

Fuzz testing and input generation

17-313: Foundations of Software Engineering

<https://cmu-313.github.io>

Michael Hilton and **Josh Sunshine**

Fall 2026

Course Announcements

- Grades released: P1b, P2a
- Reminder: Regrade window

Gentle Reminder

- Please close laptops
- Tablets with stylus ok



What are the challenges and limitations of traditional, example-based testing?

Principles of Testing

Principles of Testing #1:

Avoid the *absence of defects* fallacy

- Testing shows the presence of defects
- Testing does not show the absence of defects!
- “no test team can achieve 100% defect detection effectiveness”



Effective Software Testing: A developer's guide. Maurizio Aniche

What about exhaustive testing?

Idea: Try all values!

- **age: int** (2 - 117) years
- **datetime: DateTime** (hh:mm + M/D/Y)
- **rideTime: int** (in minutes, 1 - 2 Hours)
- **is_public_holiday: bool** (2 values)

116 x 1440 (minutes per day) x 1826 (days in the next 5 years)
x 120 (ride time) x 2 ~ **72 Billion test cases**

What about exhaustive testing?

Exhaustive testing is usually impractical – even for trivially small problem

Key problem: choosing test suite

- **Small enough** to finish in a useful amount of time
- **Large enough** to provide a useful amount of validation

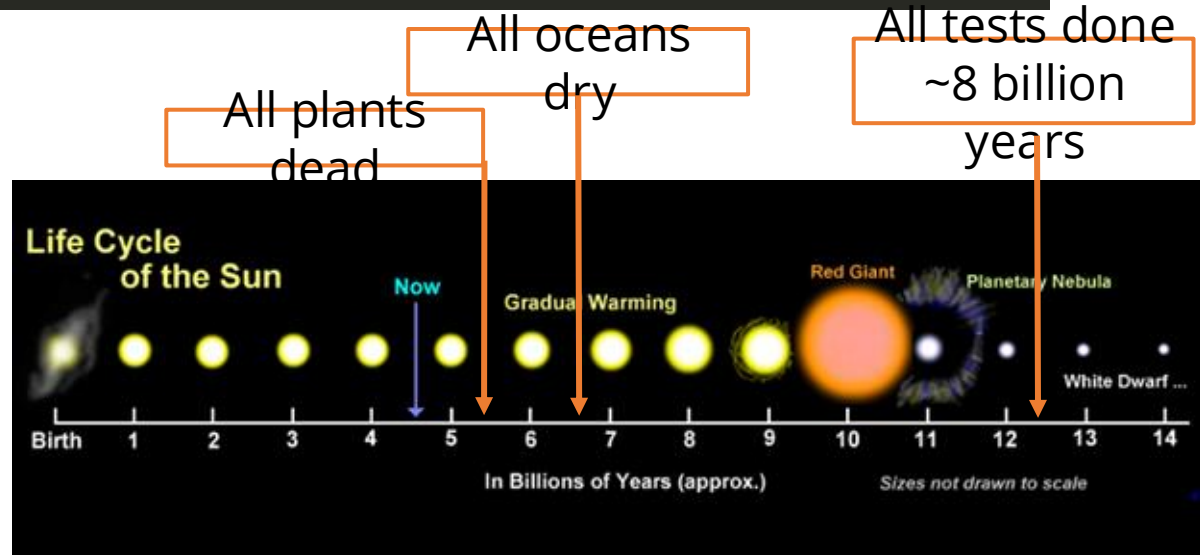
Alternative: **Heuristics**

Principles of Testing #2:

Exhaustive testing is impossible

```
1 def is_valid_email(email: str) -> bool:
2     ...
```

- A simple function, 1 input, string, max. 26 lowercase characters + symbols (@, ., _, -)
- Assume we can use 1 zettaFLOPS: 10^{21} tests per second



Effective Software Testing: A developer's guide. Maurizio Aniche

Principles of Testing #3:

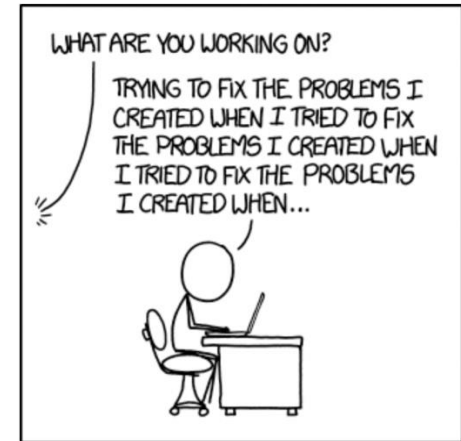
Start testing early

- To let tests guide design
- To get feedback as early as possible
- To find bugs when they are cheapest to fix
- To find bugs when have caused least damage

Principles of Testing #4:

Defects are usually clustered

- “Hot” components requiring frequent change, bad habits, poor developers, tricky logic, business uncertainty, innovative, size, ...
- Use as heuristic to focus test effort

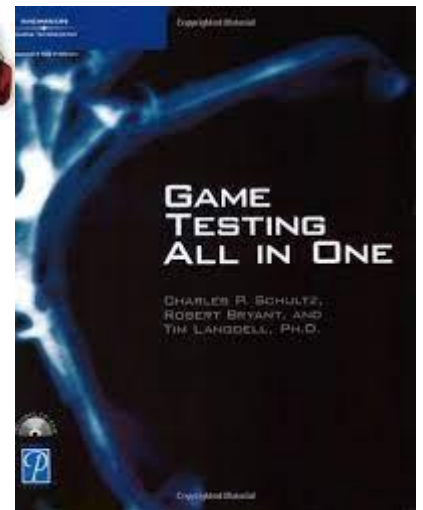
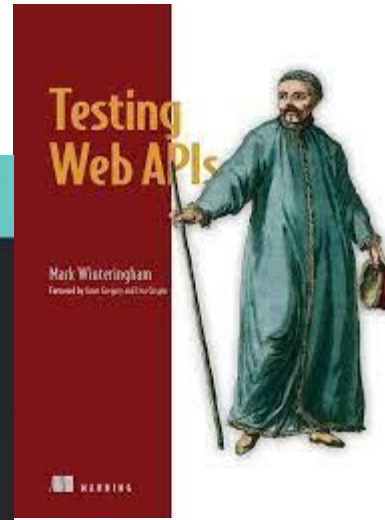
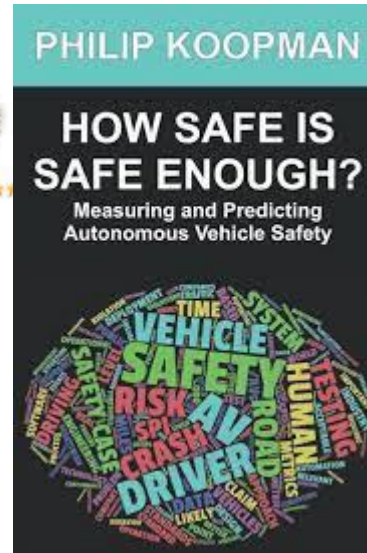
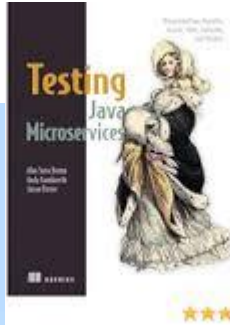


Principles of Testing #5: The pesticide paradox

“Every method you use to prevent or find bugs leaves a residue of subtler bugs against which those methods are ineffectual.”

- Re-running the same test suite again and again on a changing program gives a false sense of security
- Variation in testing

Principles of Testing #6: Testing is context-dependent



Effective Software Testing: A developer's guide. Maurizio Aniche

Principles of Testing #7: Verification is not validation

Verification

- Does the software system meet the requirements specifications?
- Are we building the **software right**?

