One Sheet Summary

"Software Engineering at Google: Testing"

by Winters, Manshreck, et. al.

"The adoption of developer-driven automated testing has been one of the most transformational software engineering practices at Google."

Automated testing saves time, decreases stress, and increases profit.

MANUAL TESTING DOESN'T SCALE

The cost increases exponentially or worse as the system grows. At the same time, testing efficacy decreases, leading to more bugs, more fix time, more cost, and slower feature and release cycles.

IMPORTANT TEST QUALITIES

Speed and Determinism
Slow tests risk being skipped. Flakey (nondeterministic) tests cost investigation time.

Accurate
Invokes system same as user would

Hermetic, clear, consise
Contains all and only info required to run

Unchanging (ideal)
Unless requirements change

TEST SIZE

small run in single process
medium run on single machine
large run wherever they want

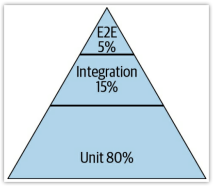
TEST SCOPE

narrow (unit) class/method
medium (integration) interactions between small number of components
large (end-to-end) interaction of several distinct parts of the system.

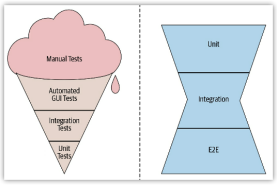
Prefer testing state over interaction
Prefer testing behaviors over methods
Prefer clear, useful, even verbose test names
Prefer realism or isolation (mock only when needed)

Google recommends mandating which testing frameworks everyone uses.

Strive for this



Avoid antipatterns



Creating a testing culture

"Any mandate on how to develop code would be seriously counter to Google culture and likely slow the progress, independent of the idea being mandated. The belief was that successful ideas would spread, so the focus became demonstrating success."

Instead:
* Include testing info in employee orientation
* Provide clear test adoption techniques and metrics
* Raise awareness (Testing on the Toilet)

"Changing the testing culture takes time."

Brittle Tests
* fail due to unrelated changes
* over-specify expected outcomes
* rely on complicated boilerplate
* misuse mocks

```
Code coverage only measures
   that a line was invoked,
not what happened as a result.
```

QA should do what humans do best: creative discovery. In short, *exploratory testing*.

Mocking Framework
Easy way to get doubles and stubs
Double
Object/function stands in for real implementation.
Fake
Behaves similar to prod, e.g. in-memory database. (And yet, "the team that owns the real implementation should write and maintain a fake."). If unavailable, wrap the API to create a fake.
Stub
Specify return values
Interaction
Verify function was called

Use these based on their applicability and fidelity. They can only be used if the code base is testable (dependency injection). Remember: the API behavior you're mocking might change!

"The primary reason larger tests exist is to address *fidelity*."
Challenges: who owns the test's failure? Lack of standardization.
New code accidentally only testable via E2E ("legacy within days")

```
"Tests that involve both frontends and
backends become painful because user interface
(UI) tests are notoriously unreliable and
costly: UIs often change in look-and-feel ways
that make UI tests brittle but do not actually
impact the underlying behavior."
```

```
Techniques for writing good tests can be
 the opposite for writing good production code.
"It can often be worth violing the DRY principle
 if it leads to clearer tests." This implies TDD
is hard because of context-switching, and writing tests
 is a different skill than writing production code
```

"The openness of our codebase . . . implies that many people will make changes in a part of the codebase owned by someone else."