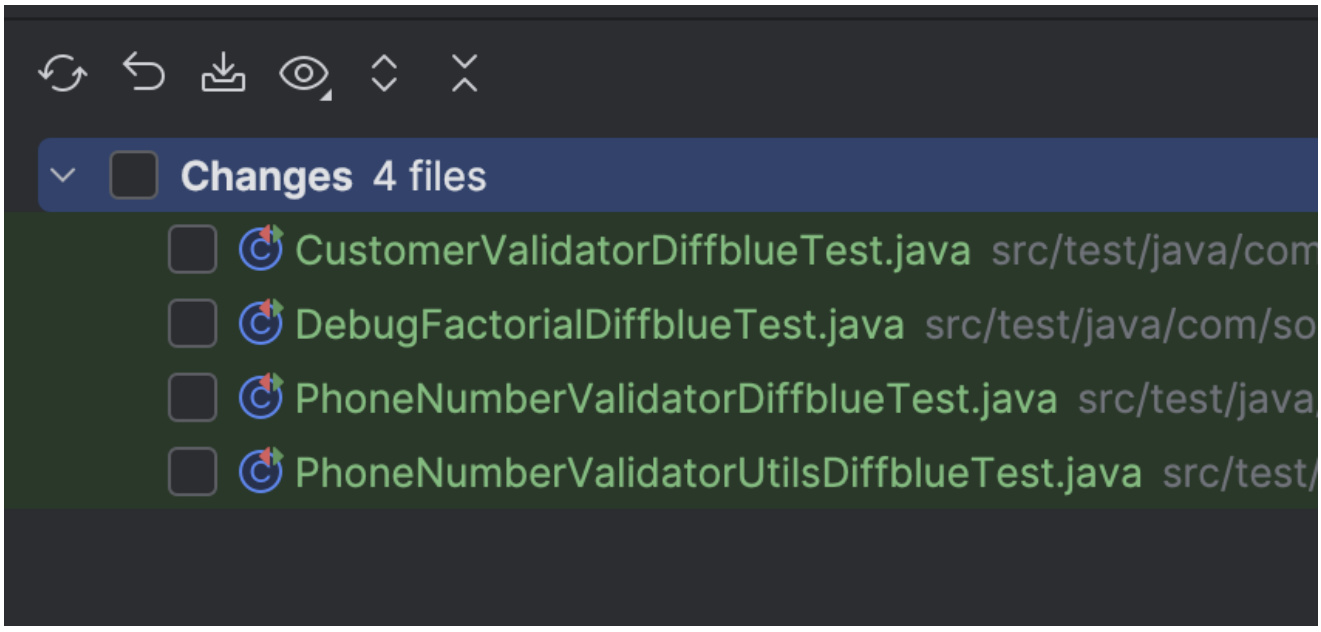
Both generate JUnit tests automatically and have IntelliJ integrations.

Differences

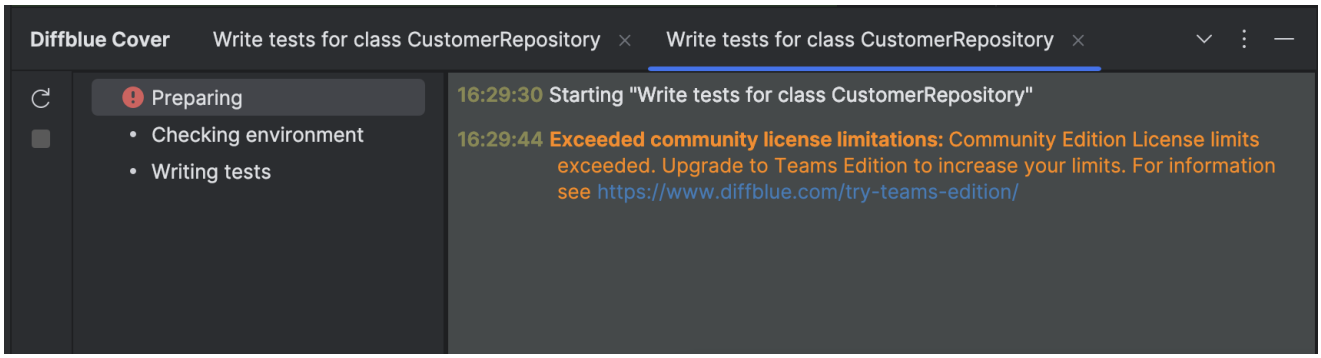
- Massive Pricing Differences
- Diffblue appears as a bigger and advanced company
- Diffblue uses reinforcement learning
- SquareTest uses templating and rule-based-algorithms
- SquareTest is IntelliJ only. Diffblue also has support for the command line, CI, and analytics. The free tier is only a intellij plugin, which is rate limited.

