Software Testing (2)

September 22, 2022

Byung-Gon Chun

(Slide credits: George Candea, EPFL and Armando Fox, UCB)

Other Testing

- Acceptance testing
  - Performed by user upon receiving product
- Smoke/sanity testing
  - Quick test to check for serious errors
  - E.g., does it compile? Does it do the basic stuff?
- Compatibility testing
  - Does app work with other hw/sw?
  - E.g., web app with smartphone
- Fault injection
  - Does app work in the presence of bad inputs, bad returns from libraries, etc.?
  - Can inject exceptions, simulate failures
- Performance testing
  - Goal is to check performance characteristics
  - Load/stress/scalability testing
- Usability testing
  - Can users accomplish their objectives with the software as designed?

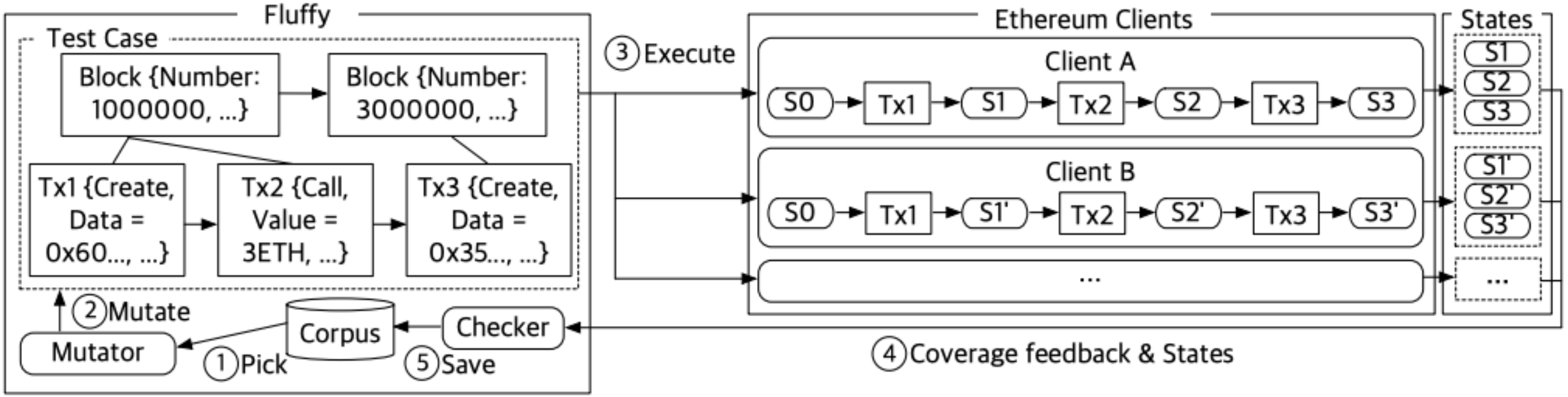
Other Testing Terms You May Hear

- Mutation testing: if introduce deliberate error in **code**, does some test break?
- Fuzz testing: 10,000 monkeys throw random (or guided) **input** at your code
  - Find ~20% MS bugs, crash ~25% Unix utilities
  - *Tests app the way it wasn't meant to be used*
- DU-coverage: is every pair <define x/use x> executed?

Fuzz Testing a PDF Viewer

- Crawl pages to build a corpus
- Use fuzzing tool (or script to)
  1. Grab a file
  2. Mutate that file
  3. Feed it to the program
  4. Record if it crashed (and input that crashed it)

Fluffy (OSDI 2021, SPL)

