The Way of the Web Tester

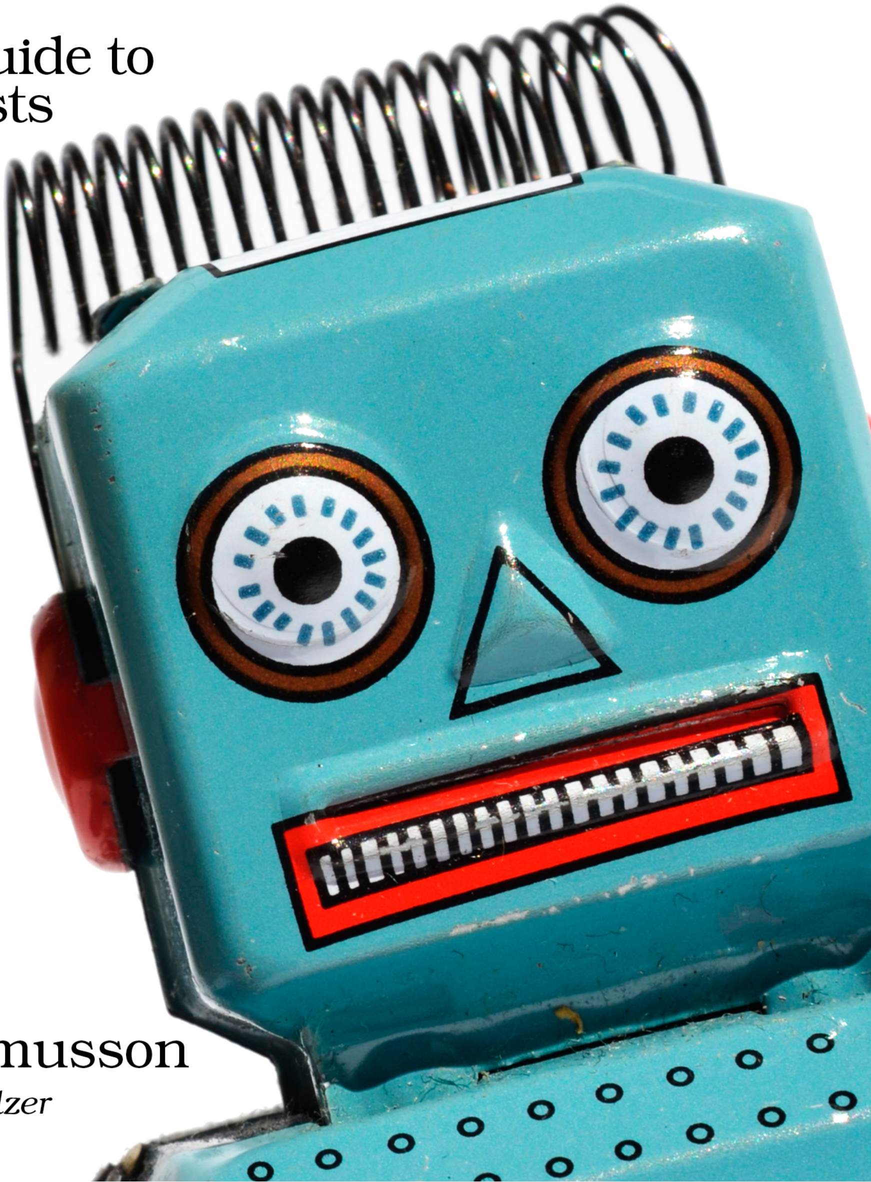
A Beginner's Guide to Automating Tests

The Pragmatic Bookshelf
Raleigh, North Carolina

The Pragmatic Programmers

The Way of the Web Tester

A Beginner's Guide to Automating Tests



Jonathan Rasmusson

edited by Susannah Pfalzer

The Way of the Web Tester

A Beginner's Guide to Automating Tests

Jonathan Rasmusson

The Pragmatic Bookshelf
Raleigh, North Carolina

The yellow adhesive note graphic in Chapter 11 is designed by Layerace from Freepik.com.

