Lessons Learned Breaking the TDD Rules

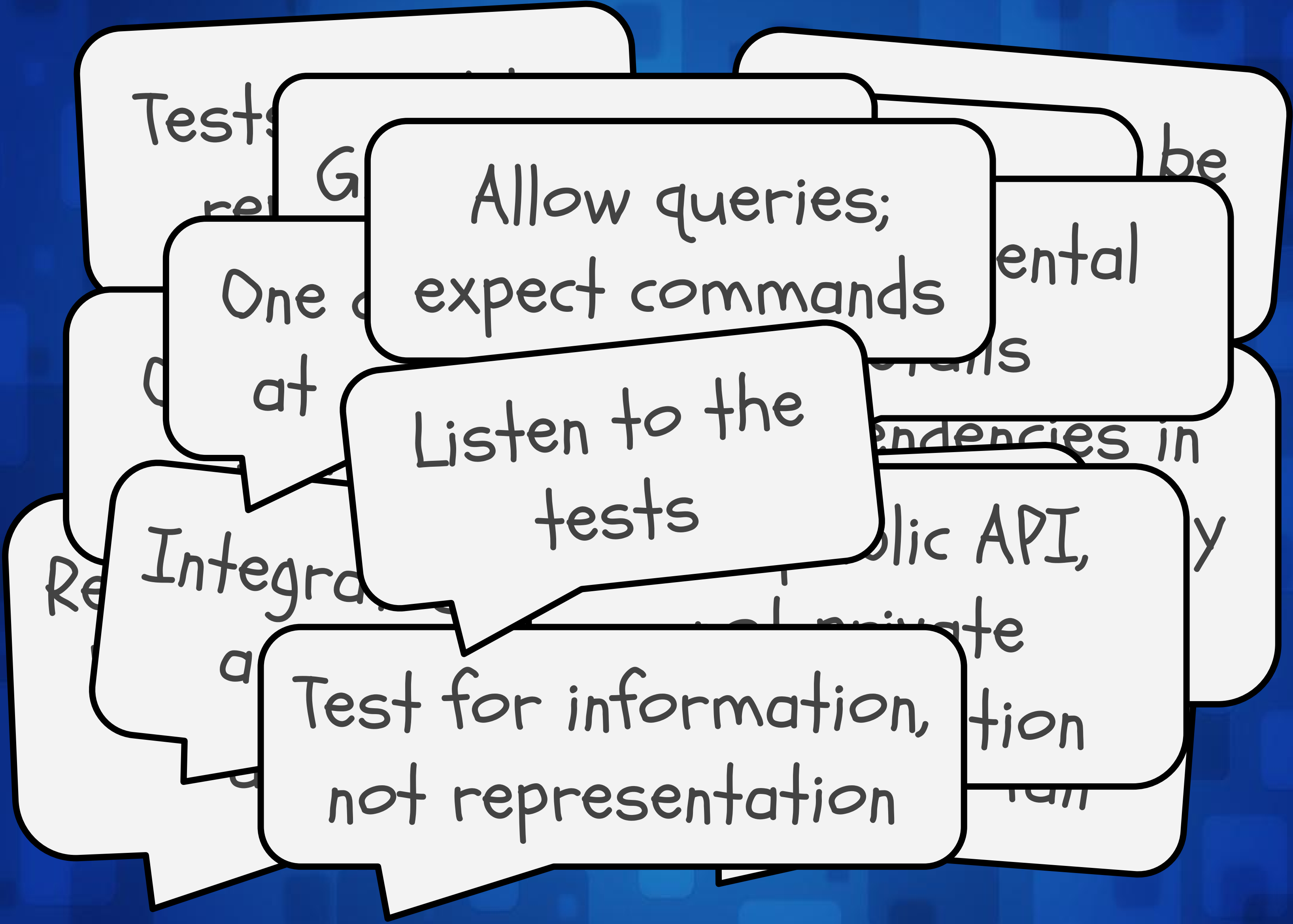
Nat Pryce
http://www.natpryce.com
info@natpryce.com
@natpryce
github.com/npryce

“You are not allowed to write any production code unless it is to make a failing unit test pass.

