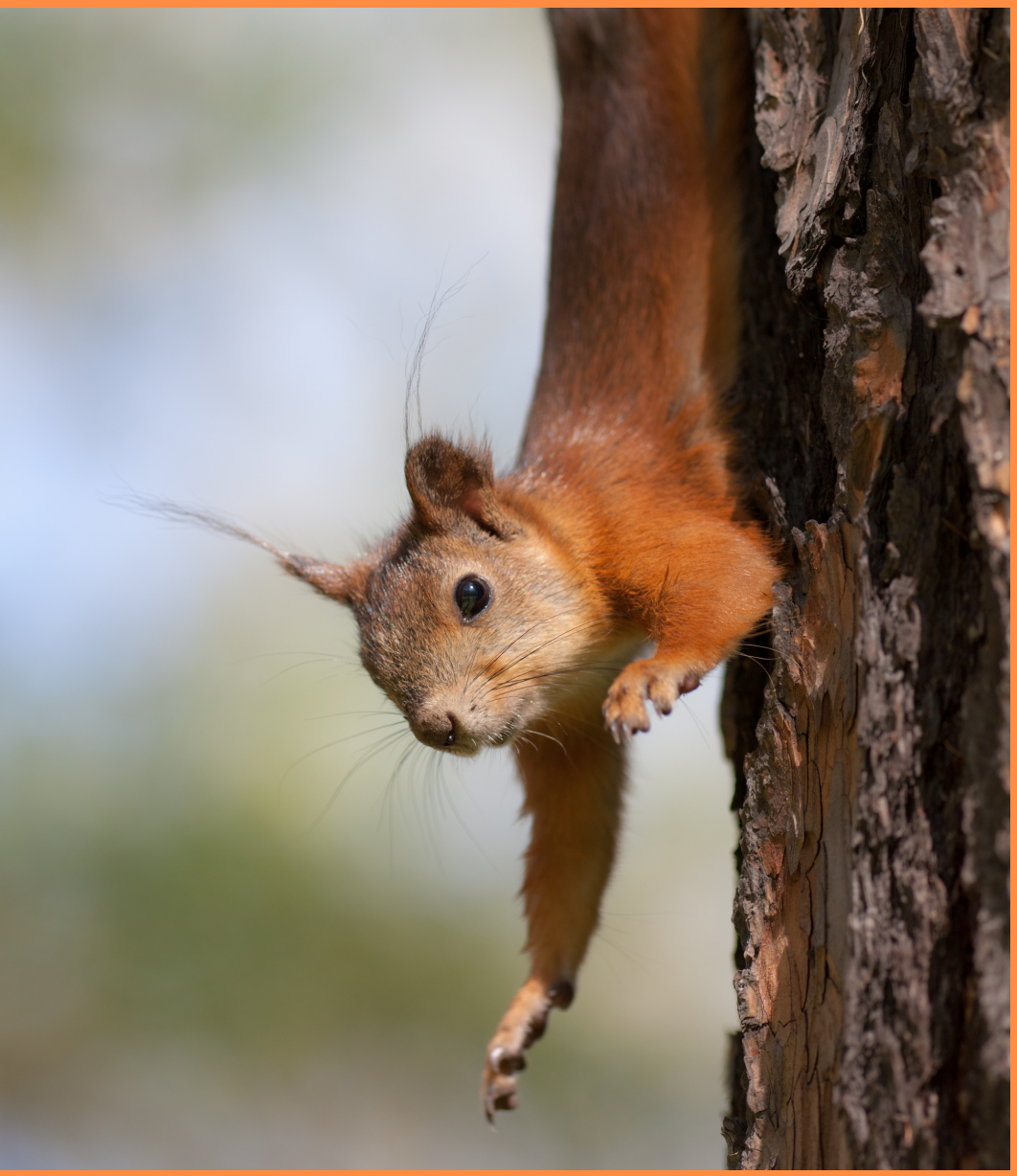
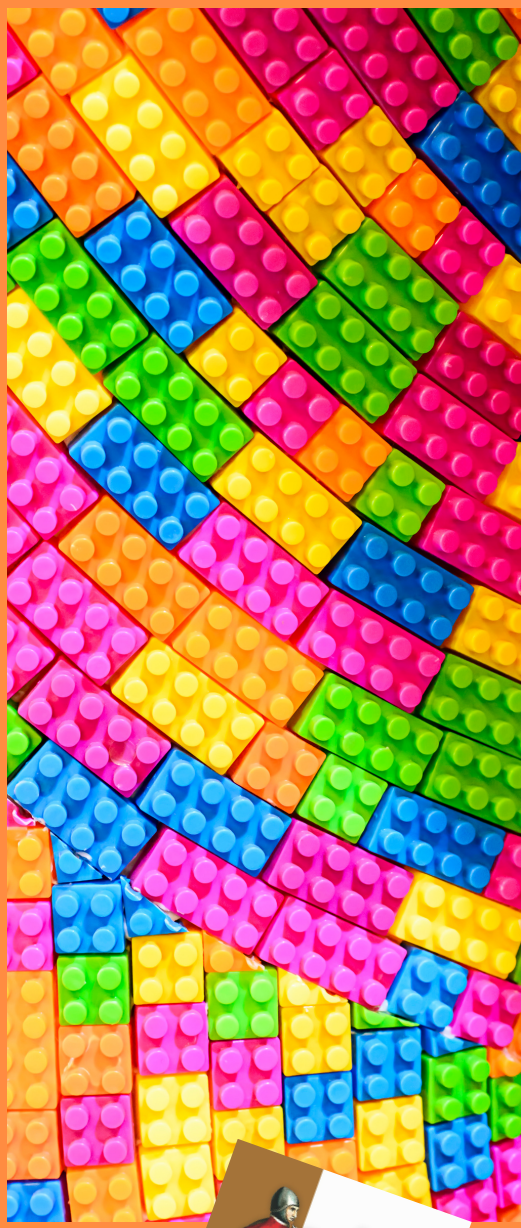
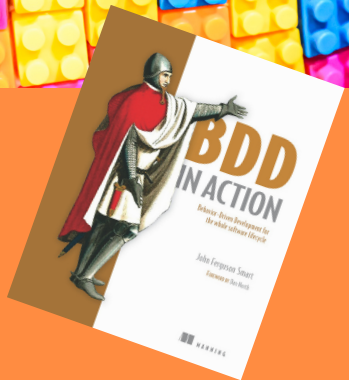
Serenity DOJO