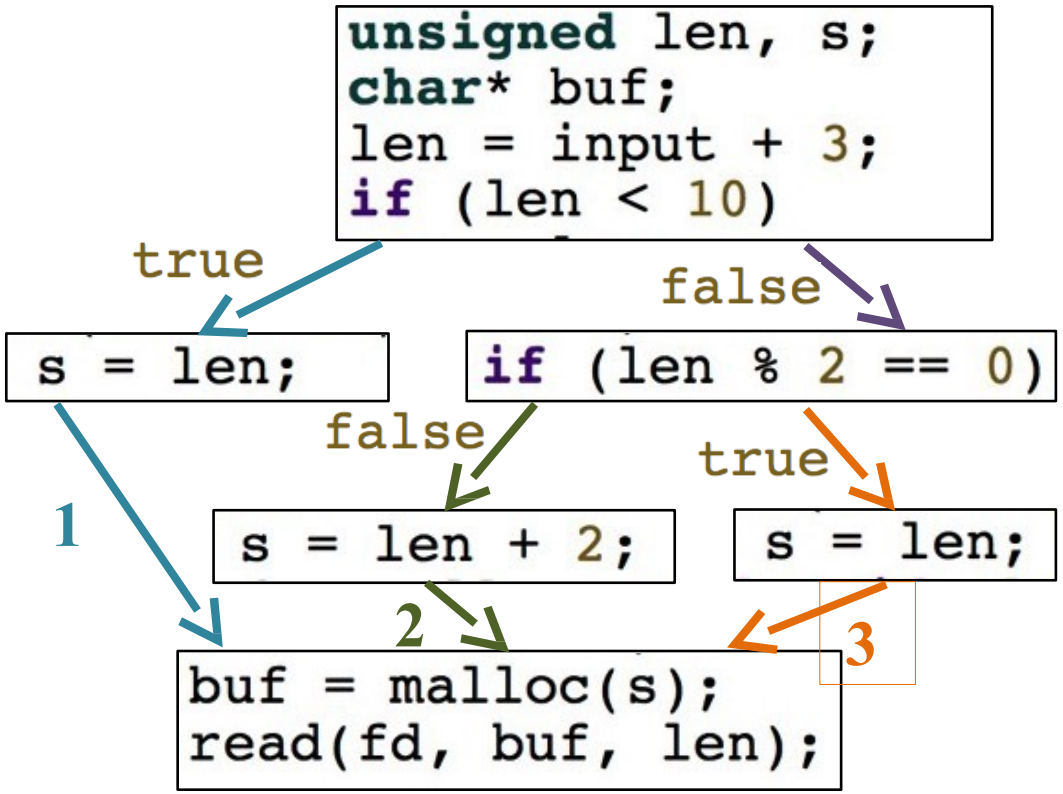


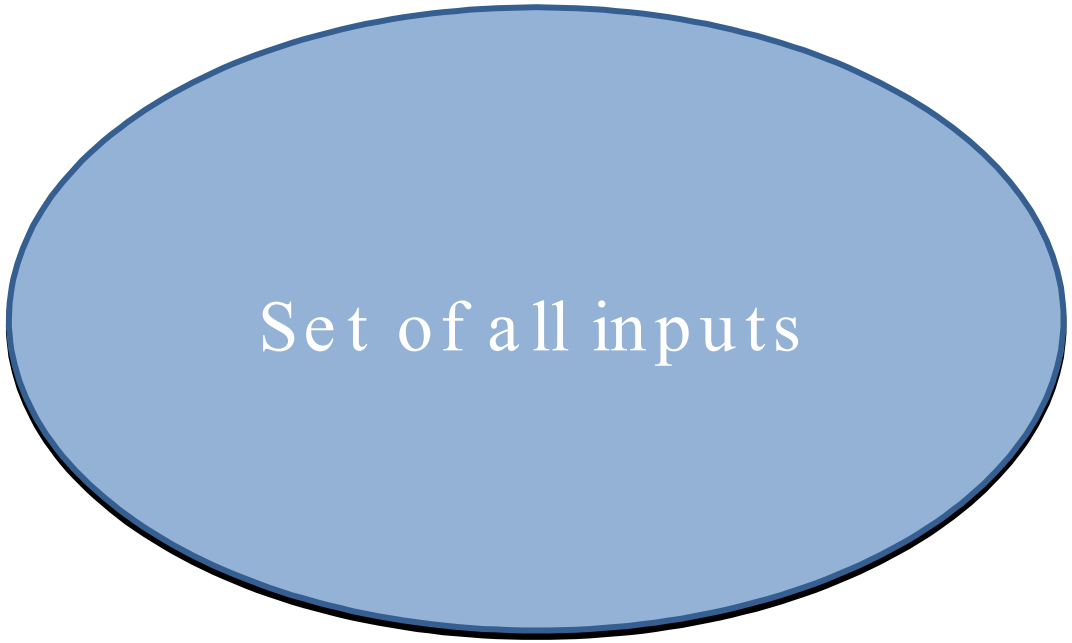
Finding Consensus Bugs in Ethereum via Multi-transaction Differential Fuzzing