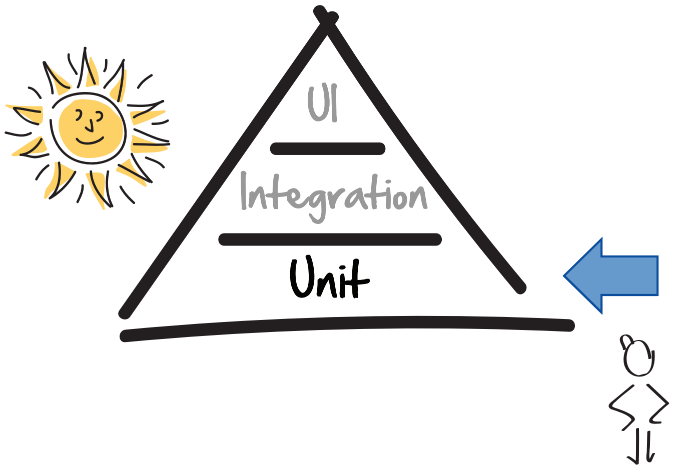
The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (index)
Nicole Abramowitz (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-183-4
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—September 2016



In this chapter, we're going to look at these small little tests developers write called unit tests.

While this chapter is primarily focused on developers, it's a worthwhile read for testers too. Learning what goes on down at the base of the pyramid will not only help testers spot potential gaps at the upper levels, it will also give them great insight into where they should go with their exploratory testing.

Regardless of whether you are a developer or a tester, by the end of this chapter, you will know what unit tests are, how to write them, and why they form the base of our pyramid.

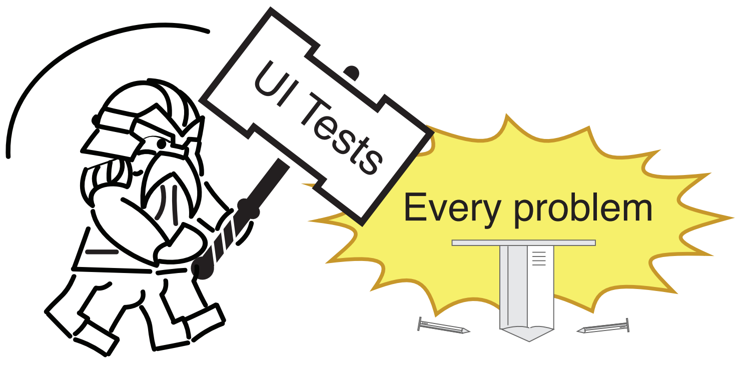
Everything Is Awesome!

Yeah! With our newfound UI testing superpowers, everything is suddenly awesome! Not only can we write high-level smoke tests, but we can also write UI tests for practically anything!

Need a smoke test? UI test.

Got a bug? UI test.

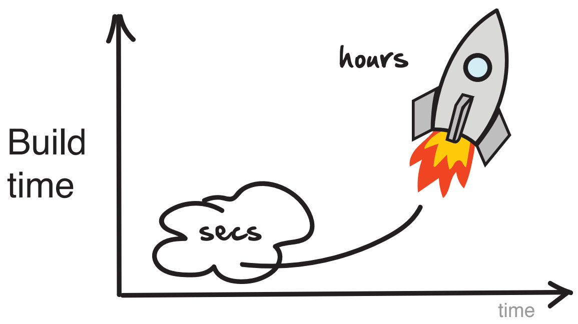
Need someone to fill out that pesky weekly timesheet? No problem—UI test (yes, we actually did that).



Yes, to us the world's problems can now all be solved with one more UI test, and things are going great! Except…

The Challenge with UI Tests

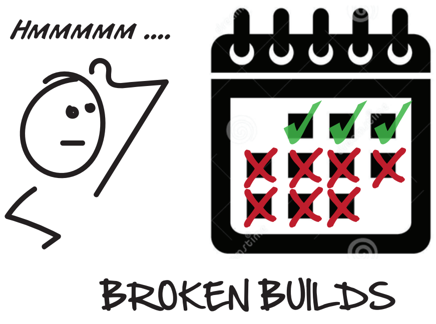
Hey! Have you noticed that our build times have started to take off?