You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

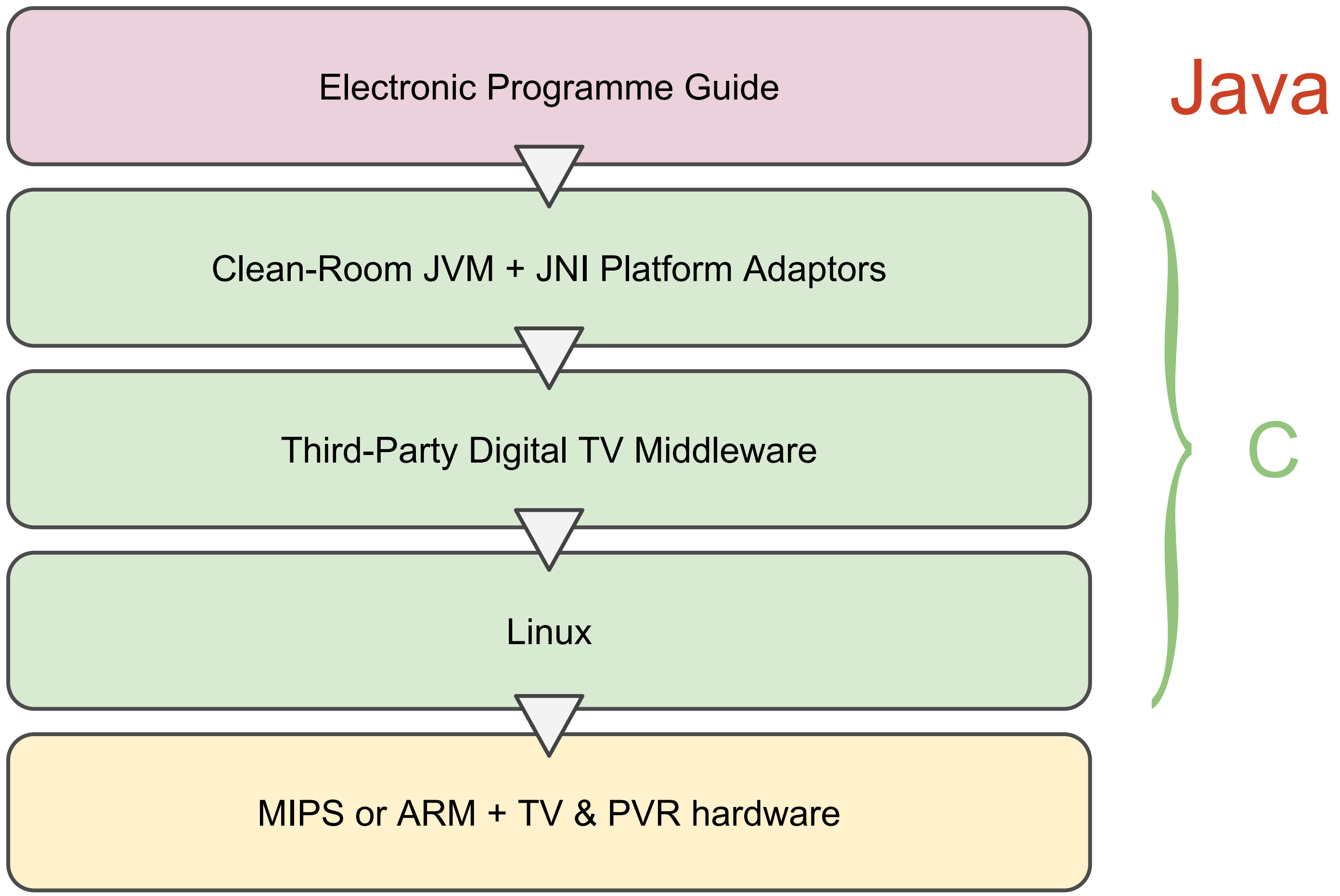
You are not allowed to write any more production code than is sufficient to pass the one failing unit test.”