# FLIPPED TESTING

**Build rock-solid test automation frameworks from scratch fast, and that speed up the whole team, without missing vital scenarios or failing to deliver on time!**

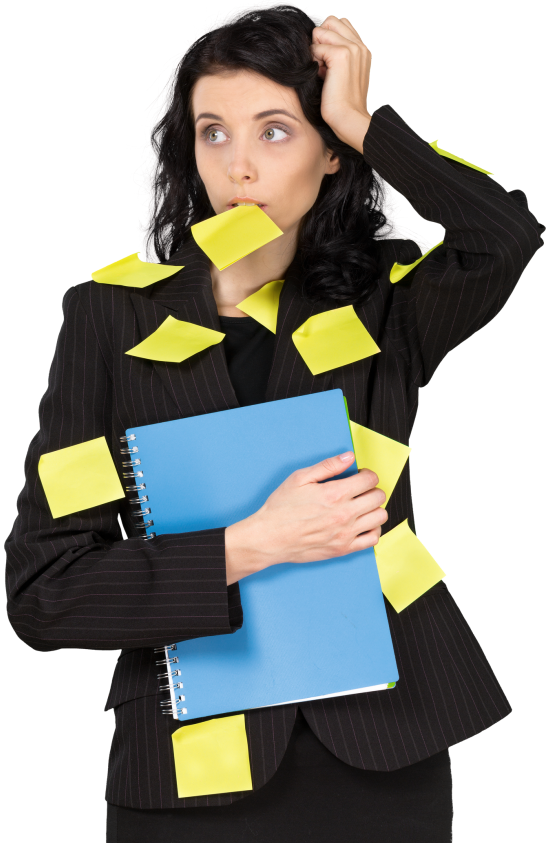







FROM BEST-SELLING AUTHOR

John Ferguson Smart



# TEST AUTOMATION CAN BE OVERWHELMING

**Do you struggle with test automation?** Asking questions like:

- Which test cases should I automate first?
- How should I structure my test suite when it grows?
- How can I possibly finish all the automation in time
- Or simply, where on earth do I start?