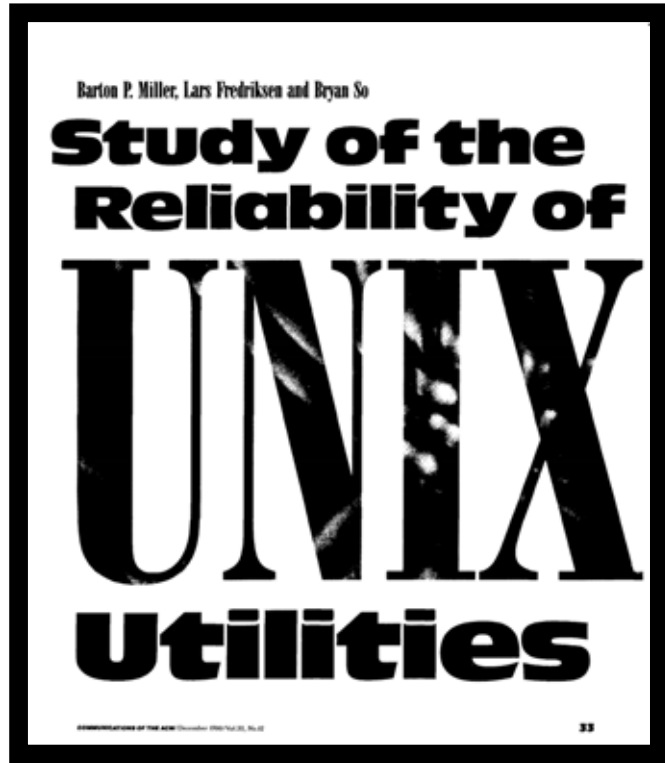
Validation

- Does the software system meet the user's real needs?
- Are we building the **right software**?



Credit: Philip Koopman

Effective Software Testing: A developer's guide. Maurizio Aniche



“

On a dark and stormy night one of the authors was logged on to his workstation on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash.

”

How can we identify these bugs?

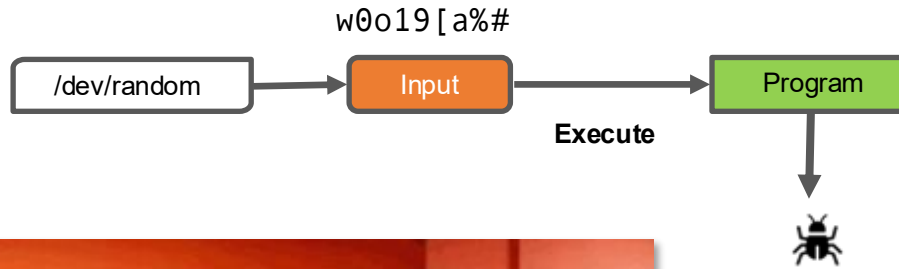
Infinite Monkey Theorem

*“a **monkey** hitting keys **at random** on a typewriter **keyboard** for an **infinite amount of time** will almost surely type any given text, including the complete works of **William Shakespeare**. ”*



https://en.wikipedia.org/wiki/Infinite_monkey_theorem

Fuzz Testing randomly generates inputs and checks for program crashes



A 1990 study found crashes in:
adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi



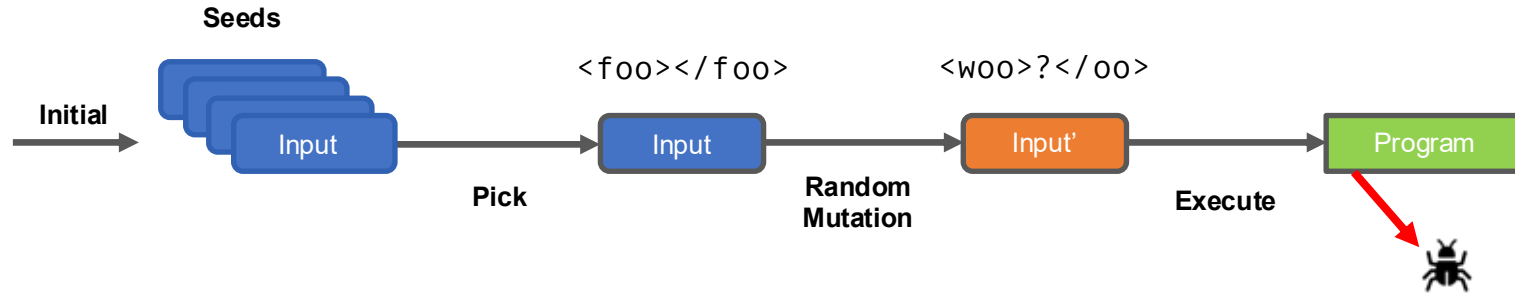
Common Fuzzer-Found Bugs in C/C++

Causes: incorrect arg validation, incorrect type casting, executing untrusted code, etc.

