In-Depth Look at Mutation Testing, Property-Based Testing, and Unit Testing

Jason Walsh <j@wal.sh>

Introduction to Testing Techniques (5 min)

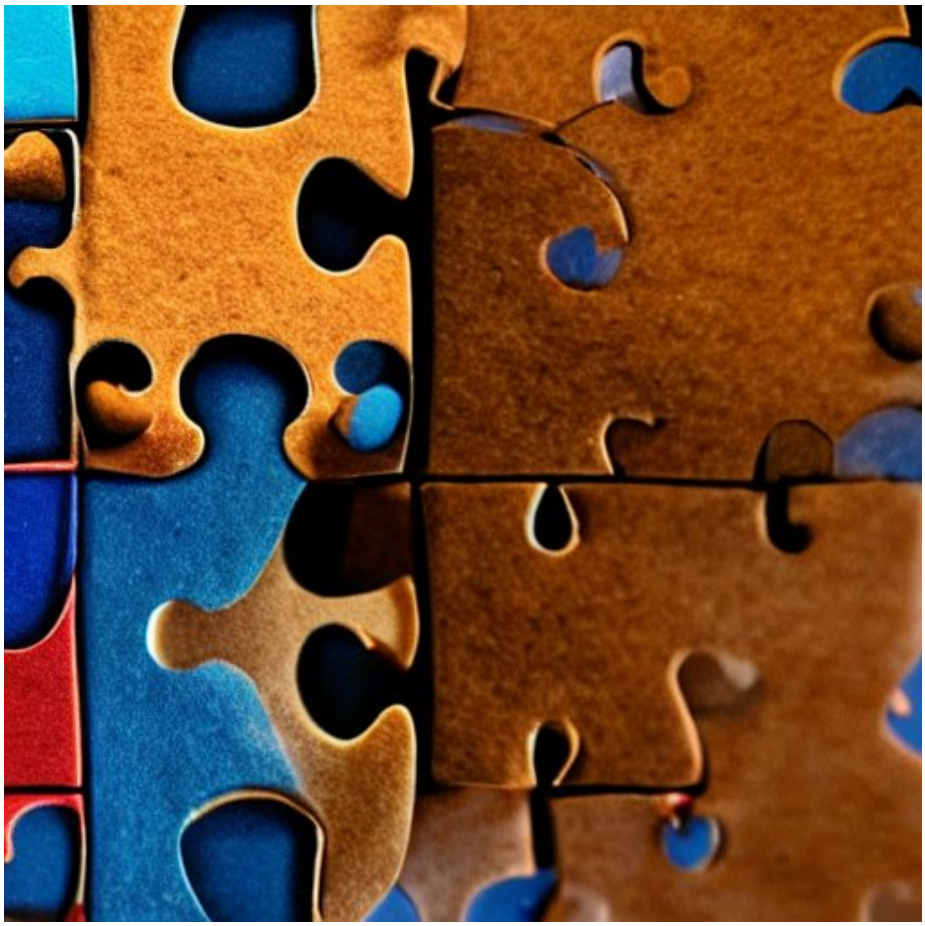

Figure 1: Puzzle pieces fitting together representing different testing techniques

- Explanation of mutation testing, property-based testing, and unit testing.
- How these techniques complement each other.

Mutation Testing with PIT (Java & Kotlin) (10 min)

