Assurance and Sustainability

Ross Anderson

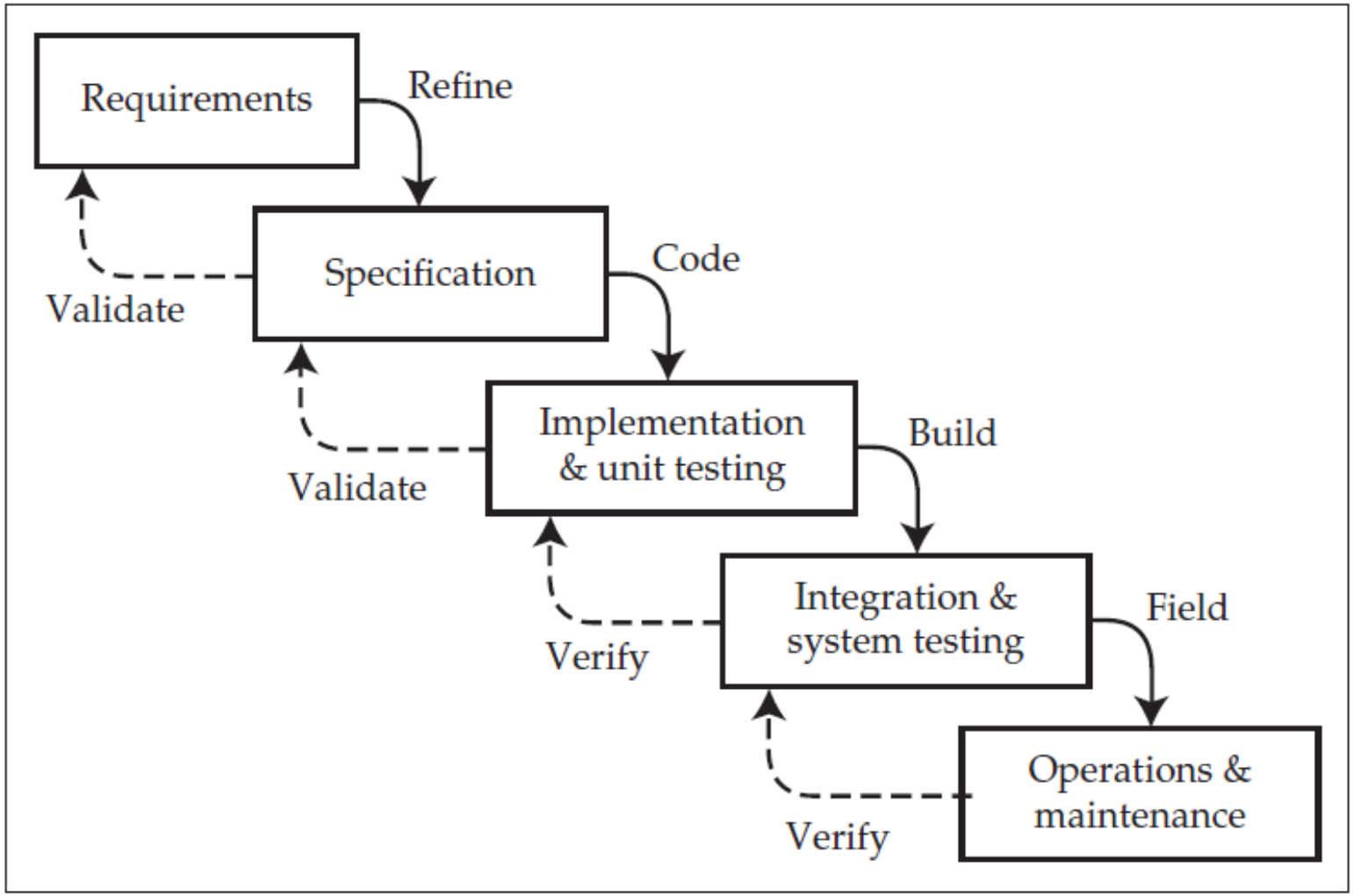
Definitions

- Assurance: whether a system will work, and how you're sure of this.
- Compliance: how you can satisfy other people of this.
- Sustainability: how long will it work for?
- Secure systems need Incentives, Policy, Mechanisms and Assurance. Usability cuts through all four!

