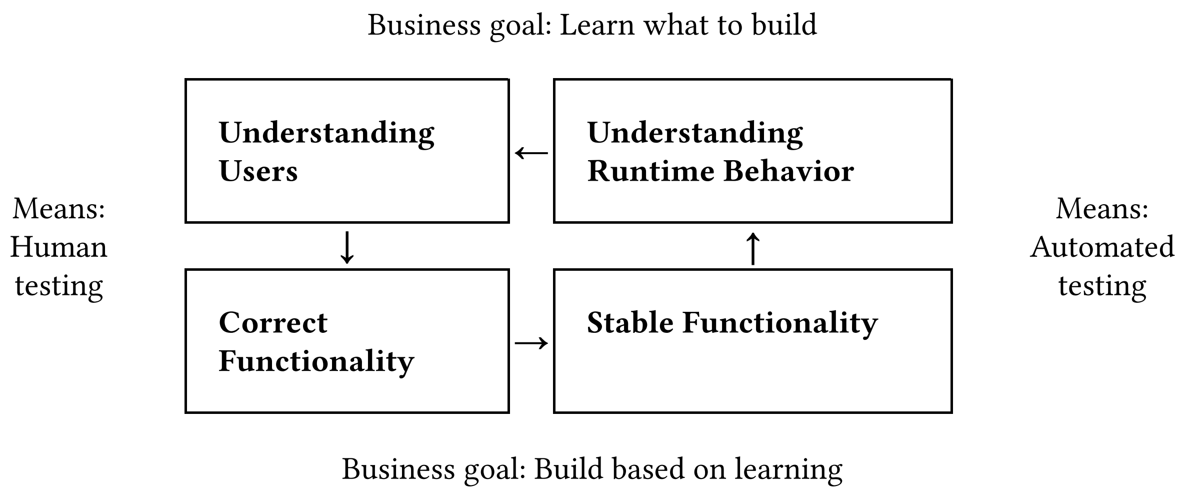
Software Testing: The Big Picture

Itamar Turner-Trauring (itamar@codewithoutrules.com)



Choose the appropriate ones to use for your particular needs and situation.

Understanding Users

- Will people buy your product?
- Will a design change result in more signups?
- Will users understand how your software works?

These are all questions that cannot be answered by comparing your software to a specification. Instead you need empirical knowledge: you need to observe what actual human beings do when presented with your software.

Relevant testing techniques include:

- Usability tests.
- Minimum viable product testing ("Lean Startup").
- A/B testing.

Understanding Runtime Behavior

- How does your software behave under load?
- Does your software have race conditions?
- Does your software break with unexpected inputs?

These questions can't always be answered by comparing your software to a specification. Once your software is complex enough you can't fully understand or predict how it will behave. You need to observe it actually running to understand its behavior.

Relevant testing techniques include:

- Stress tests and soak tests.
- Gathering tracebacks and exceptions from your production logs.

Correct Functionality

- Does your software actually match the specification?
- Does it do what it's supposed to?

It's tempting to say that automated tests can prove this, but remember the unit test that checked that 2 + 2 is 5. On a more fundamental level, software can technically match a specification and completely fail to achieve the *goal* of the specification. Only a human can understand the meaning of the specification and decide if the software matches it.

Relevant testing techniques include:

- Manual user interface tests (e.g. a QA person using your website).
- Code review.

Stable Functionality

- Does your public API return the same results for the same inputs?
- Is your code providing the guarantees it ought to provide?

Humans are not a good way to test this. Humans are pretty good at ignoring small changes: if a button changes from "Send Now" to "Send now" you might not even notice at all. In contrast, software will break if your API changes from `sendNow()` to `send_now()`, or if the return type changes subtly.

This means a public API, an API that other software relies on, needs stability in order to be correct. Writing automated tests for private APIs, or code that is changing rapidly, will result in high maintenance costs as you continuously update your tests.

Relevant testing techniques include:

- Unit tests, integration tests, and other similar automated API tests.
- Automated user interface tests (e.g. Selenium tests for websites).
- Compiler checks and linters.