Bob Martin

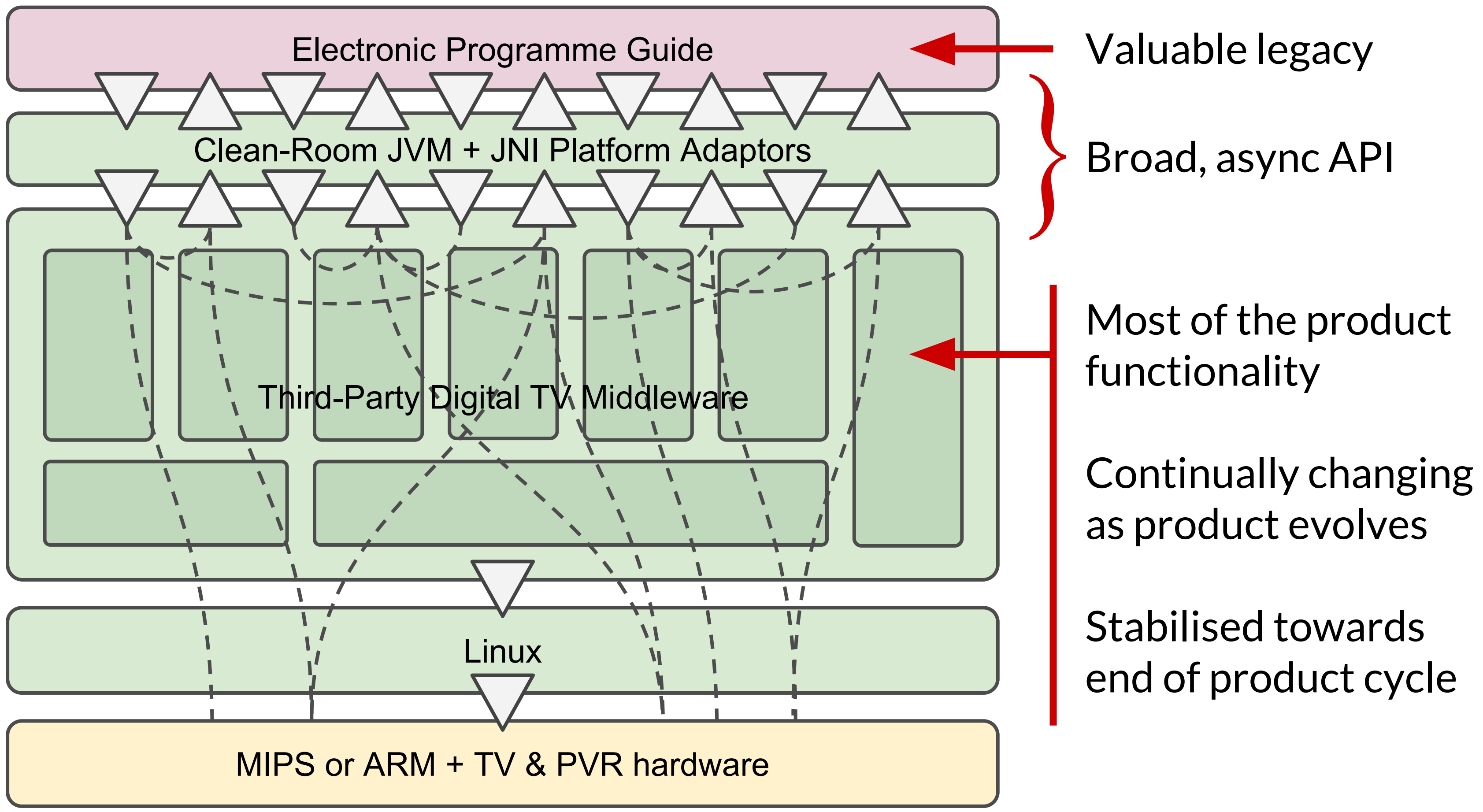

Allow queries; expect commands
Listen to the tests
Test for information, not representation

Digital TV PVR

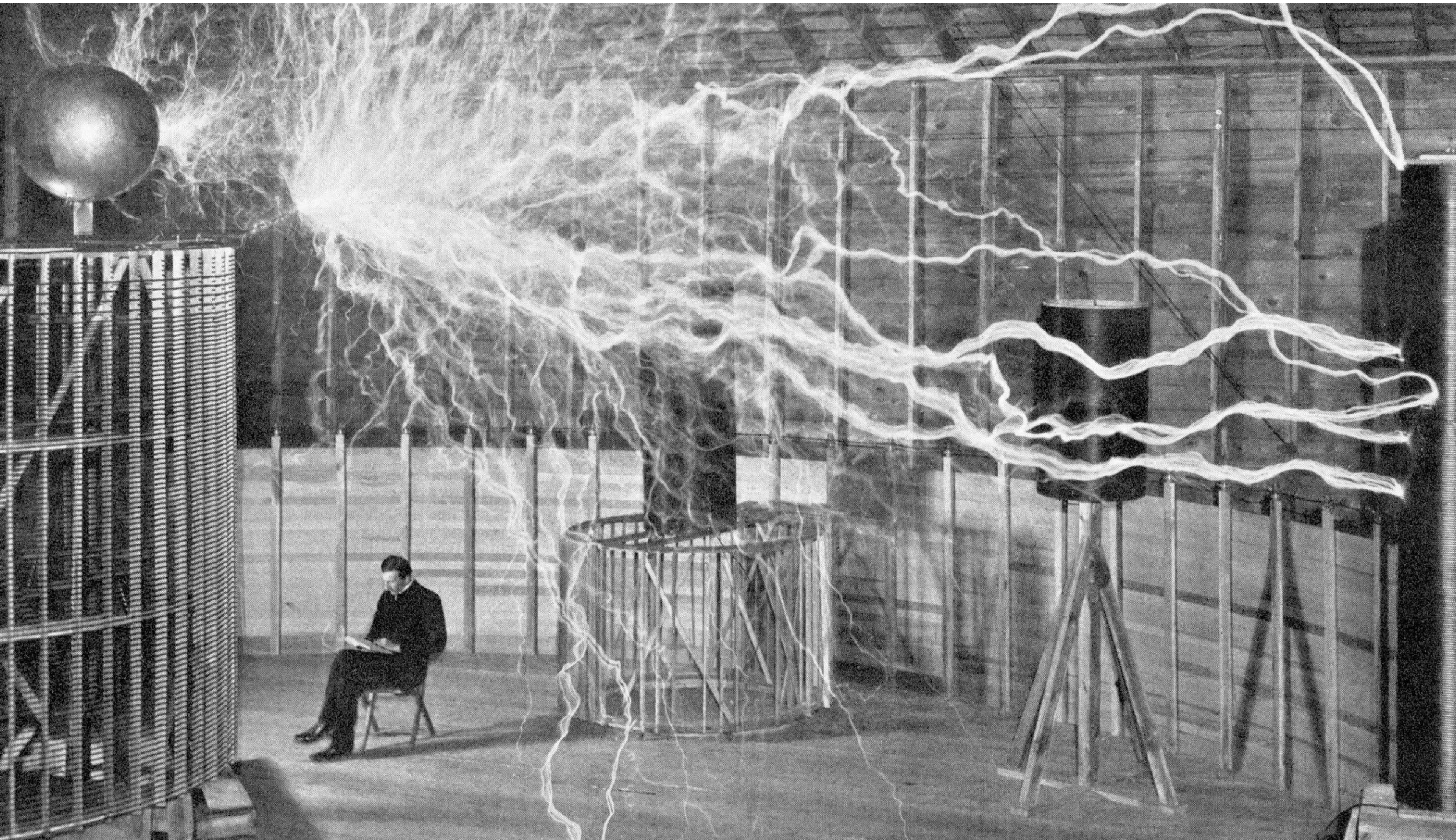
PVR Platform Stack


Electronic Programme Guide
Java
Clean-Room JVM + JNI Platform Adaptors
Third-Party Digital TV Middleware
C
Linux
MIPS or ARM + TV & PVR hardware