Other Testing Terms You May Hear

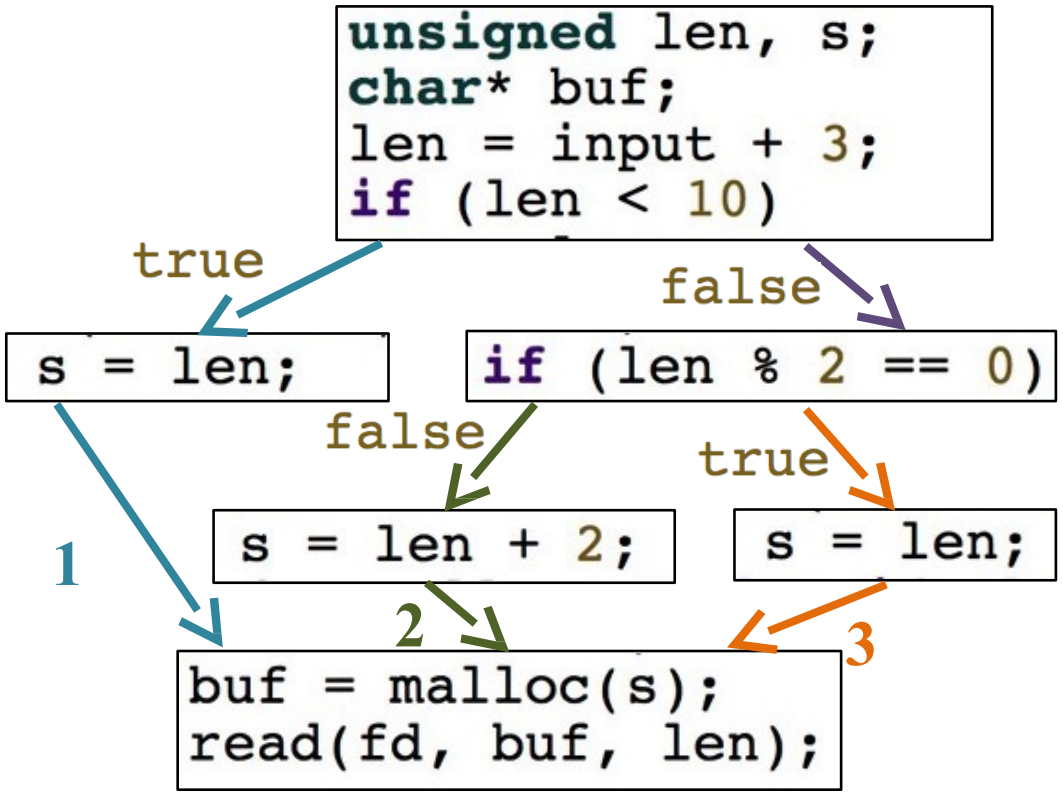
- Symbolic execution
  - Evaluate the program on symbolic input values
  - Use an automated theorem prover to check whether there are corresponding concrete input values that make the program fail
- Concolic testing
  - Hybrid of symbolic and concrete execution