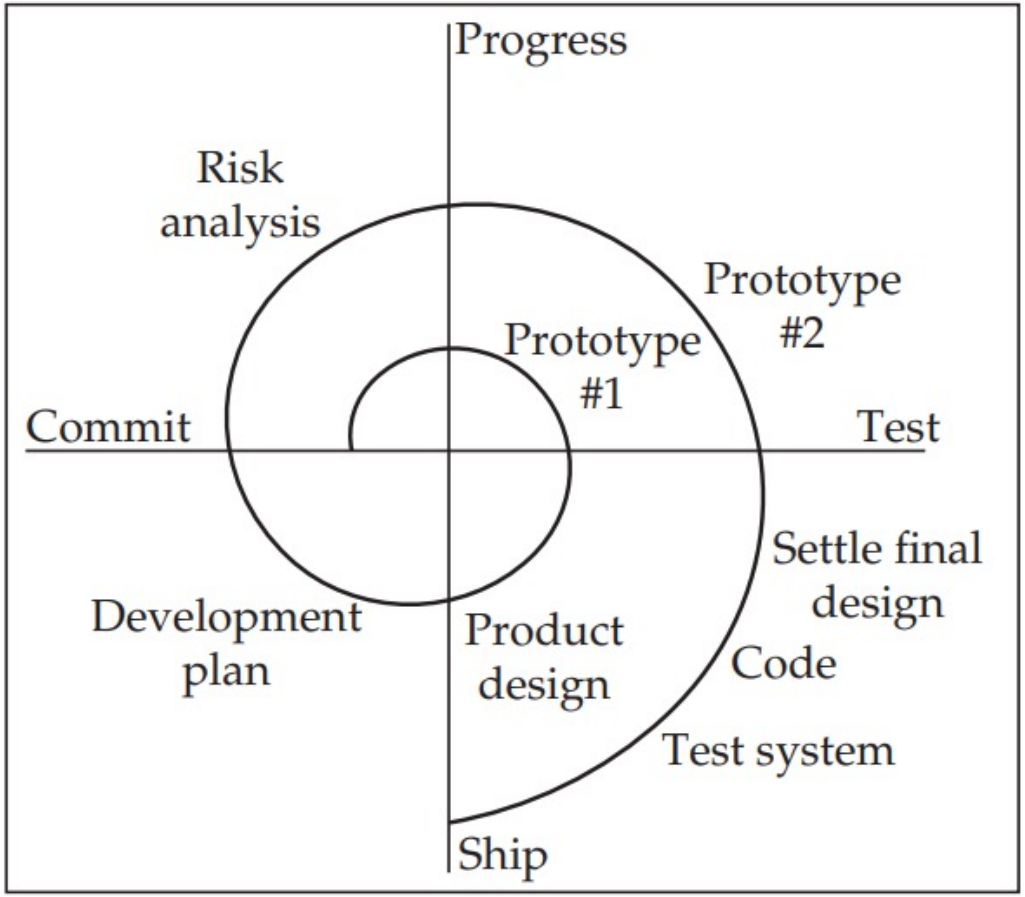
DevSecOps

- DevOps: blur development and execution
- DevSecOps: Add security in to the entire lifecycle too!
- DevSecOps involves a “shift left”
- Solve your hardest problems first: Spiral and Agile.