A More Realistic View

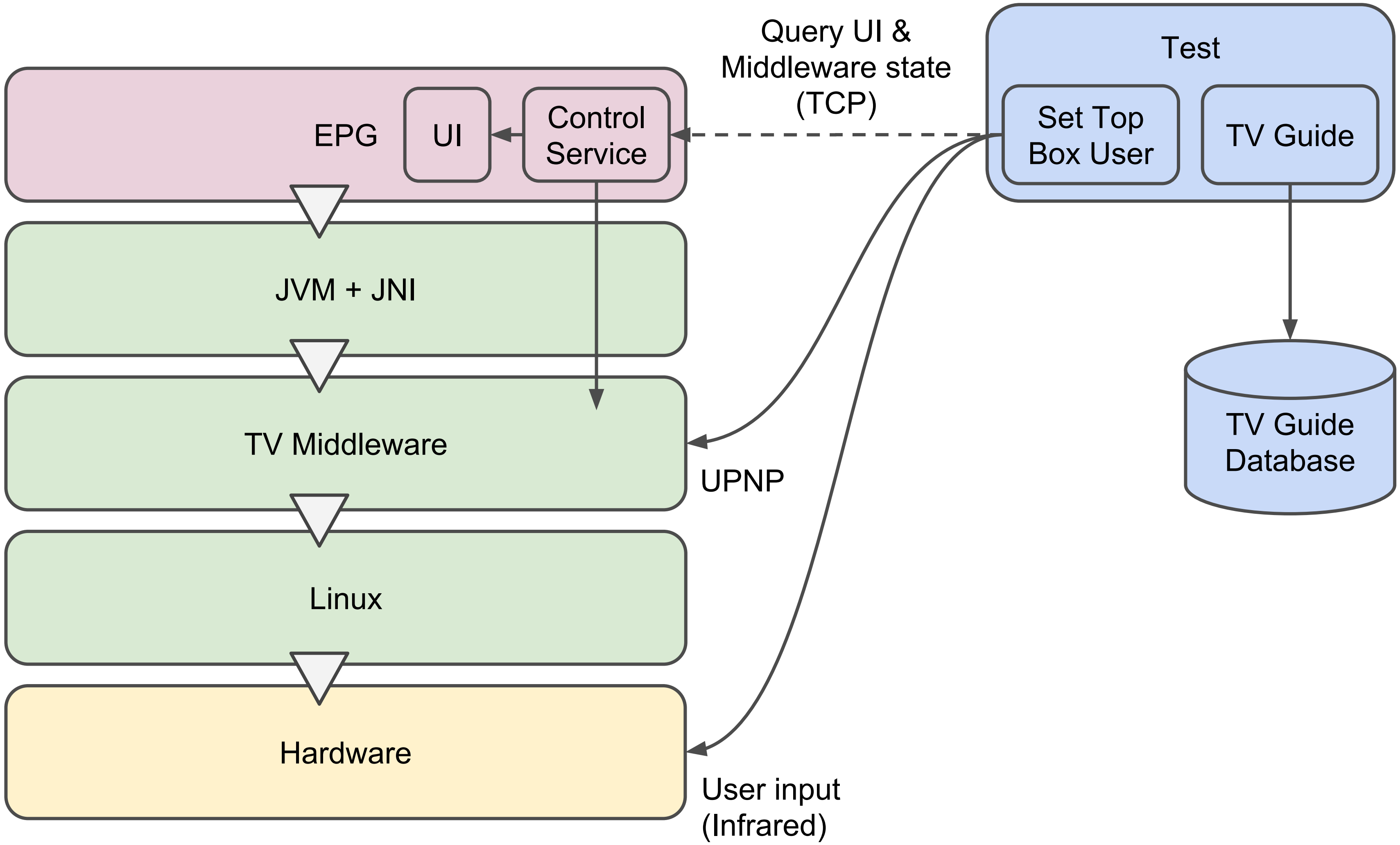

Electronic Programme Guide
Clean-Room JVM + JNI Platform Adaptors
Third-Party Digital TV Middleware
Linux
MIPS or ARM + TV & PVR hardware
Valuable legacy
Broad, async API
Most of the product functionality
Continually changing as product evolves
Stabilised towards end of product cycle

Shock! Testing with Live Data



We know the TV schedule

Functional Test Strategy



(Idealised) Functional Test

```
@Test public void
can_record_free_to_air_programme_from_guide_screen() {
    Showing showing = tvGuide.find(aShowing(
                    onAFreeToAirChannel(),
                    onAir(now()),
                    withDuration(greaterThan(minutes(5)))));

    Activity recordAndPlayShowing =
        on(Guide.SCREEN, Guide.record(showing)).then(
        on(Recordings.SCREEN, Recordings.findAndPlay(showing)));

    SetTopBoxUser user = startUsingSetTopBox();
    user.perform(recordAndPlayShowing);
    user.assertIsOn(FullScreenVideo.SCREEN);
    user.assertThat(FullScreenVideo.isPlaying(showing));
}
```

Unit-Level Fuzz Testing

```
JsonResponseParser parser = new JsonResponseParser();

@Test public void parsesResponseSuccessfullyOrThrowsIOException() {
    Mutator<String> mutator = new JsonMutator().forStrings();
    for (String validResponse : validResponses())
        for (String mutant : mutator.mutate(validResponse, 100))
            assertParsesSuccessfullyOrThrowsIOException(mutant);
}

void assertParsesSuccessfullyOrThrowsIOException(String json) {
    try {
        parser.parse(json);
    } catch (IOException _) {
        // allowed
    } catch (Exception e) {
        fail("unexpected exception for JSON input: " + json, e);
    }
}
```

http://github.com/npryce/snodge

Both Tests have the Same Structure

$$\forall\, x \in X\ P(x)$$

…as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns – the ones we don't know we don't know."