Effects: buffer-overflows, memory leak, division-by-zero, use-after-free, assertion violation, etc. (“crash”)

Impact: security, reliability, performance, correctness

Mutation-Based Fuzzing (e.g., Radamsa)



<https://gitlab.com/akihe/radamsa>

Mutation Heuristics

- **Binary Input**

- bit flips, byte flips
- modify, insert, delete random byte chunks
- set randomly chosen byte chunks to interesting values e.g. INT_MAX, INT_MIN, 0, 1, -1, ...

- **Text Input**

- insert random symbols relevant to format (e.g. "<" and ">" for xml)
- insert keywords from a dictionary (e.g. "<project>" for Maven POM.xml)

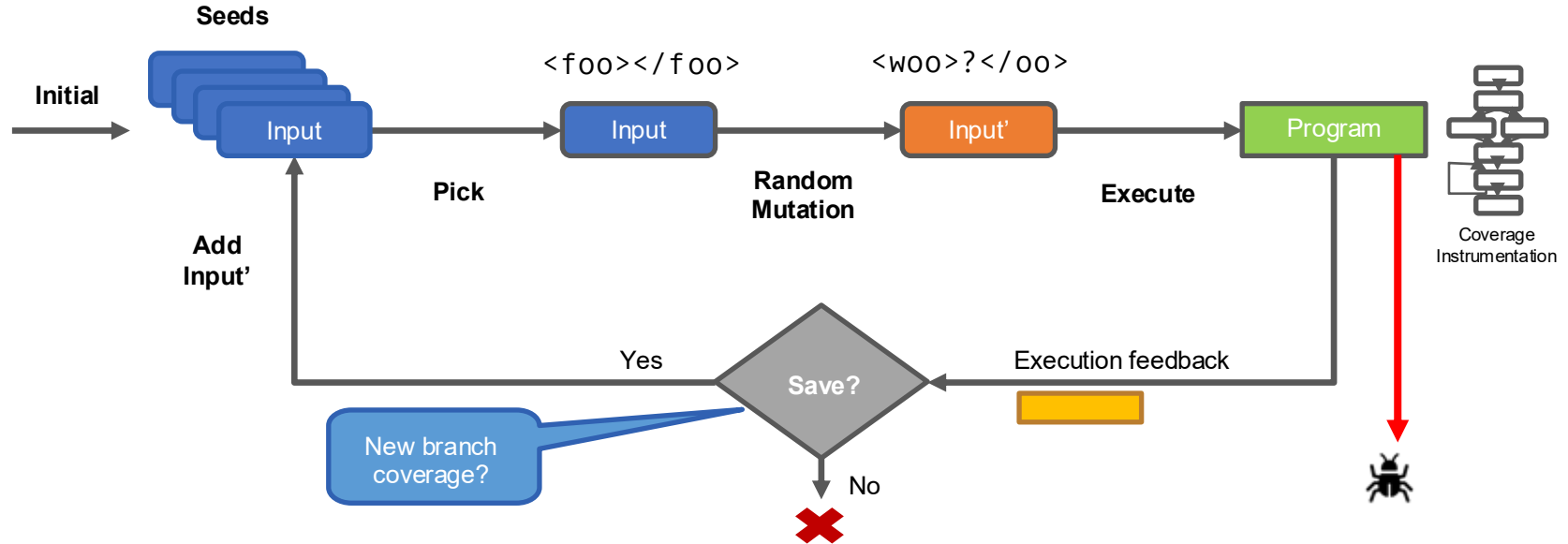
- **GUI Input**

- change click types and targets
- change text
- click different buttons



```
<html><head><title>Hello</title></head><body>World<br/></body></html>
```

Coverage-Guided Fuzzing (e.g., AFL)



<https://lcamtuf.coredump.cx/afl/>

Finding Security Bugs =



Meta Bug Bounty

If you believe you have found a security vulnerability on Meta (or another member of the Meta family of companies), we encourage you to let us know right away.