Waterfall

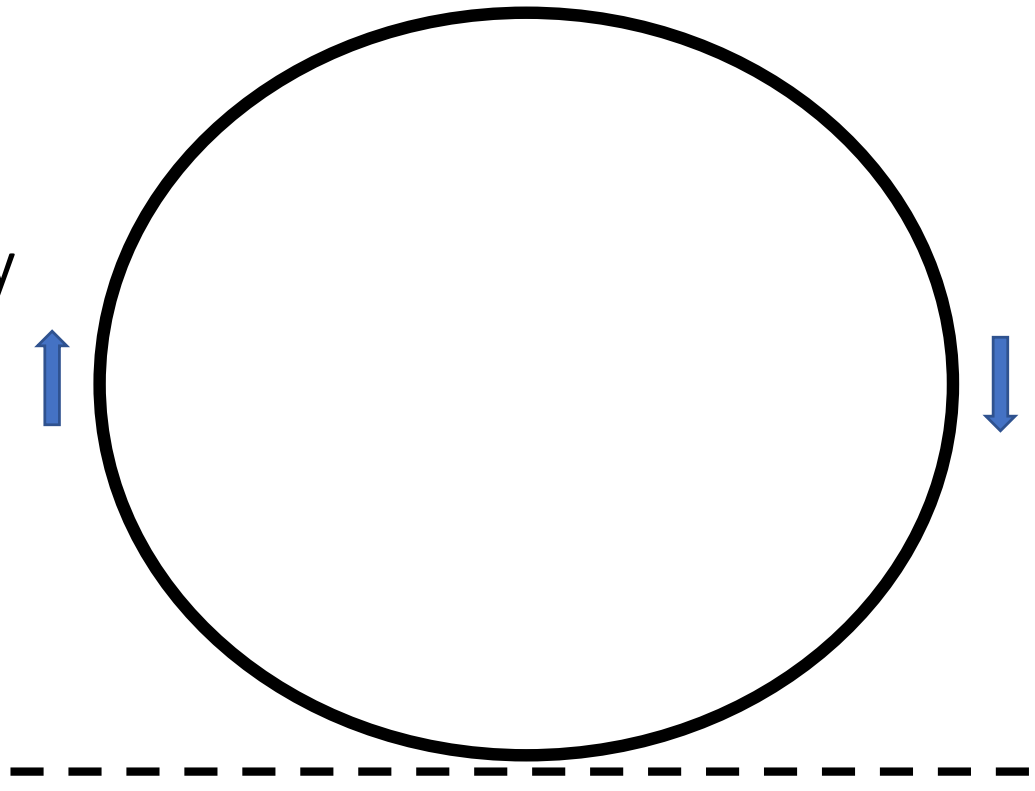

Requirements
Refine
Specification
Code
Validate
Implementation
& unit testing
Build
Validate
Integration &
system testing
Field
Verify
Operations &
maintenance
Verify

Spiral


Progress
Risk analysis
Prototype #2
Prototype #1
Commit
Test
Settle final design
Development plan
Product design
Code
Test system
Ship

Agile

What's my hardest / riskiest problem?



2 weeks

DevSecOps (2)

- Technical debt: shortcuts have to be repaid later!
- Run your DevOps environment “debt-free”.
- Automate configuration as well as build!
- Use proxy tools where possible
- Google: set a realistic reliability target of e.g. 99.9% and use the rest for failure recovery, upgrades and experiments.

Design for Testability: Unit Tests

- JUnit (Java), GoogleTest (C++), xUnit (for all X)
- Test-driven development (TDD): write the tests first!
- Refactor code for testability: abstractions to avoid “flaky tests”.

```
@Test(timeout=100)
public void testFib() {
    assertEquals(55,fibonacci(10));
    assertEquals(1,fibonacci(1));
}
```

Design for Testability: Integration Tests

Jenkinsfile (Declarative Pipeline)