You're not alone!

Test automation ain't easy. And it is especially tricky with modern agile projects.

**But what if the reason we struggle is that we are going about it all wrong?**

## What if there was a better way?

A **more effective, more efficient approach**, far **better suited** to **modern agile projects**?

**I want to show you such an approach.**

It's an approach I have been personally using and teaching for over a decade, one that I learned working with some of the best agile test automation engineers in the world.

It's a proven approach that's helped build fast, rock-solid frameworks sporting thousands of business scenarios each.

**It works, very, very well.**

**It's an approach I call "Flipped Testing".**

**So read on, to see how you too can learn to build a test automation framework you can be proud of.**

# Traditional Test Automation Methods Do Not Work For Agile Projects

Automated testing tools have been around for ages. Almost three decades in fact.

The thing is, traditional testing strategies haven't really evolved much over this time.

**They still focus on automating your manual test cases, after the work is finished.**

Now if you're new to test automation, this approach can feel intuitive. It kinda feels like home - it's the way you do things with manual testing.. There are even a host of vendor tools that promise to let you magically automate your test cases without writing a single line of code.

**Problem is, it doesn't work. Simply automating manual test cases is actually the worst thing you can do.**

## Why? Because it'll slow you down

Writing automated scripts for your manual test cases traps you in a **vicious cycle** of late test automation, which **piles on the pressure** and makes it **impossible to finish automation in time** for the end of each sprint.

Simply put, if you only start automating after the features are done and tested, there's *no way* you'll have time to complete all the automation work you need to do within a sprint.

**But that's not the only issue.**

## Worse, it slows your team down!

Sure, it might catch the occasional regression defect.

But when you automate your manual test cases after the work is done, it's **way too late** in the game to help the developers on the team. No one will say thanks.

If the devs missed something or screwed up, they need to know about it **when they are working on it**, not afterwards. In other words, **during the sprint**.

## Plus, it's an *expensive* way to find defects

**Do you like wasting your time?** I know I don't!

But the fact is, automating your manual test cases is a **thankless, expensive chore**.

When you automate manual test cases, the natural approach is to automate via the UI. It's also the main approach all the vendor products promote.

But this is actually **the slowest and most inefficient way** to automate.

Sure, you need some UI tests, but having your whole test suite built on top of them is **a recipe for disaster**.

Worse still, the number of defects you find when you automate your manual test cases is actually **much smaller** than what you would find with with exploratory testing.

So while it has benefits for automated regression testing, we will see there are **must more cost-effective ways** to build up a comprehensive set of regression tests and to actually find defects in time.

**But there's another issue...**

## You end up testing the wrong things!

Most test cases, and most automated test scripts, focus on testing **what** an application does. You click on this button, then you enter this value, then you click on this other button, and so on.

But by just concentrating on what the application does (or how the user interacts with the application), it's easy to lose focus on something much more important: **what the application should do**, and **what the business needs it to do for them.**

You spend so much time worrying about how to click this button or select that dropdown value, that you forgot to check whether the application actually allows the users to do their job.

Which is the most important thing we can be checking! As the old saying goes, you can't see the forest for the trees.

Read on to discover

**three simple steps to truly agile test automation...**

# Agile Test Automation In Three Simple Steps

## Step #1) Automate requirements, not test cases

### "Cut out the middle-man"

Test cases are at best second-hand information - they are interpretations of the original requirements documents or acceptance criteria. So *automated test cases* are, at best, third-hand information.