[Submit a report](#)

Total rewards for 2025

\$4,353,212

Total rewards to date

\$25,497,082

\$300K*

Mobile RCE
WhatsApp
Private
Processing

\$130K*

Account
Takeover

\$30K*

Quest
Persistent full
secure boot
bypass

\$20K*

2FA Bypass

\$10K*

Contact point
deanonymization

\$5K*

Page admin
disclosure

\$500*

Minimum
bounty

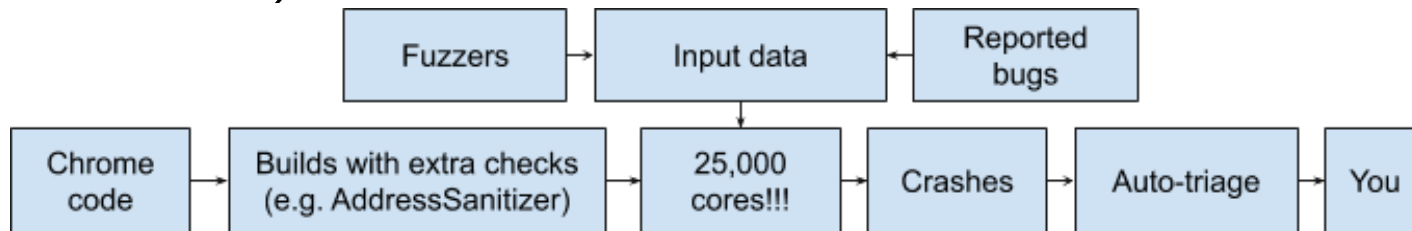
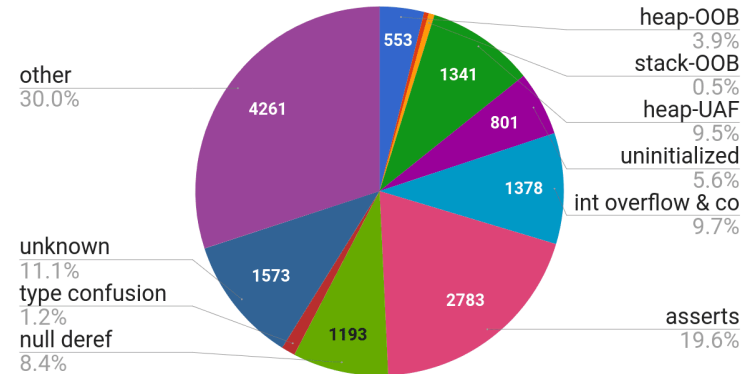
Fuzzing in Practice



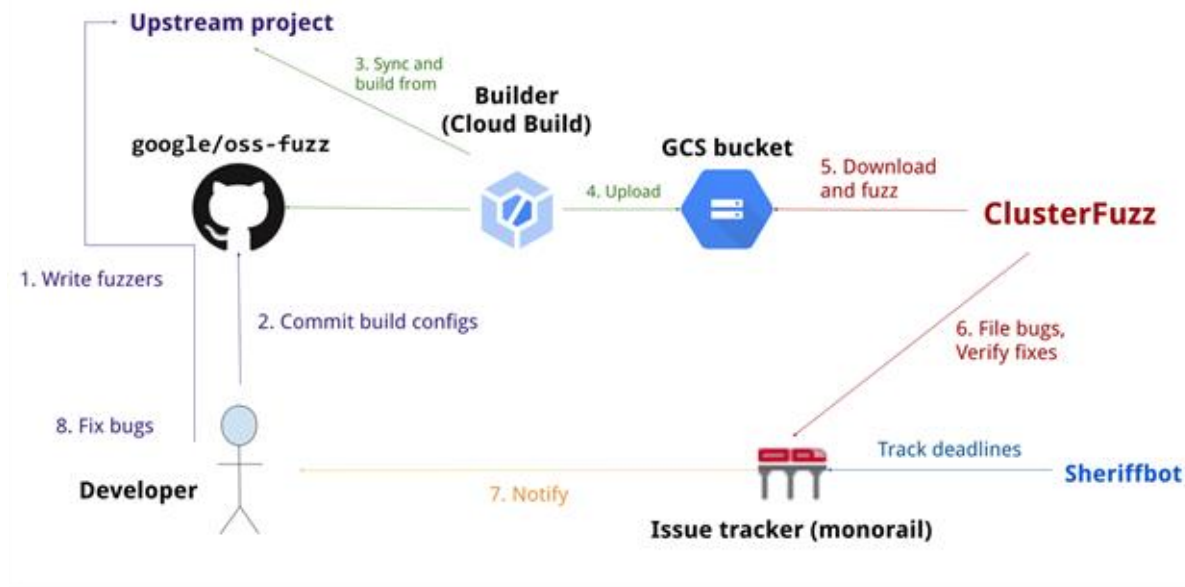
- After the OpenSSL Heartbleed vulnerability discovered in 2016, Google launched **OSS-Fuzz**
- Free service for open-source projects
 - “The project must have a significant user base and/or be critical to the global IT infrastructure.”
- OSS-Fuzz privately alerts developers to vulnerabilities