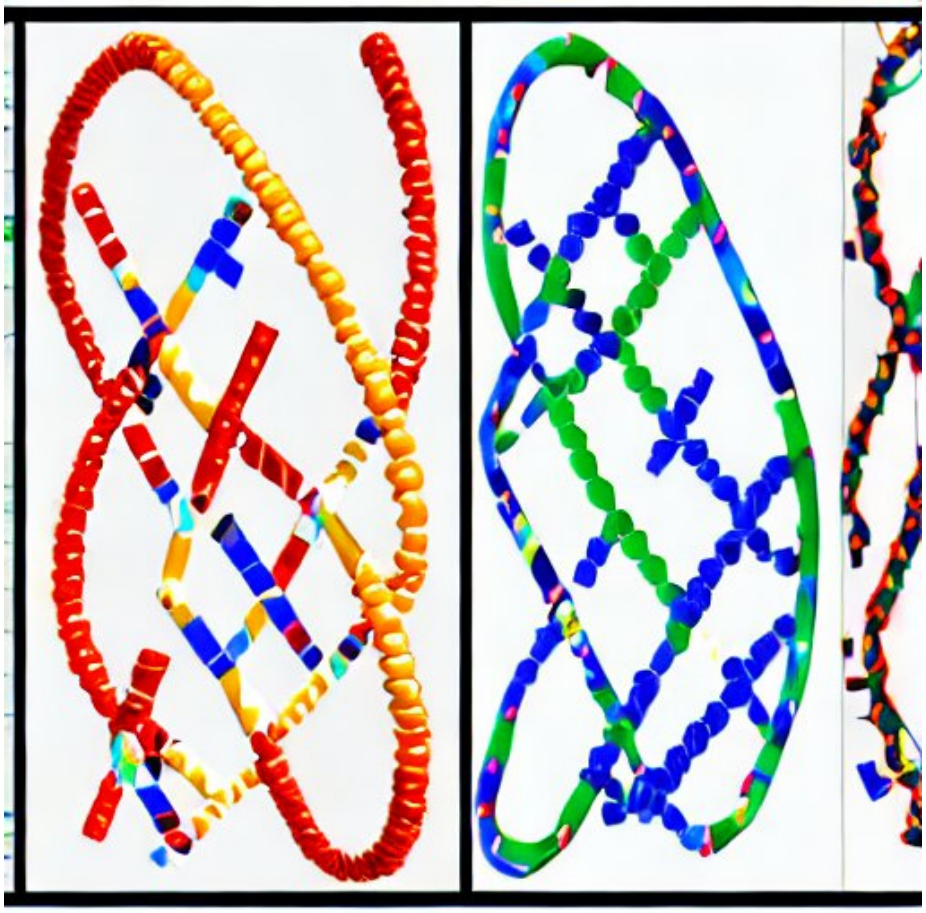

Figure 2: DNA strands with mutations representing code mutation testing

- Introduction to PIT and its features.
- Code Example: Calculator (Java)

```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
```

```
    }
    // ... More methods
}
```

- Code Example: Temperature Converter (Kotlin)

```
class TemperatureConverter {
    fun celsiusToFahrenheit(celsius: Double): Double {
        return (celsius * 9.0 / 5.0) + 32.0;
    }
    // ... More methods
}
```

Mutation Testing with MutPy (Python) (10 min)

- Introduction to MutPy and its features.
- Code Example: String Reversal (Python)

```
def reverse_string(s):
    return s[::-1]
```

Property-Based Testing with Hypothesis (Python) (10 min)

- Introduction to Hypothesis and its features.
- Code Example: List Reversal (Python)

```
from hypothesis import given
import hypothesis.strategies as st

@given(st.lists(st.integers()))
def test_reverse_list(xs):
    assert xs == list(reversed(reversed(xs)))
```

Contract Programming in Racket (Scheme) (10 min)

- Introduction to contract programming in Racket.
- Code Example: Contracts for Functions (Racket)

```
(define/contract (add a b)
  (-> number? number? number?)
  (+ a b))
```