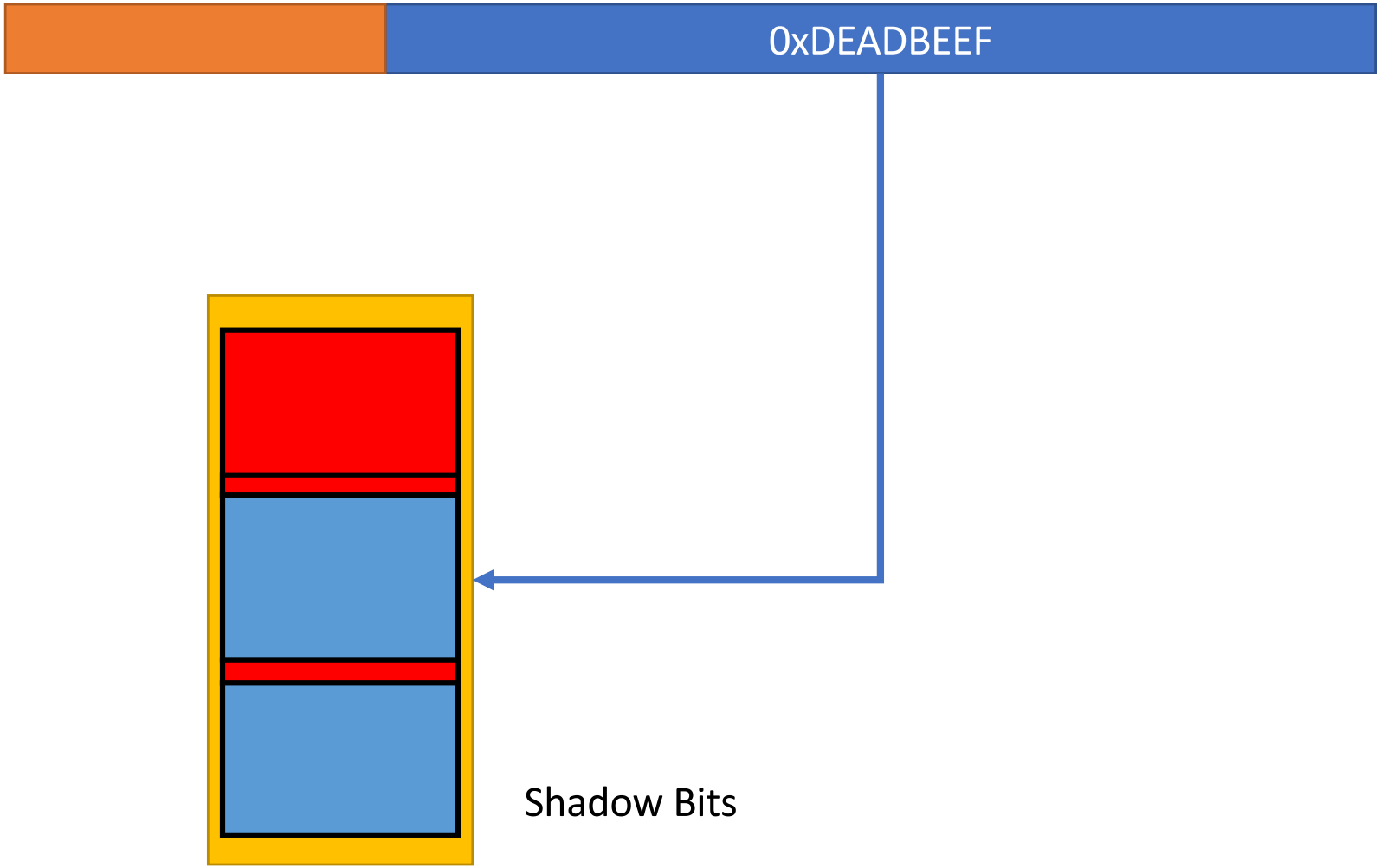
```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

Design for Testability: Integration Tests (2)

- Integration tests should use the real interfaces, not abstractions!
- Be careful around the privilege of your tests!
- Don’t use real data (too flaky and secret), and don’t leave secrets in the code.

Dynamic Analysers: AddressSanitizer



Dynamic Analysers (2): Sanitizers and Mitigators

- Sanitisers: AddrSan, UBSan, LSan, MSan, TSan, Valgrind, Helgrind.
- Mitigators: Scudo Hardened Allocator, Clang CFI, MarkUs/MineSweeper
- Mitigators: no false positives.
- Sanitisers: some false positives allowed. Hide with e.g. __attribute__((no_sanitize("undefined")))

Dynamic Analysers (3): Fuzzing

- Combine your sanitizers with fuzzing.
- Dumb fuzzing: RNGs.
- Smart fuzzing: domain-specific dictionaries.
- FuzzedDataProvider in LLVM: format conversion for random input
- LibFuzzer: generate a corpus based on code coverage!
- Combine with chaos engineering: inject faults into both tests and production!

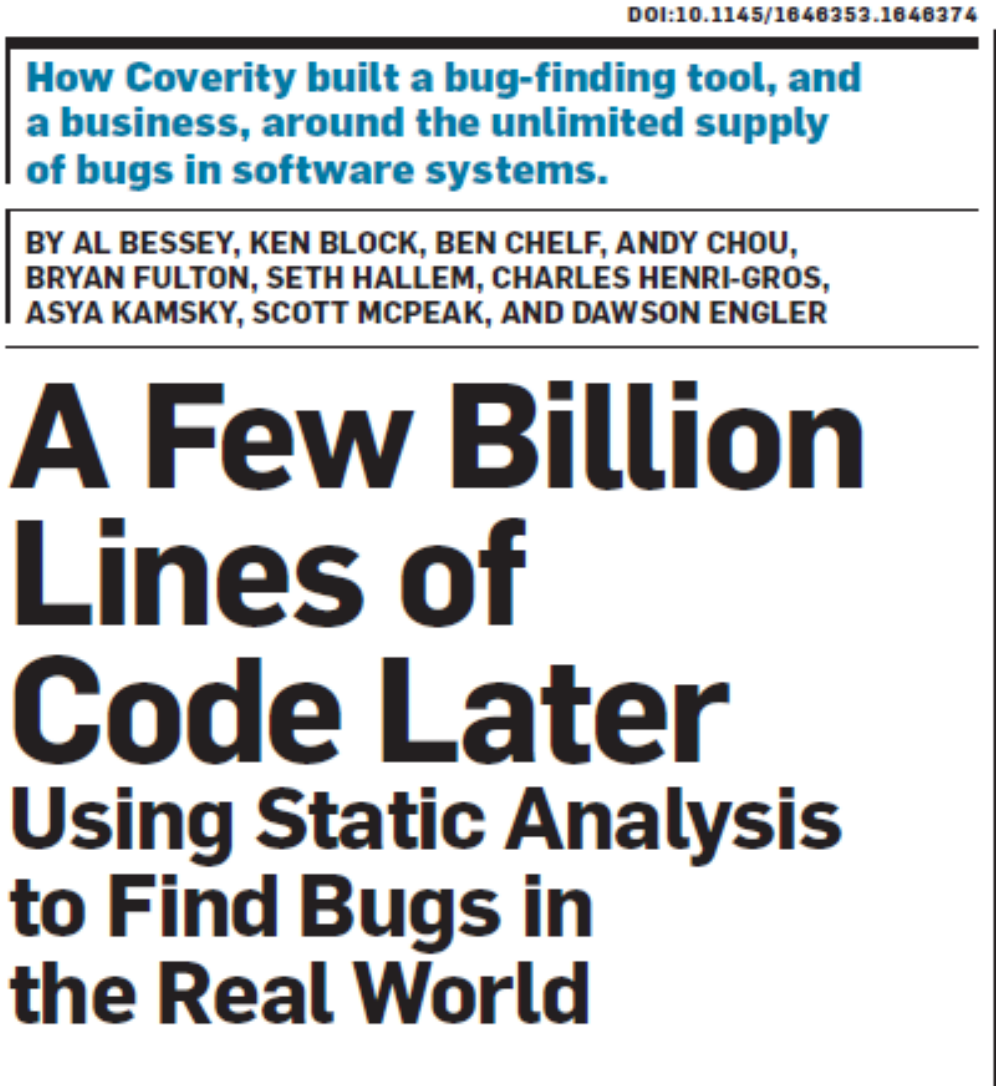
Static Analysers: Linters

- Error Prone for Java / Clang-Tidy for C(++).