LONGER BUILD TIMES

Huh! That's strange. What used to take a couple of seconds and minutes now takes tens of minutes and hours!

And what's up with the state of our builds? Why have they all of a sudden started to break?

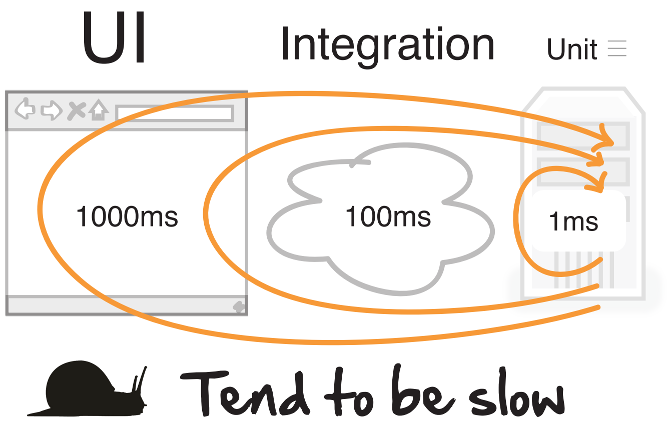


Beats me! But all I know is with our builds taking longer, and the tests constantly breaking, we are spending way more time fixing broken tests than adding new features to our software.

I thought these automated test thingies were supposed to help!

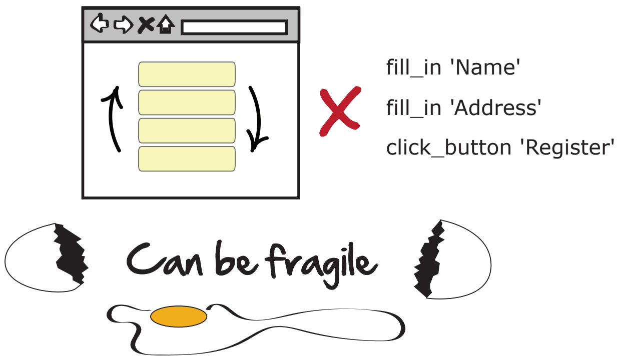
This real-life story of teams simultaneously discovering the magic and pain of going with lots of automated UI tests is unfortunately all too common.

It's not that UI tests are bad. They are not. They are just not made for the two things we crave above all else when doing rapid iterative development: feedback and speed.



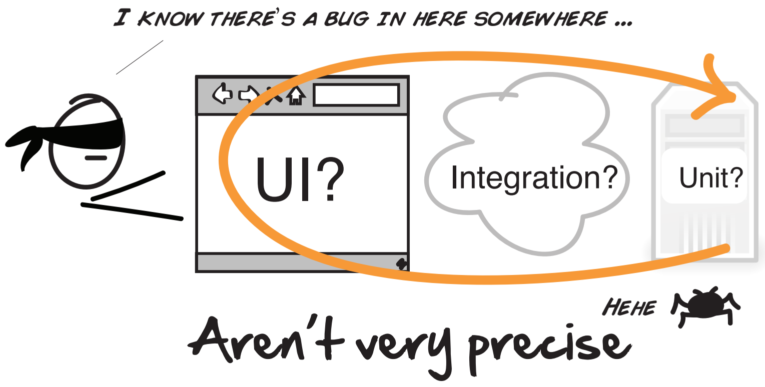
You see, UI tests are slow. Really slow. What takes milliseconds in a unit test can take seconds in a UI test. And while that may not sound like a long time, once you start to get a lot of these longer running tests, the cumulative time can really start to add up.

Not only are UI tests slow, they have a reputation for being flaky and fragile.



Flaky means they don't always run reliably—sometimes they pass, sometimes they fail (we will talk more about why later). But more than that, because UI tests are so closely tied to the user interface, the smallest change in functionality can end up breaking a UI test, even though it looks like it had nothing to do with it.

Finally, while UI tests are great at telling you that something's wrong, they are lousy at telling you where the problem is.