Fuzzing in Practice

- Google uses **ClusterFuzz** to fuzz all of their products
 - supports multiple fuzzing strategies
- *As of February 2026, ClusterFuzz has found 30,000+ bugs in Google code (e.g., Chromium)*



OSS-Fuzz: Free Fuzzing for Open-Source Software



As of May 2025, OSS-Fuzz has helped identify and fix over 13,000 vulnerabilities and 50,000 bugs across 1,000 projects. (e.g., nodejs, django, openvpn, openssl)

Fuzz-testing in the AI era

Rediscovering an old technique for new challenges



< Engineering Stack



Testing

Security

Blog

By **Richard Gall**

Published: July 07, 2025

Fuzz testing is a software testing technique that's been around for some time. But, despite being nearly forty years old [↗](#), the technique hasn't been widely adopted by software development teams. While it's commonly used in specialist fields like penetration testing, it's often viewed as somewhat marginal by the industry mainstream.

Activity: Wordle Input Generator

1. How can you generate random inputs for ``match_guess``?
2. What inputs are more likely to find bugs in ``match_guess``?
3. Describe a strategy for generating these “likely bug” inputs.

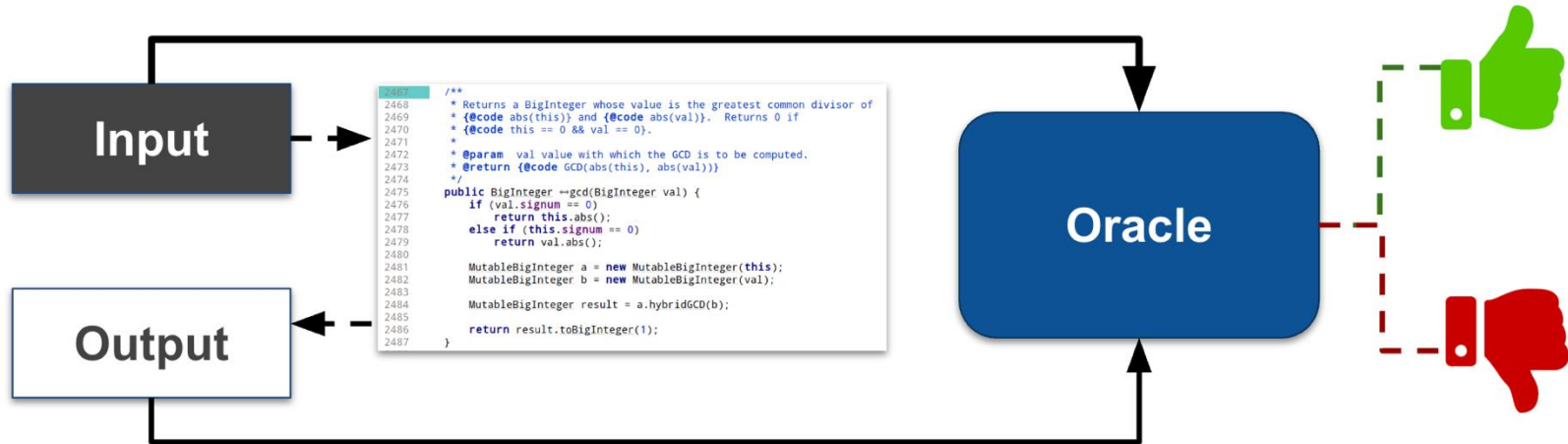
G	U	E	S	S
W	H	I	C	H
C	R	A	Z	E
J	O	I	N	S
T	I	M	E	S
G	A	M	E	S

What are the limitations of random input generation?