Diffblue

After creating tests for the files listed below, I have encountered numerous rate-limiting issues.
Old tests are deleted and Diffblue tests added.



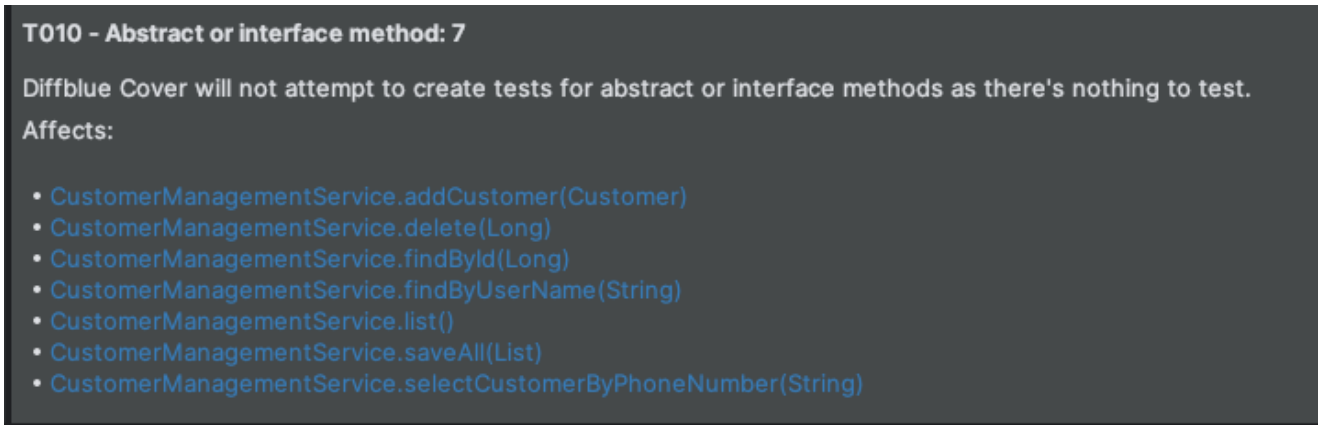
Ex. Diffblue rate limited me for the next file.