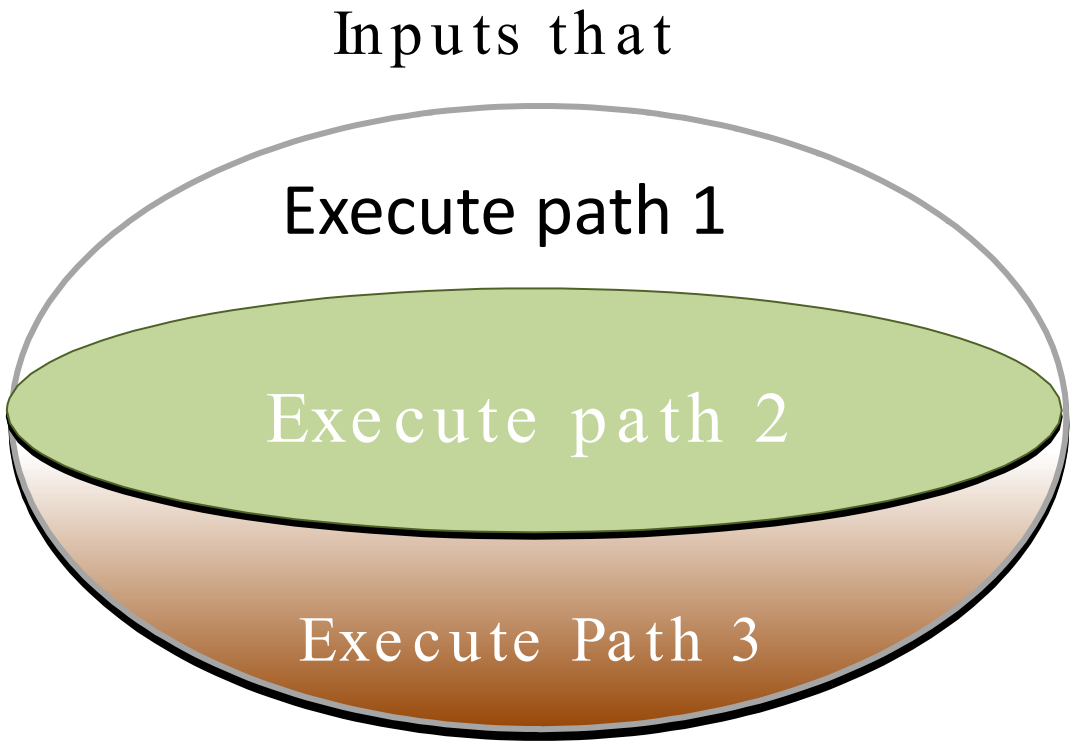
Focus on Sets of Values





Focus on Sets of Values





Goal: find one element of each set

Symbolic analysis provides a way to directly manipulate sets

Symbolic vs. Explicit Representation

Explicit (i.e., concrete) representation

x	-3	-1	1	3
y	0	2	4	6

x	-7	-5	-3	-1	1	3	5	7
y	-4	-2	0	2	4	6	8	10

x	...	-5	-3	-1	1	3	5	...
y	...	-2	0	2	4	6	8	...

Symbolic representation

```
x > -4 && x< 4
&& x % 2 == 1 && y == x + 3
```

```
x > -8 && x < 8
&& x % 2 == 1 && y == x + 3
```

```
x % 2 == 1 && y == x + 3
```

Encodes a set of values
in terms of **properties** of those values

Satisfiability

- A formula is satisfiable if there is a way to assign values to variables and make the formula

(x > 0 && x < 20 && x == y + y) is _____ by (x:10,y:5)
(x > 0 && x < 20 && x == y + y) is _____ by (x:13,y:6)

A formula is satisfied by a satisfying assignment.

A formula is unsatisfiable if every assignment of values to variables makes the formula false.