Oracles

Testing is Only as Good as your Oracle

- An oracle decides if behavior is correct for a given input
 - **strong oracles** catch bugs that **weak oracles** miss
 - designing strong oracles is difficult and often the bottleneck



Oracle: Assertions in Example-Based Tests

- **This is the most common type of oracle in traditional tests**
- These assertions are often hardcoded to a specific test input
 - tedious to write for complex outputs (e.g., documents, actions)
 - can be very brittle (e.g., formatting changes lead to test failures)
 - non-determinism and environment coupling lead to flaky tests

```
it('should redirect to login if user is not logged in', async () => {  
  const { response, body } = await request.get(`${nconf.get('url')}/me/bookmarks`);  
  assert.equal(response.statusCode, 200);  
  assert(body.includes('Login to your account'), body.slice(0, 500));  
});
```

Oracle: The Program Shouldn't Crash!

- This is the oracle used by most fuzzing approaches
- This oracle is a **generic property** that is not tied to any test inputs
 - that allows us to automatically generate and test any input
 - but the oracle is **weak** (i.e., not crashing does not imply correct)
- We can make the oracle slightly stronger by using **sanitizers**
 - detects illegal program states that might not cause an immediate crash
 - instruments the program at compile time (e.g., `-fsanitize=address`)
 - finds more safety issues but slows down execution / fuzzing
 - doesn't reveal logic bugs



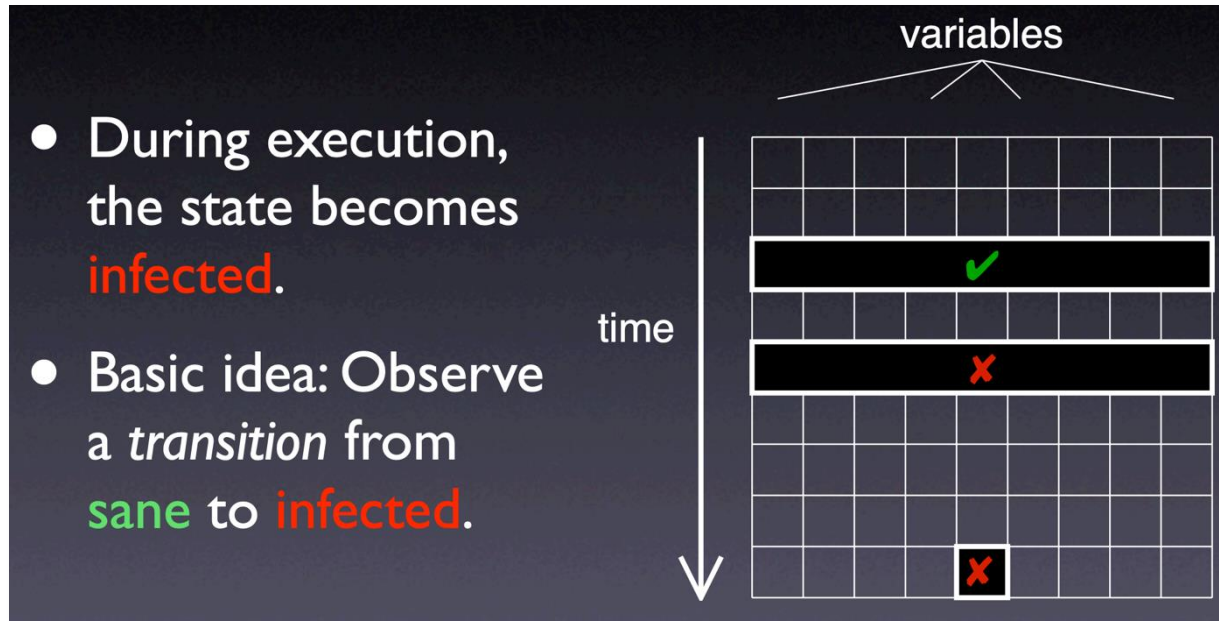
Oracle: Assertions in Source Code

- Assertions are **executable specifications**
 - document intended behavior (pre/postconditions, invariants)
- This oracle is generic and **not tied to any test inputs**
 - if we add assertions, we can use fuzzing to find some logic bugs!

```
function toUSD(amountCents: number): string {  
    assert(Number.isInteger(amountCents), 'amount must be integer cents');  
    assert(amountCents >= 0, 'amount must be non-negative');  
    const dollars = (amountCents / 100).toFixed(2);  
    return `$$${dollars}`;  
}
```