Issues encountered with Diffblue

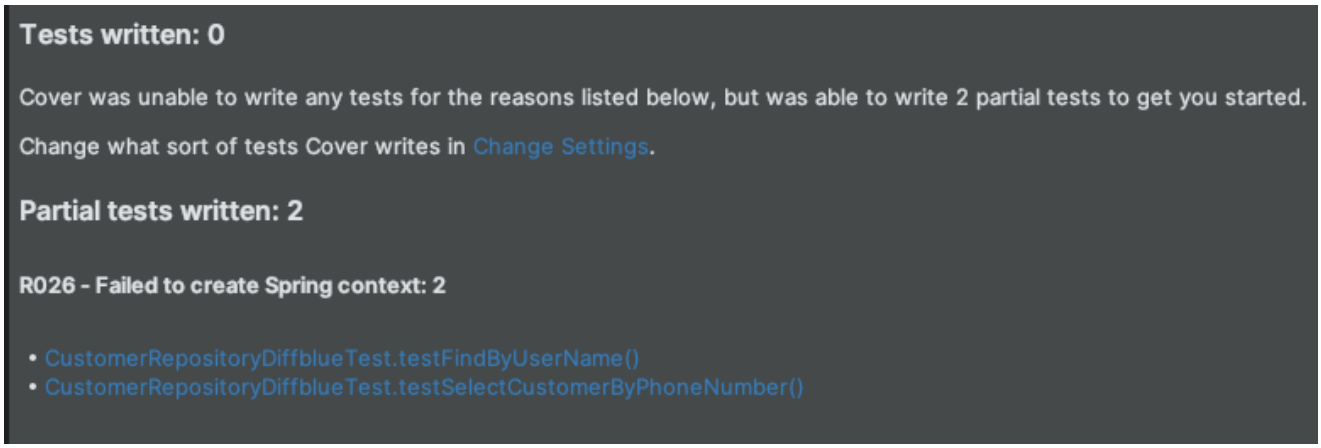
`CustomerManagementService.java`

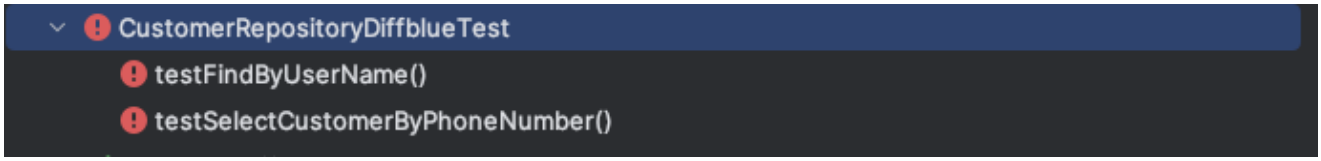
Diffblue appears to be picky when it comes to generating tests for abstract or interface methods.



`CustomerRepository.java`

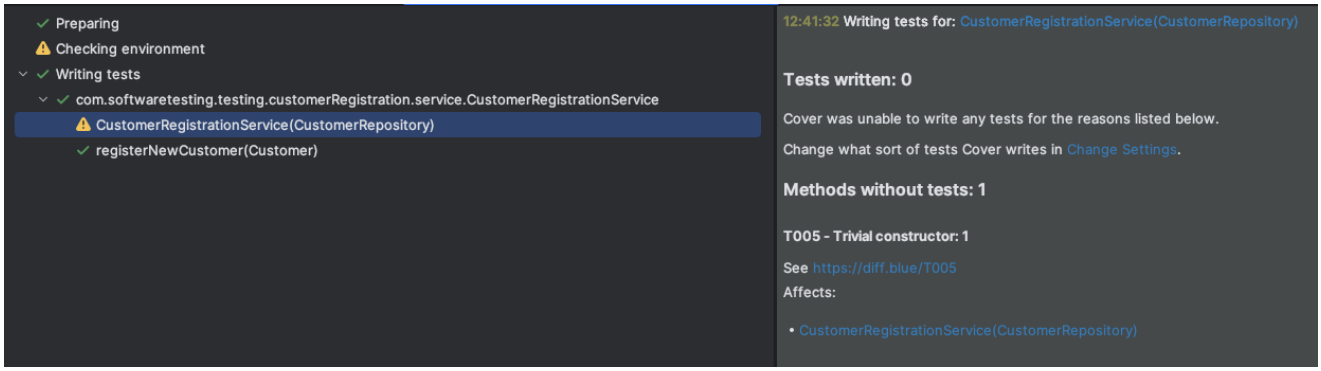
Diffblue requires some extra configuration, or will just refuse to write "difficult" test cases. The tests will be generated, but will fail on execution.





`CustomerRegistrationService.java`

Also, Diffblue refuses to create tests for "trivial" code.



Mutation Coverage

Removing the partial tests, we come to the following result:

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
13	43%	82/190	58%	60/104	88%	60/68

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.softwaretesting.testing.config	1	0%	0/16	0%	0/4	0%	0/0
com.softwaretesting.testing.customerManagement.controller	1	0%	0/12	0%	0/6	0%	0/0
com.softwaretesting.testing.customerManagement.service	1	17%	4/23	17%	2/12	100%	2/2
com.softwaretesting.testing.customerRegistration.controller	1	0%	0/6	0%	0/1	0%	0/0
com.softwaretesting.testing.customerRegistration.service	1	100%	11/11	100%	3/3	100%	3/3
com.softwaretesting.testing.dto.inbound	1	0%	0/15	0%	0/7	0%	0/0
com.softwaretesting.testing.dto.outbound	1	0%	0/23	0%	0/4	0%	0/0
com.softwaretesting.testing.model	1	64%	16/25	75%	12/16	92%	12/13
com.softwaretesting.testing.util	2	83%	25/30	84%	32/38	86%	32/37
com.softwaretesting.testing.validator	3	90%	26/29	85%	11/13	85%	11/13