(x > 0 && x < 20 && x == y + y && x%2 == 1) is _____

Solvers

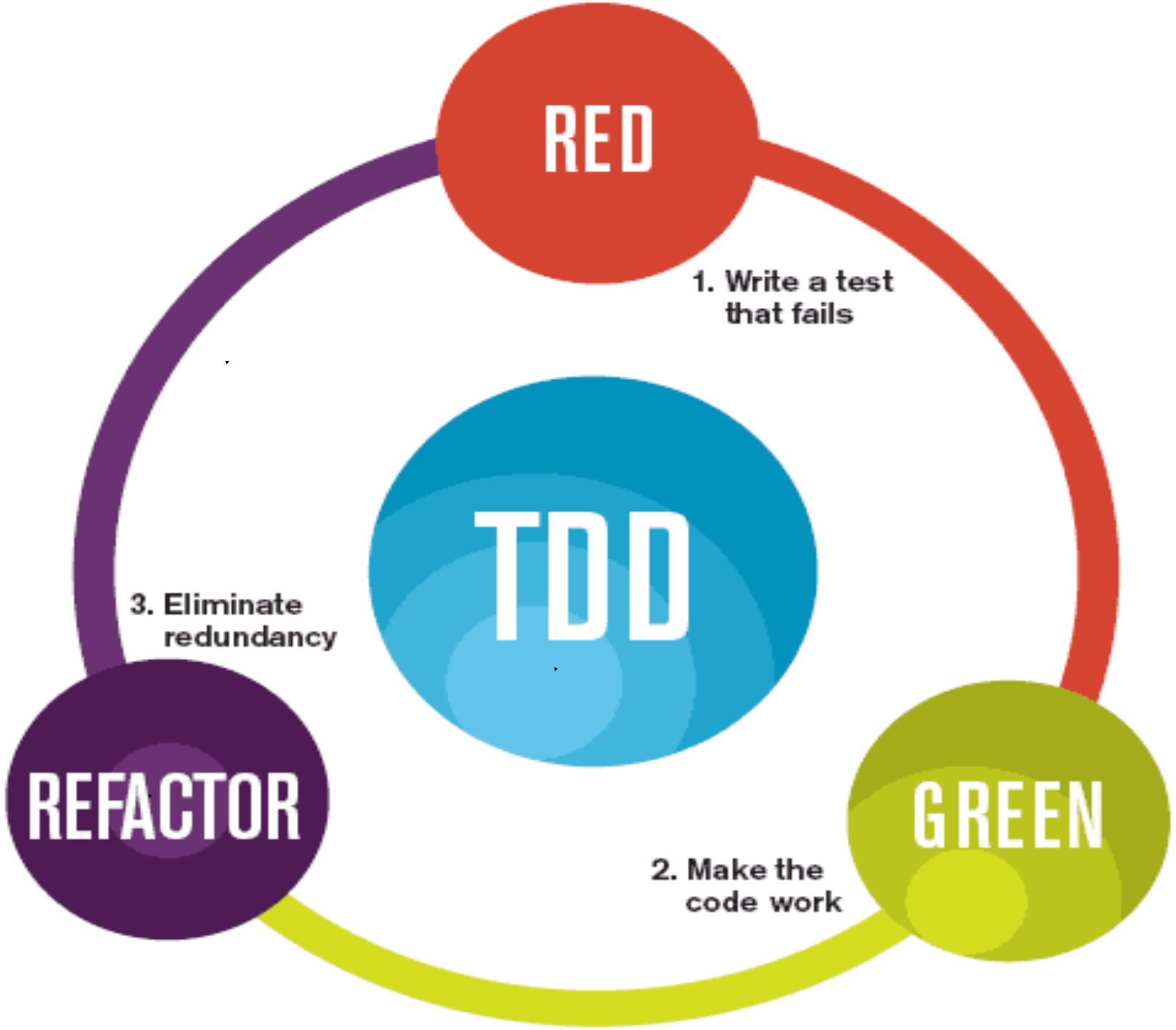
QA in Agile

- Quality assurance is the responsibility of a separate group rather than the result of a good process => Antiquated for SaaS apps
- Developers bear far more responsibility for testing their own code and participating in reviews
- QA engineers have largely shifted to improving the testing tools infrastructure, helping developers make their code more testable, and verifying that customer-reported bugs reproducible

TDD

How To Do TDD


TDD
ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT


RED
1. Write a test
that fails
TDD
GREEN
2. Make the
code work
REFACTOR
3. Eliminate
redundancy

TDD Principles

- You write code in order to make the tests pass
  - You don’t write tests in order to check code you wrote
  - Tests drive the design and implementation
- Invest time early in writing tests => save time later
- It’s not a silver bullet

TDD Reduces Bug Density

Metric description	IBM: Drivers	Microsoft: Windows	Microsoft: MSN	Microsoft: VS
Defect density of comparable team in organization but not using TDD	W	X	Y	Z
Defect density of team using TDD	0.61W	0.38X	0.24Y	0.09Z
Increase in time taken to code the feature because of TDD (%) [Management estimates]	15 – 20%	25-35%	15%	20-25%

Ref. Nagappan et al., "Realizing quality improvement through TDD: results and experiences of four industrial teams", Empirical Software Engineering, 13(3):289–302, Feb 2008, Software Engineering.

When To Use TDD?

- Good candidates
  - User interface behavior (button enabling, button logic, models, etc.)
  - Business logic
  - Pretty much any Java class / method
- Bad candidates
  - User interface appearance (layout, colors, etc.)
  - Client/server interactions (will need to do mock testing)
  - Large code bases, legacy code