Donald Rumsfeld

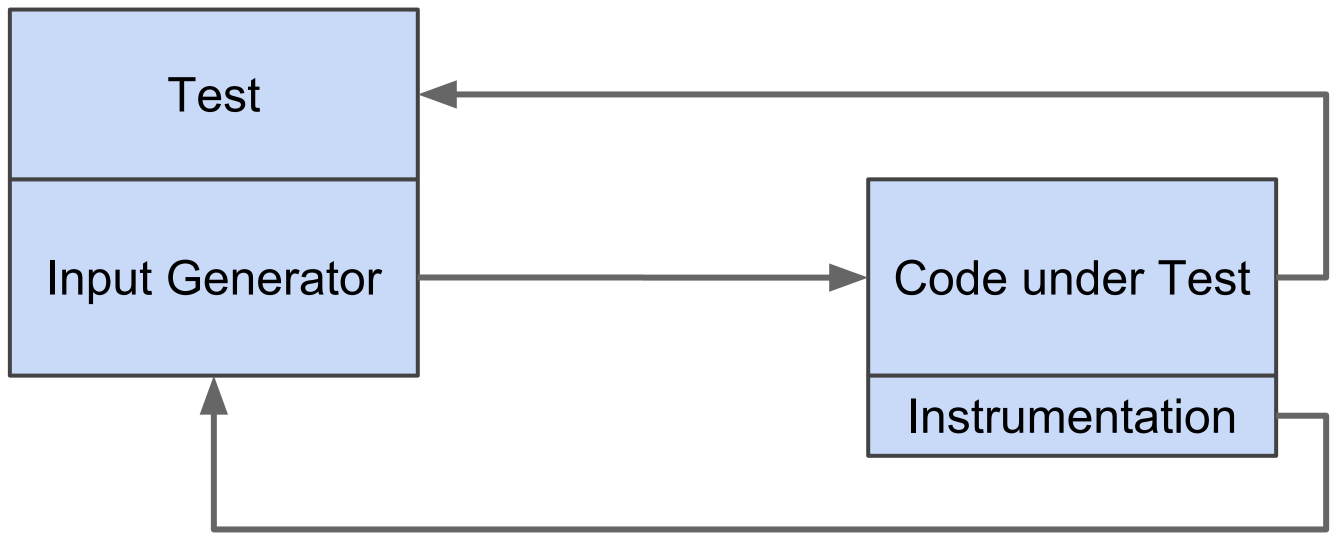
Lesson

Repeatable failure rather than repeated success

Lesson

Test automation is a search problem

Optimising Search-Based Testing



E.g. AFL http://lcamtuf.coredump.cx/afl/

A. Causevic, R. Shukla, S. Punnekkat & D. Sundmark. *Effects of Negative Testing on TDD: An Industrial Experiment*. In Proc. XP2013, June 2013.

"...it is evident that positive test bias (i.e. lack of negative test cases) is present when [a] test driven development approach is being followed. ...

When measuring defect detecting effectiveness and quality of test cases ... negative test cases were above 70% while positive test cases contributed only by 30%"

N. Nagappan, B. Murphy, and V. Basili. *The Influence of Organizational Structure on Software Quality: an Empirical Case Study*. 2008

“Organizational metrics are better predictors of failure-proneness than the traditional [software] metrics used so far.”

Organisational Measures

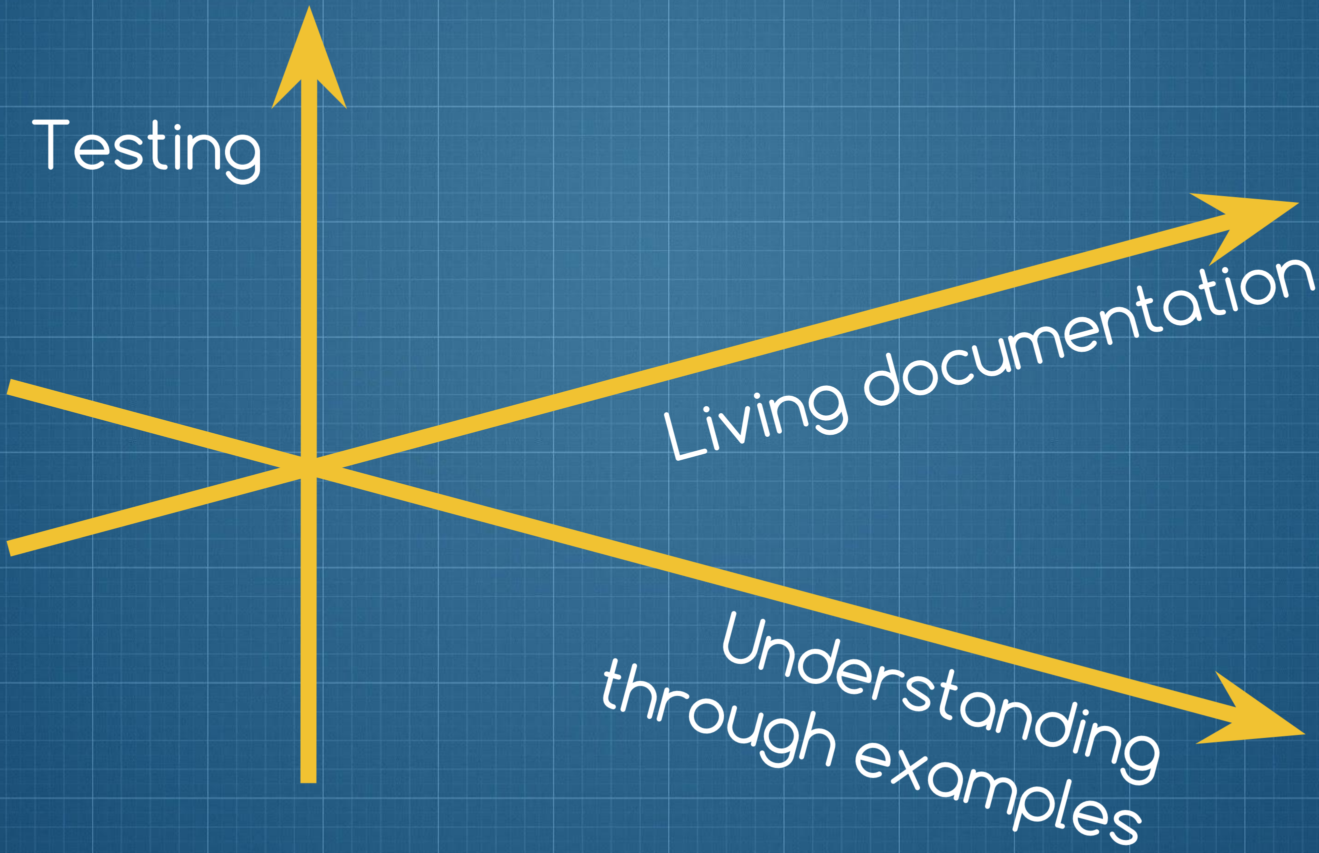
more people touch the code		→ lower quality
loss of team members	→ loss of knowledge	→ lower quality
more edits to components	→ higher instability	→ lower quality
lower level of ownership (organizationally)		→ higher quality
more cohesive contributors (organizationally)		→ higher quality
more cohesive is the contributions (edits)		→ higher quality
more diffused contribution to a binary		→ lower quality
more diffused organizations contributing code		→ lower quality