```
#define BUFLEN 42
char buf[BUFLEN];
memset(buf, 0, sizeof(BUFLEN));
// sizeof(42) ==> sizeof(int)
```

https://clang.llvm.org/extra/clang-tidy/checks/bugprone-sizeof-expression.html

Static Analysers: Bug Finders



DOI:10.1145/1646353.1646374

How Coverity built a bug-finding tool, and a business, around the unlimited supply of bugs in software systems.

BY AL BESSEY, KEN BLOCK, BEN CHELF, ANDY CHOU, BRYAN FULTON, SETH HALLEM, CHARLES HENRI-GROS, ASYA KAMSKY, SCOTT MCPEAK, AND DAWSON ENGLER

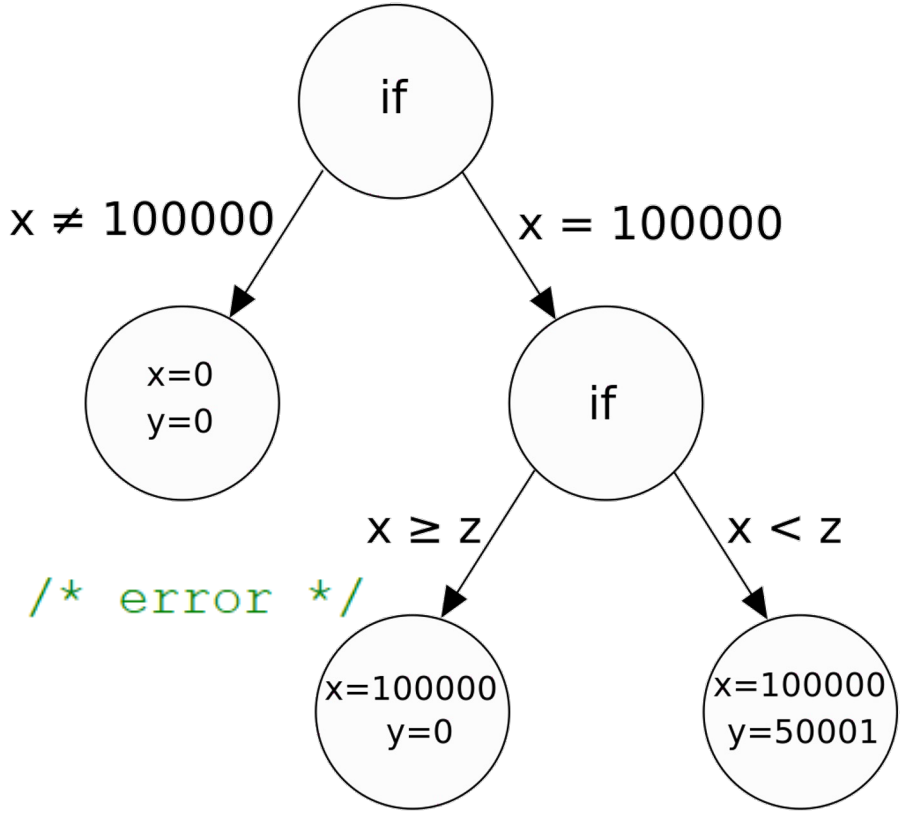
A Few Billion Lines of Code Later

Using Static Analysis to Find Bugs in the Real World

Static Analysers: Concolic Tests

- E.g. Klee