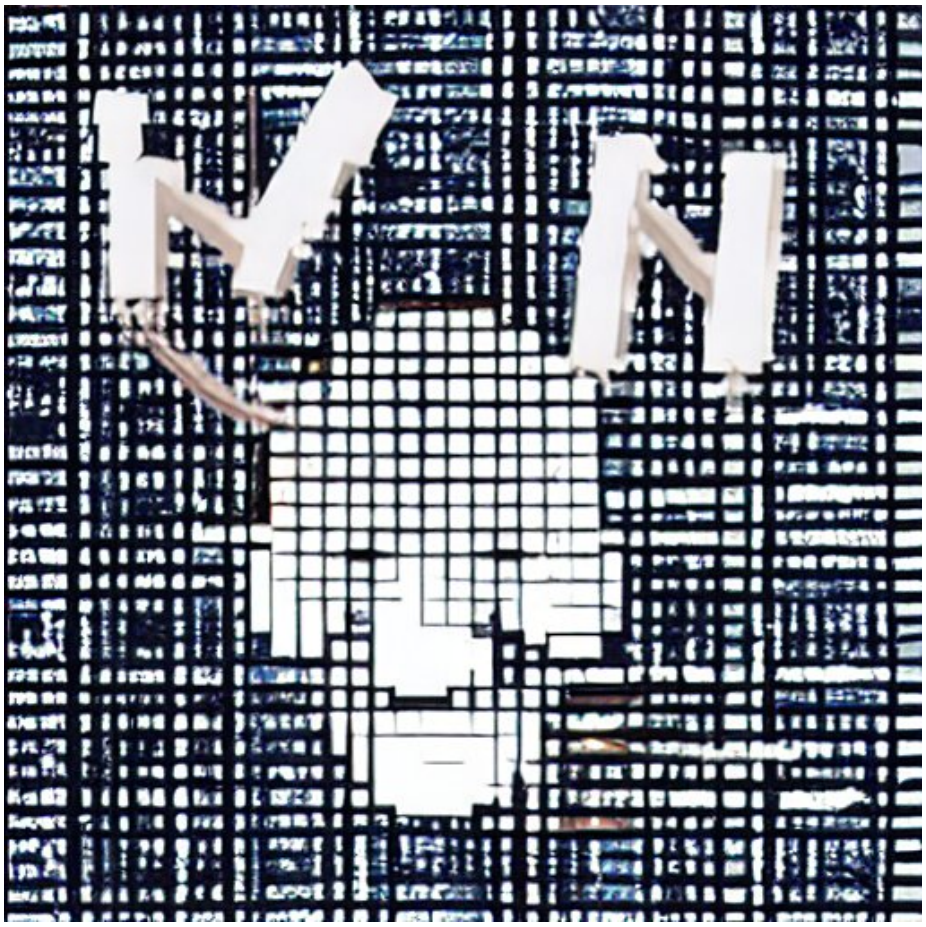

Figure 3: Two-faced character representing string reversal

Figure 4: Laboratory flask with a variety of random inputs

Figure 5: Handshake representing a formal agreement (contract)

clojure.spec and test.check (Clojure) (10 min)

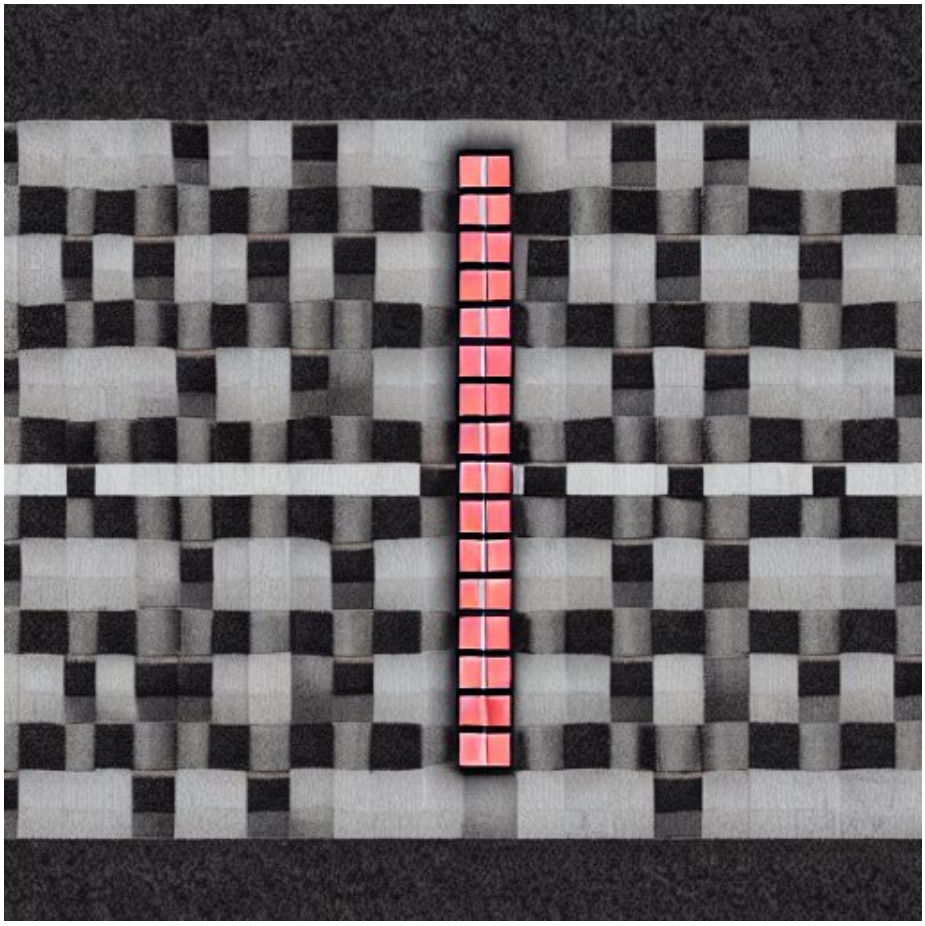

Figure 6: Checkerboard representing test.check testing and validation

- Introduction to clojure.spec and test.check.
- Code Example: Specifying Data and Functions (Clojure)

```
(require '[clojure.spec.alpha :as s])

(s/def ::age (s/int-in 0 150))
(s/def ::person (s/keys :req-un [::age]))
```