N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. *Change Bursts as Defect Predictors*. 2010

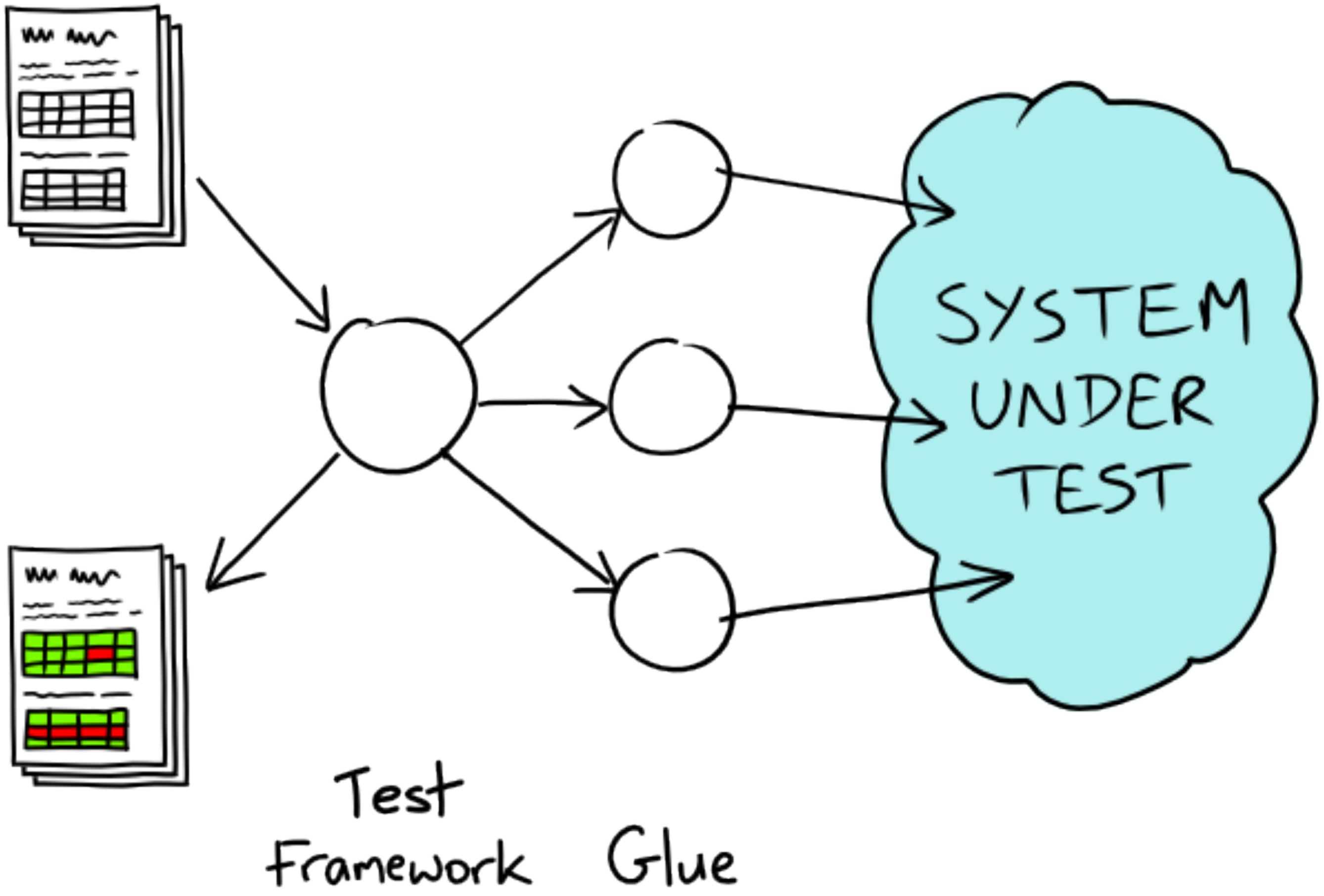
“What happens if code changes again and again in some period of time? … Such change bursts have the highest predictive power for defect-prone components [and] significantly improve upon earlier predictors such as complexity metrics, code churn, or organizational structure.”