```
void f(int x, int y) {
    int z = 2*y;
    if (x == 100000) {
        if (x < z) {
            assert(0); /* error */
        }
    }
}
```



Static Analysers: Abstract Interpretation

```
int* a = malloc(4000*sizeof(int)); //@a=[0,4000]
int sum = 0;

for(int x=0; x< 10000; x++) { // @x=[0,9999]
    int y = x+x;               // @y=[0,19998]
    int z= (y+4)&8191;         // @z=[0,8191]
    sum+= a[z];               //out of bounds!
}
```

Static Analysers: Abstract Interpretation

```
int* a = malloc(4000*sizeof(int));
int sum = 0;

for(int x=0; x< 10000; x++) { // @x={odd,even}
    int y = x+x;              // @y={even}
    sum+= a[y&1?10000:0];           //safe!
}
```

Static Analysers: Abstract Interpretation

```
int* a = malloc(4000*sizeof(int)); //@a=[0,4000]
int sum = 0;

for(int x=0; x< 10000; x++) { // @x=[0,9999]
    int y = x+x;              // @y=[0,19998]
    sum+= a[y&1?10000:0];           //out of bounds???

}
```

Static Analysers: Formal Methods

$$\frac{\{B \wedge P\}S\{Q\} \quad , \quad \{\neg B \wedge P\}T\{Q\}}{\{P\}\texttt{if } B \texttt{ then } S \texttt{ else } T \texttt{ endif}\{Q\}}$$

Static Analysers: Type Systems

- PHP -> Hack, JavaScript -> TypeScript.