- Code Example: Property-Based Testing with test.check (Clojure)

```
(require '[clojure.test.check :refer [quick-check]])
(require '[clojure.test.check.generators :as gen])
```

```
(quick-check 100 (prop/for-all [v (gen/vector gen/int)]
                               (= (reverse (reverse v)) v)))
```

Standard Unit Testing (5 min)

Figure 7: Magnifying glass over code representing unit testing

- Explanation of unit testing and its purpose.
- Code Example: Factorial Function (Python)

```
import unittest

def factorial(n):
return 1 if n == 0 else n * factorial(n - 1)
```

```
class FactorialTest(unittest.TestCase):
def test_factorial(self):
self.assertEqual(factorial(5), 120)
self.assertEqual(factorial(0), 1)

if name == 'main':
unittest.main()
```

Comparison and Synergy of Techniques (5 min)

Figure 8: Scales balancing different testing techniques

- Comparing mutation testing, property-based testing, contract programming, and unit testing.
- Advantages and limitations of each approach.

Conclusion and Q&A (5 min)

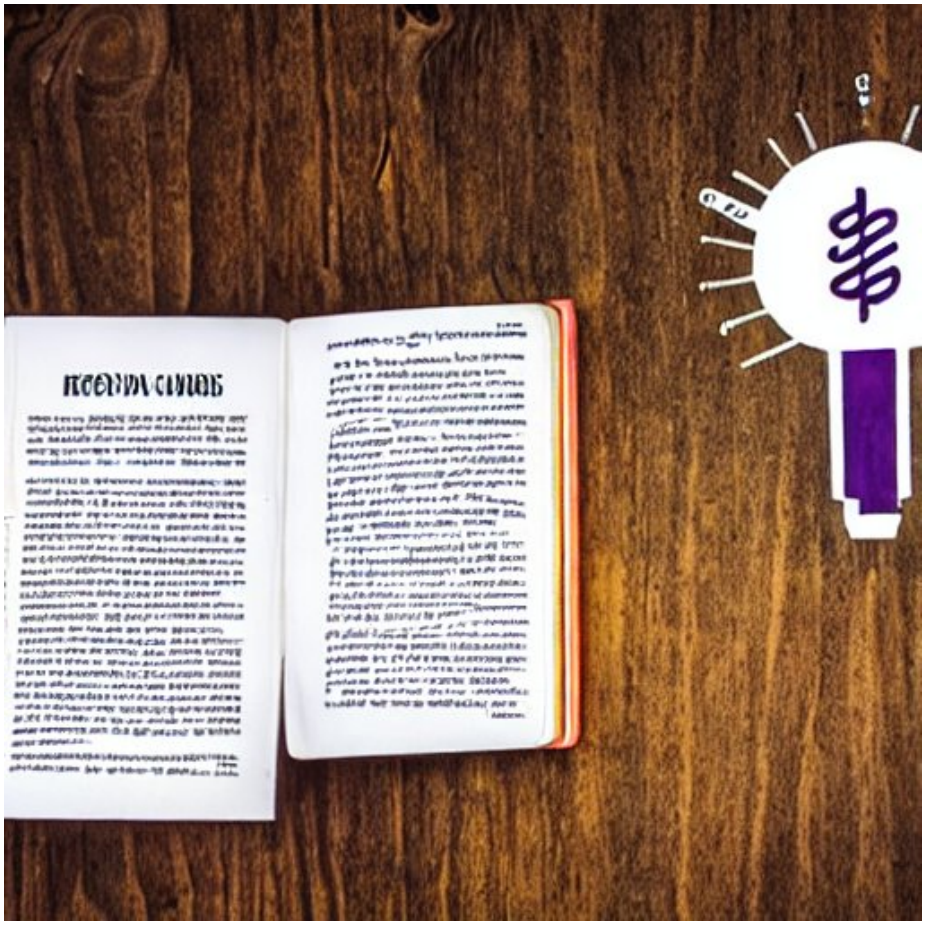

Figure 9: Summary book or a lightbulb representing key takeaways

- Recap of key points and takeaways.
- Open the floor for questions and discussion.