Assertions catch infections earlier

- Finds more bugs (e.g., during fuzzing) and helps to localize them



Assertions should always be true unless you have a bug in your code

- Assertions state invariants: conditions that must always hold if the program is correct (e.g., impossible states, internal consistency).
 - **Never rely on asserts for control flow or user-visible behavior**
 - Make sure that your assertions **don't contain side effects**
- Use exceptions and returns for errors that can reasonably happen and should be handled (e.g., invalid inputs, failed API calls).

Assertions in the Wild: Apache Cassandra

- Used to enforce an **invariant** that must hold throughout sorting

```
218      *
219      * @param a the array in which a range is to be sorted
220      * @param lo the index of the first element in the range to be sorted
221      * @param hi the index after the last element in the range to be sorted
222      * @param start the index of the first element in the range that is
223      *           not already known to be sorted (@code lo <= start <= hi)
224      * @param c comparator to used for the sort
225      */
226      @SuppressWarnings("fallthrough")
227      private static void binarySort(long[] a, int lo, int hi, int start,
228                                   LongComparator c) {
229          if (DEBUG) assert lo <= start && start <= hi;
230          if (start == lo)
231              start++;
232          for ( ; start < hi; start++) {
233              long pivot = a[start];
234              // Set left (and right) to the index where a[start] (pivot) belongs
235              int left = lo;
236              int right = start;
237              if (DEBUG) assert left <= right;
```

Assertions in the Wild: SQLite & LLVM

- Used to enforce a **precondition** and find bugs at call sites

```
/*
** Insert a new entry into the cache.  If the cache is full, expel
** the least recently used entry.  Return SQLITE_OK on success or a
** result code otherwise.
**
** Cache entries are stored in age order, oldest first.
*/
static int jsonCacheInsert(
    sqlite3_context *ctx, /* The SQL statement context holding the cache */
    JsonParse *pParse      /* The parse object to be added to the cache */
){
    JsonCache *p;

    assert( pParse->zJson!=0 );
    assert( pParse->bJsonIsRCStr );
    assert( pParse->delta==0 );
    p = sqlite3_get_auxdata(ctx, JSON_CACHE_ID);
    if( p==0 ){
        sqlite3 *db = sqlite3_context_db_handle(ctx);
```

lldb / include / lldb / Interpreter /  OptionValueUInt64.h 

Code

Blame

99 lines · 3.08 KB

```
69         m_current_value = value;
70         return true;
71     }
72     return false;
73 }
74
75 ▼ bool SetDefaultValue(uint64_t value) {
76 ▼     assert(value >= m_min_value && value <= m_max_value &&
77         "disallowed default value");
78     m_default_value = value;
79     return true;
80 }
```

Assertions in the Wild: Firefox

- Used to enforce a **postcondition** that makes sure audio packet is the correct size after processing

```
namespace mozilla {  
void AudioInputProcessing::Process(AudioProcessingTrack* aTrack,  
  
    // Postconditions of the audio-processing logic.  
    MOZ_ASSERT(static_cast<uint32_t>(mSegment.GetDuration()) +  
                mPacketizerInput->FramesAvailable() ==  
                mPacketizerInput->mPacketSize);  
    MOZ_ASSERT(mSegment.GetDuration() >= 1);  
    MOZ_ASSERT(mSegment.GetDuration() <= mPacketizerInput->mPacketSize);  
}
```

Activity: Wordle Oracle

1. What precondition assertions could you add to ``mark_guess``?
2. What postcondition assertions could you add?

G	U	E	S	S
W	H	I	C	H
C	R	A	Z	E
J	O	I	N	S
T	I	M	E	S
G	A	M	E	S