What About Specification by Example?

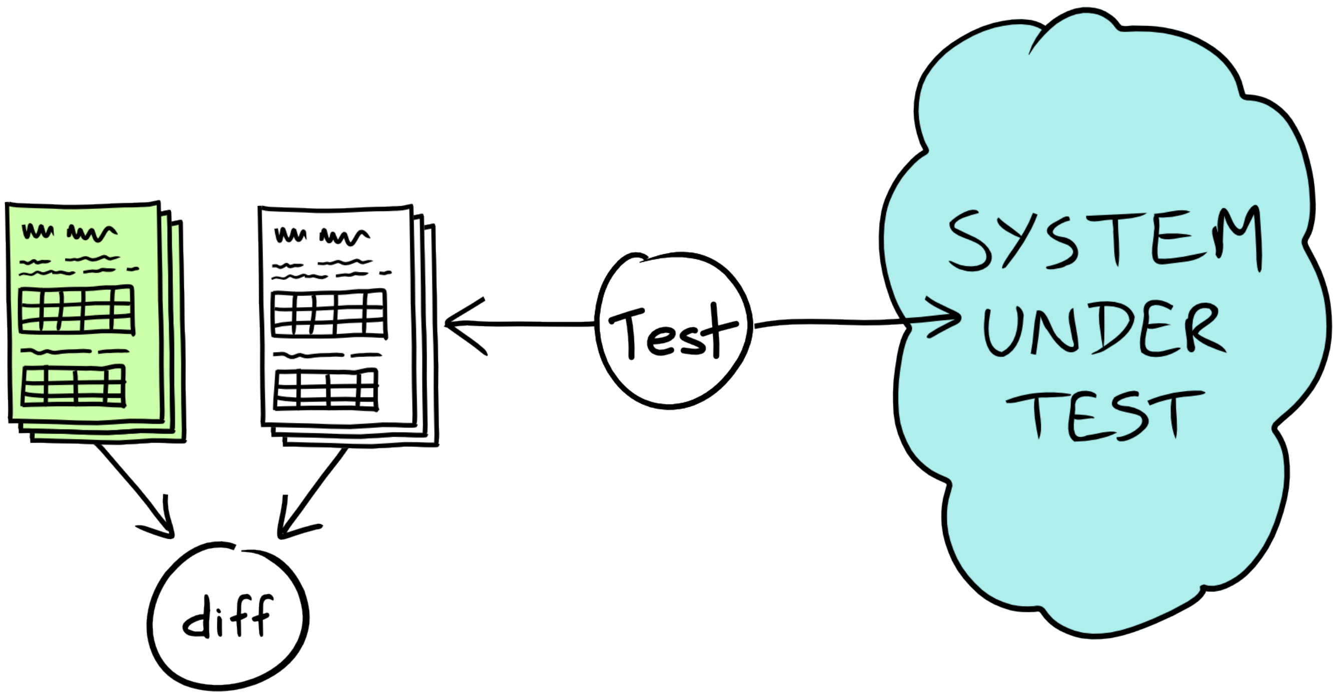
Lesson - Separate Concerns



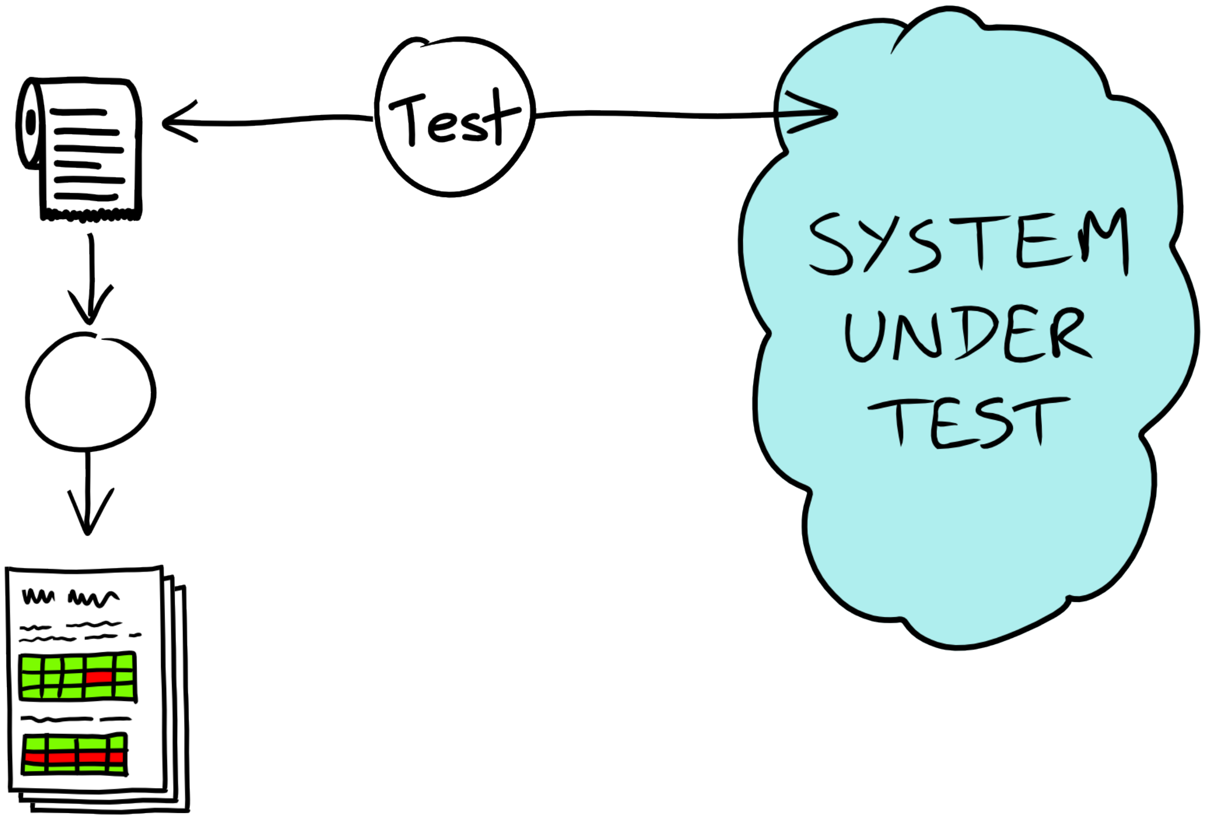
Specification by Example Tools



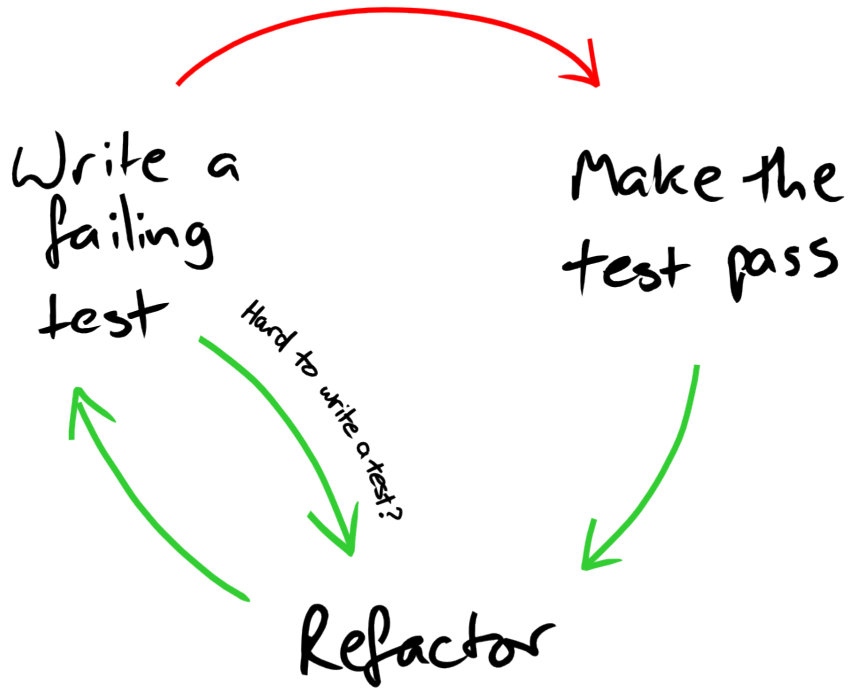
Approval Testing Tools



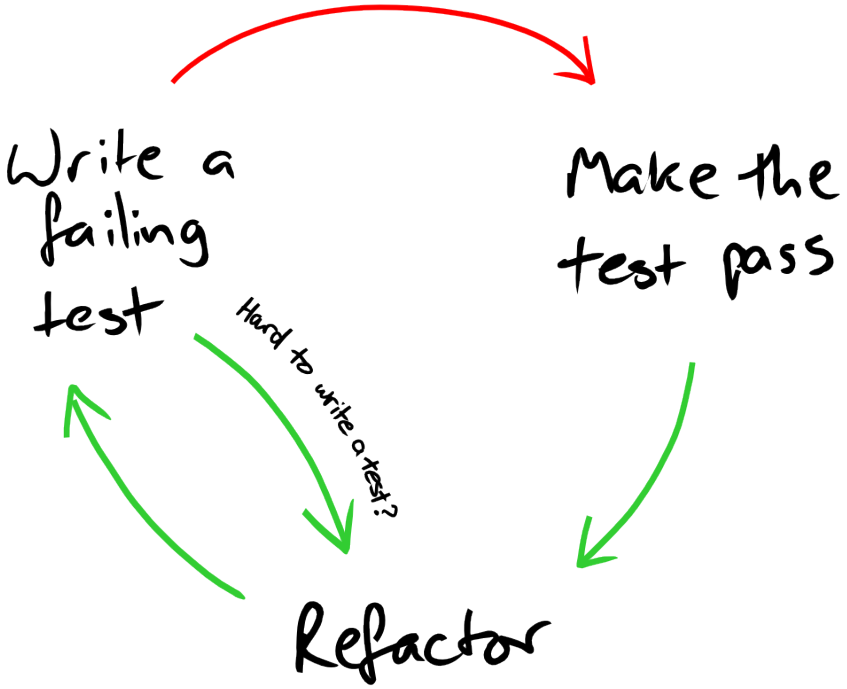
Generate Documentation from Test Log



Very few rules define TDD



Very few rules define TDD



The rest are made to be broken

Very few rules *define* TDD

The rest are made to be broken!

Nat Pryce
http://www.natpryce.com
info@natpryce.com
@natpryce
github.com/npryce
speakerdeck.com/npryce