Remember—these tests go end-to-end. So finding and fixing a bug can be a lot like searching for a needle in a haystack.

Nope. As good and as cool as UI tests are, they alone are not enough. What we need is another kind of test. Something that's:

- Fast
- Cheap
- Precise
- Gives us rapid feedback

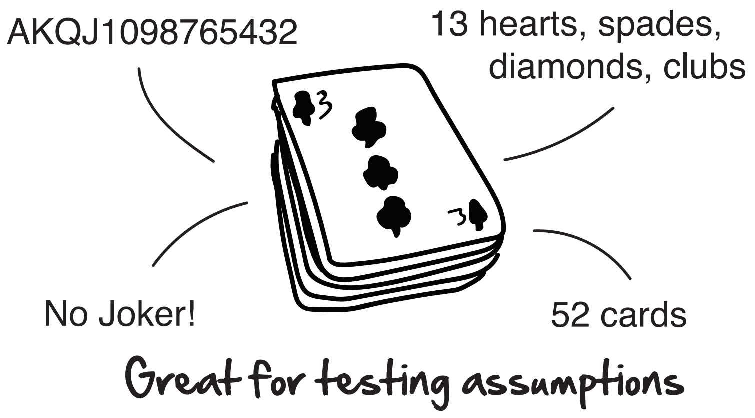
Enter the Unit Test

Unit tests are small, method-level tests developers write to prove to themselves their software works.

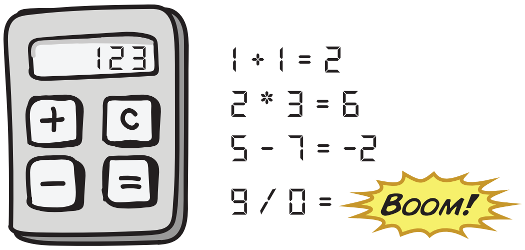
For example, say you were writing a program that could play blackjack, and you wanted to verify that all newly shuffled decks contained fifty-two cards. You could write a unit test for that. Something like this:

```
def test_full_deck
  full_deck = Dealer.full_deck
  assert_equal(52, full_deck.count)
end
```

Unlike UI and integration tests, unit tests are small and fast. They don't go end-to-end through all the layers of a system. They tend to be more local. And it's this smallness that makes them fast, focused, and easy to work with. You can write a unit test for just about anything—like testing assumptions.


AKQJ1098765432
13 hearts, spades, diamonds, clubs
No Joker!
52 cards
Great for testing assumptions

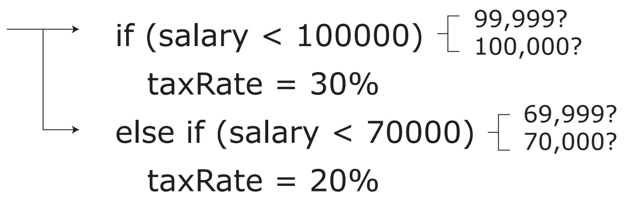
Assumptions get us all the time in software. Not anymore. With automated unit tests, hidden assumptions can now be tested and verified, along with business logic.



Calculations and business logic

Business logic can be complex. The rules we apply easily as humans all need to be codified and somehow tested in the software. What better way to verify we got the rules right than to code them up in the form of automated tests?

And when it comes to edge cases, unit tests have our backs there too.



```
if (salary < 100000)        99,999?
                            100,000?
   taxRate = 30%
else if (salary < 70000)    69,999?
                            70,000?
   taxRate = 20%
```

Edge cases and boundary conditions

Every time you can think of a new edge case, off-by-one error, or an error in logic, you can write a unit test to confirm these things are working as expected.

For these reasons, unit tests have become an indispensable tool for writing software today. This is why now every modern programming language has them.

Unit tests

FOR ALL YOUR COMPUTING NEEDS

Business logic	101	Edge cases
Program flow	011	Off-by-one errors
Assumptions		Permutations

Instant feedback!

100% Satisfaction! Guaranteed!

OK. So that's what unit tests are. Let's now dig a little bit deeper and see how these things work.