```
let x = "alphabet";
x = 45;
```

Static Analysers: Prepared Statements

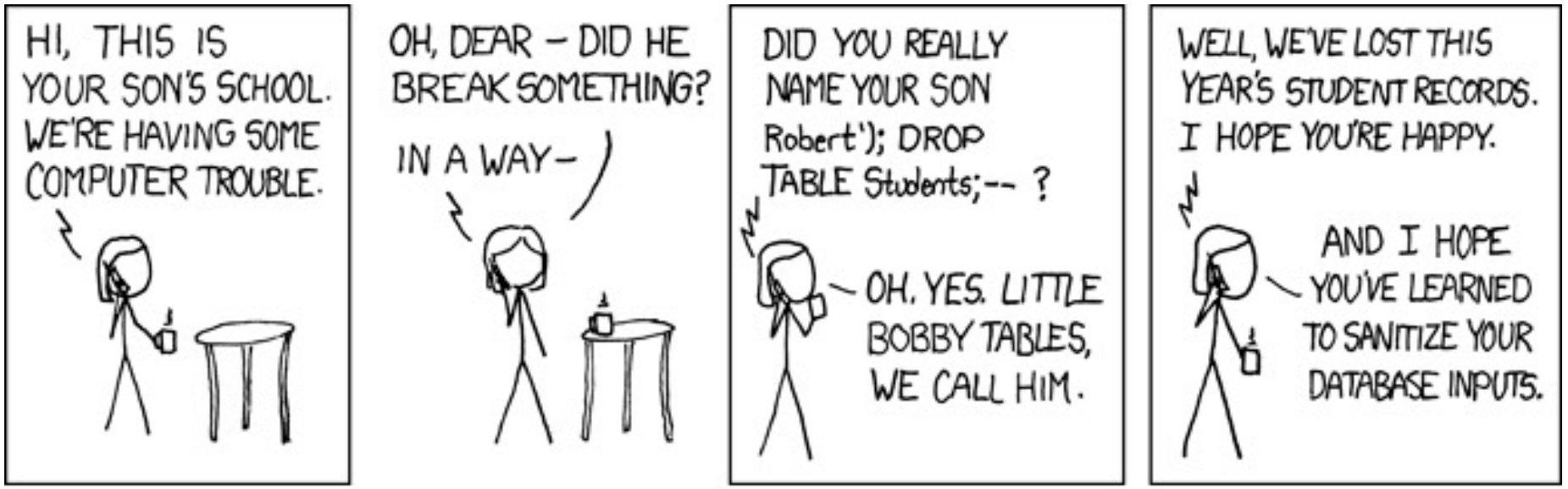
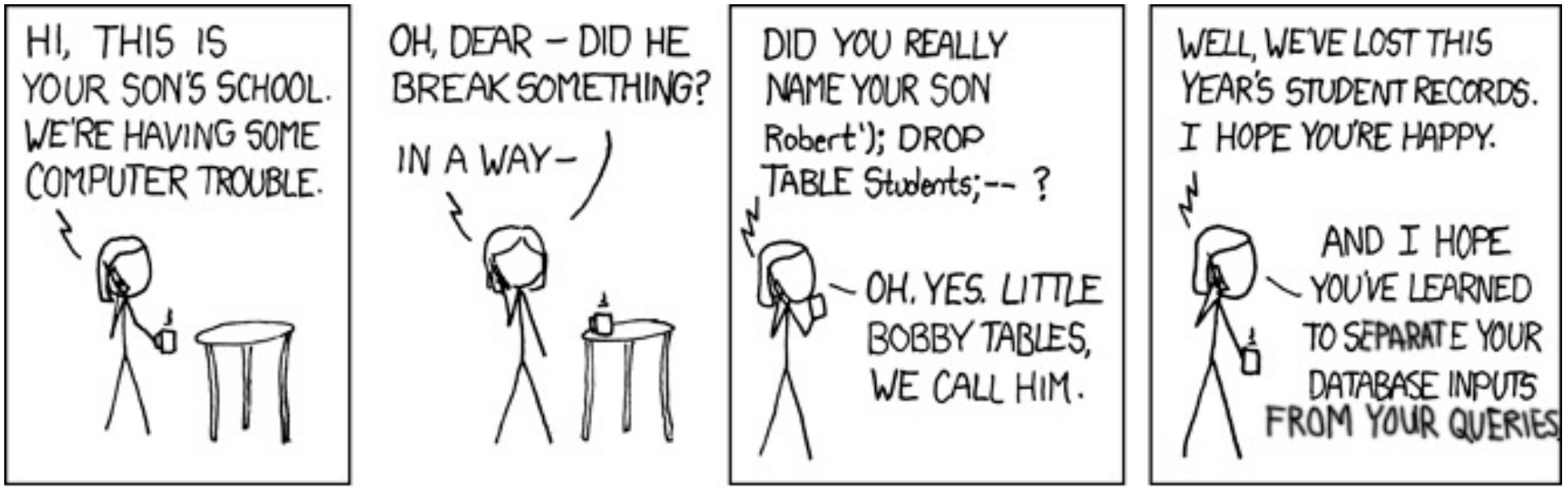


Image from XKCD

https://xkcd.com/327/

Static Analysers: Prepared Statements



Don’t just Sanitize – use e.g. SafeSQL to make the input command static!

Getting your Disclosure Policy Right

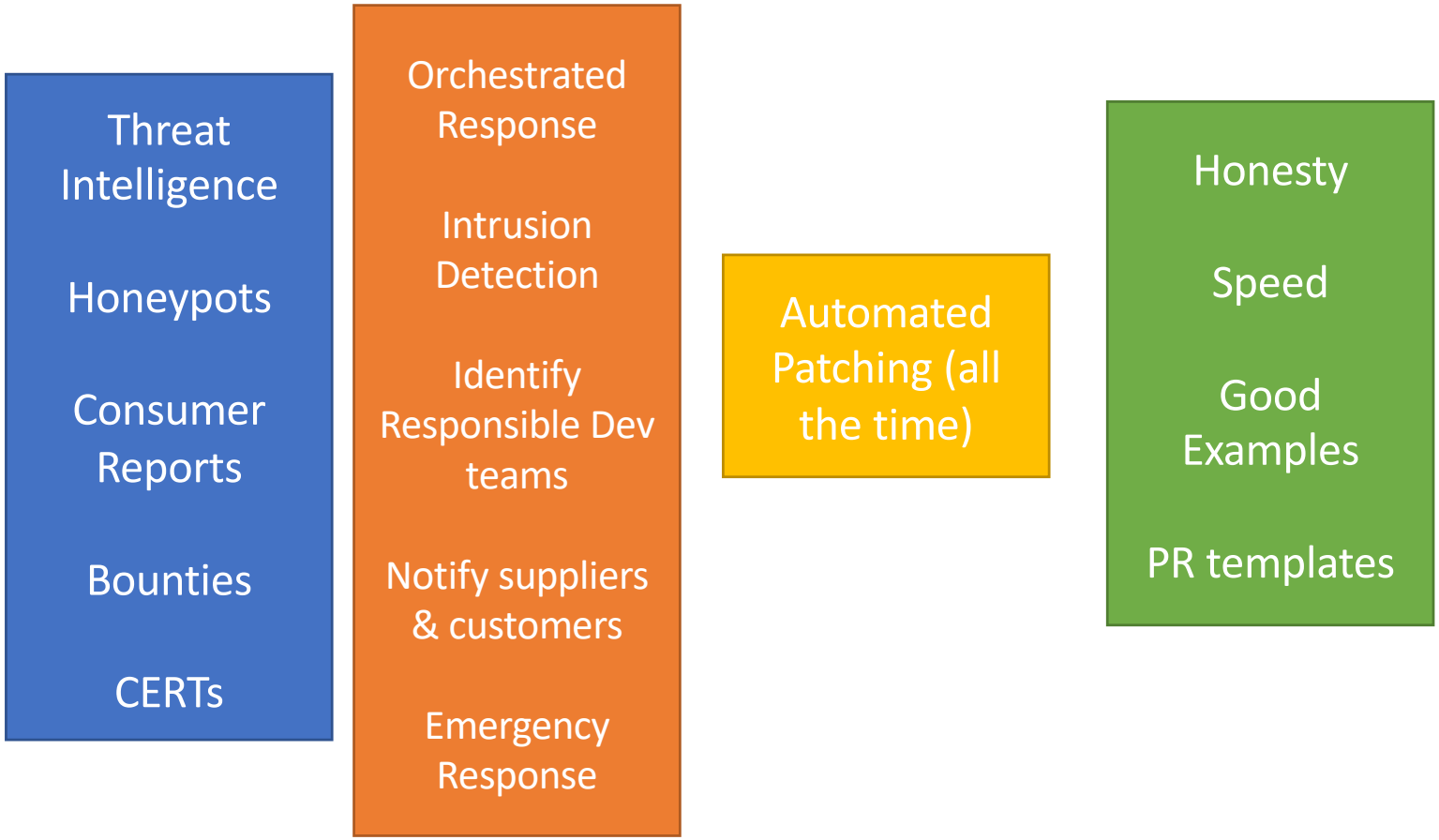
- How do you get people to report bugs to you before they disclose publicly?
- How do you avoid disclosures selling your bugs to someone else?
- How do you get your CEO to not deny/deflect?

Incentivising Finding Bugs

- External Researchers: Bug bounties and "vulnerability pricing"
- Red and Blue teams within your company.
- Chaos Engineering: make things break all the time!

Security Incident and Event Management

- Monitor -> Repair -> Distribute -> Reassurance.



The Patch Cycle

- Google SRS: *"Before you tackle a same-day zero-day vulnerability response,* ***make sure you're patched for the `top hits' to cover critical vulnerabilities from recent years."***
- *"If you are privy to information about a vulnerability under embargo, and rolling out a patch would break the embargo,* ***you must wait for a public announcement before you can patch along with the rest of the industry****. If you're involved in incident response prior to the announcement of a vulnerability, work with other parties to agree on an announcement date* ***that suits the rollout processes of most organizations—for example, a Monday***"

The Patch Cycle (2)

- Patch and Scan: patch everything you can, then develop tools to find the stragglers.
- Expedited rollout of 0-days: use the same tools, or you'll have trouble!
- Get your PR ready, and have a plan.
- Track outlier machines that can't be locked down.

Risk Management

- Insiders are the biggest risk, from carelessness AND malice.
- Need to embed control in the culture.
- Need policies that can handle 1% of staff going bad each year.
- Accountability: be way of shopping for compliance from audits, rather than security!

Risk Management (2)

- ISO27001 and Common Criteria largely failures (more in the Governance and Regulation lecture): principle of maximum complacency.
- Don’t let tickboxes get in the way of critical thought!
- Being a CISO is often thankless.
- Blame (and accountability) matters.
- You won’t know where the next disaster will come from, so be adaptive!