Report generated by PIT 1.8.0

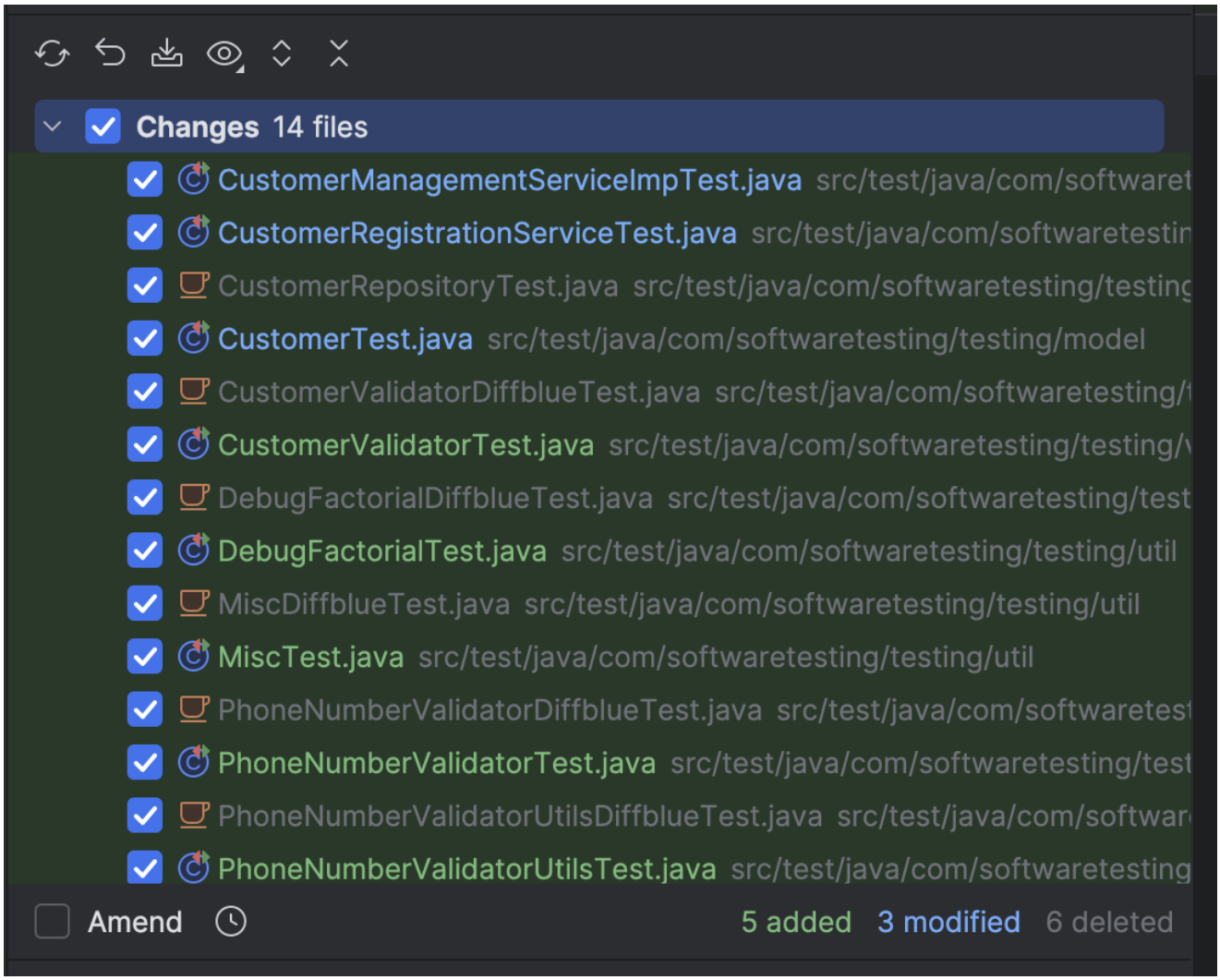
Comparing to the PIT report with manually written tests, we get a performance of:

- -19% Line Coverage
- -20% Mutation Coverage
- -11% Test Strength

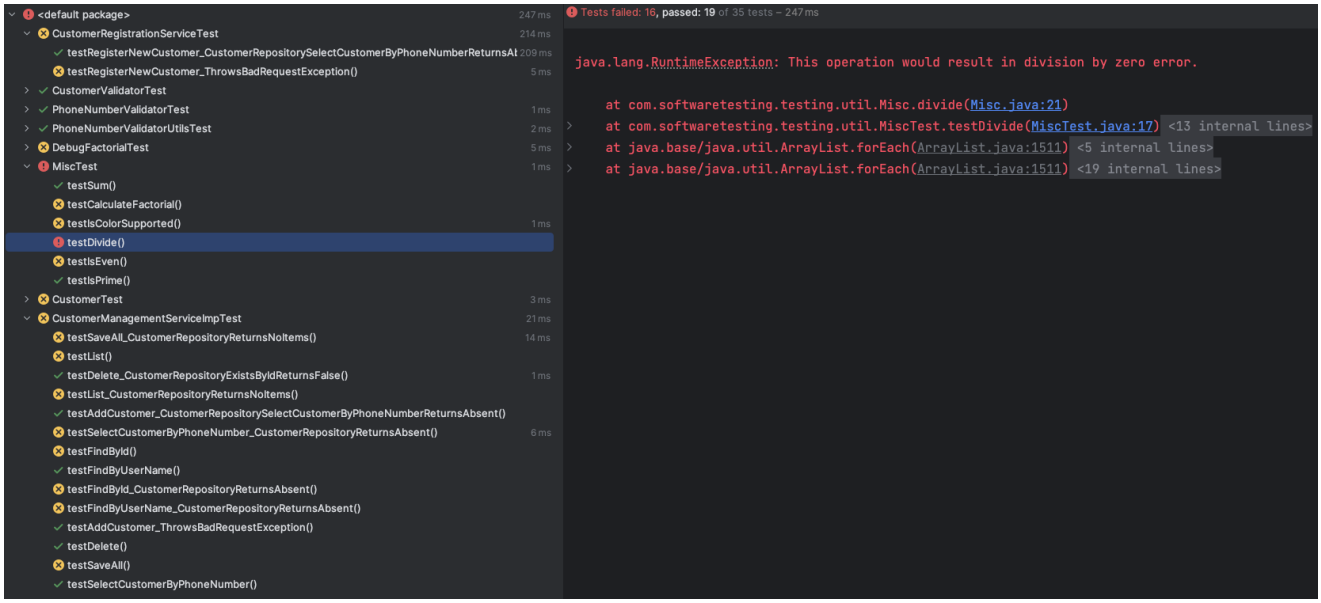
Squaretest

When using Squaretest after Diffblue, the speed performance is the first observation that one makes. WIth Diffblue Test generation for a class taking about 20-30 seconds to generate some tests for it, the generation of Squaretest is almost instantly.

Futhermore it appears taht Squaretest is picky about interfaces; it does not write tests for them. This is similar behaviour as seen on Diffblue.



And like Diffblue, we get some tests which are failing. However, the ratio of failing tests is much higher.



16 Failed, 19 Passed of a total of 35 tests.

When taking a look at the meaningfulness of the tests generated by Squaretest, one example stands out:

```
class DebugFactorialTest {
        // ...
    @Test
    void testMain() {
        // Setup
        // Run the test        DebugFactorial.main(new String[]
{"args"});

        // Verify the results
    }
}
```

But overall I found the structure of the tests generated by Squaretest more deterministic.

Removing the failing tests, we come to the following result:

Category	Manual	Diffblue	Squaretest	Winner
Line Coverage	62%	43%	37%	Manual
Mutation Coverage	78%	58%	25%	Manual
Test Strength	99%	88%	65%	Manual

Conclusion

Squaretest performs with the exception of generational speed worse then Diffblue. The tests generated by both tools seem to cover the bare necessities of a test suite, but do not necessarily meet the expectations of a comparable human developer.

The lack of feature coverage on interfaces and abstract classes is present in both tools and both tools even generate failing tests. The quality of the test cases seem not particularly useful as they leave the "difficult" and "hard to Google" things out. Test generation over those tools compromises the developer experience, by firstly generating tests which must be understood by a developer but also generating tests, which are to be corrected by an experienced human developer. Therefore, those tools seem to use up more time than just writing the tests manually.