Each step in the process takes time and effort (those manual test cases won't write themselves) and introduces the risk of missing some important detail.

A much better approach is to cut out the middleman and **write requirements that can be automated**. This involves **talking to business and other team members**, and agreeing on acceptance criteria and a few key examples of expected behaviour **before work starts**, and turning these examples into **business-readable, automated tests**.

The result is a set of **executable specifications** that you can use to **clearly demonstrate** to business folk that the application does what it is supposed to do **in their own language**.

**This simple strategy helped one tester I worked with decrease the time it takes to write new automated tests by a whooping 80-90%!**

This doesn't mean executable specifications are your only tests. But they are the starting point and the goal posts. Other tests, at different levels and written by developers or testers, expand from there until you have precisely the level of confidence you need.

```
Scenario: Approving KYC review
  Given the user logs in as: user@email.com
  And the user navigates to My Reviews
  When the user selects a review category: 'KYC
  And the user selects the first review
  And the user clicks on 'Open Review'
  Then the Review Details screen should be opened
  When the user fills in the comments field with 'All good
  And the 'Save' button is clicked
  Then the validation message is shown
  When the user clicks on 'OK
  Then show the review lis
```

```
Scenario Outline: Customers in high risk business must be escalated to a senior analyst
  Given Joe has been assigned the following review:
    | customer   | business risk rating | status   | assigned to   |
    | <customer> | <risk>               | <status> | <assigned to> |
  When Joe approves the review
  Then the review should be updated to:
    | status         | assigned to         |
    | <final status> | <final assigned to> |
  Examples:
    | customer        | risk   | status  | assigned to | final status     | final assigned to |
    | Alistair Capone | high   | pending | Joe         | pending approval | Unassigned        |
    | Bonnie Clyde    | medium | pending | Joe         | approved         | Senior Analyst    |
```

## Step #2) Demonstrate the outcomes, not the application

When you automate manual test cases, it feels very natural to test everything through the user interface. And, to be fair, when all you have is a UI automation tool, everything feels like a UI test.

But when you work with executable specifications, **everything changes**.

Your goal is to **demonstrate a business outcome**, not specific interactions with the UI.

This means you have the freedom to automate some scenarios as UI tests, others as backend or API tests, and yet others as a mix of both. Just pick the **simplest, fastest way** to prove that the application allows your users to achieve a specific business goal.

```
Scenario: Viewing tasks
  Given I am on the home page
  When I enter a valid username and password
  Then I should go to the home page
  And I should see a list of tasks assigned to my group
  When I click on one of the tasks
  Then I should see the details of the task
  When the task is assigned to me
  Then I should be able to modify the fields in the task
```

```
Scenario Outline: Analyst can view tasks in their groups but only modify tasks that are assigned to them
  Given Alan is an Analyst in the Compliance group
  And Task <Task Id> in group <Task Group> is assigned to <Assigned To>
  When he views the current list of tasks
  Then he should be able to view the task details: <Is Readable>
  And he should be able to modify the task details: <Is Modifiable>
  And he should be able to assign the task to himself: <Is Assignable>
  Examples:
    | Task Id | Task Group      | Assigned To | Is Readable | Is Modifiable | Is Assignable |
    | 1       | Compliance      | Alan        | Yes         | Yes           | No            |
    | 2       | Compliance      | Joe         | Yes         | No            | No            |
    | 2       | Compliance      | Unassigned  | Yes         | No            | Yes           |
    | 3       | Financial Crime | Susan       | No          | No            | No            |
```

**This is the exact same technique that another team I know used to build a test suite of over 1500 automated business scenarios that run in under 10 minutes!**

# Step #3) Model behaviour all the way down

Writing (or recording) a new test script for each test case is unsustainable. You'll burn out, and quickly!

If you are going to do agile test automation well, within the sprint, and still leave yourself enough time for exploratory testing, you need a test suite that grows. **And do to that, you're gonna need a plan.**

**The secret to scalable test suites** is how they are organised. They use **flexible components that model business behaviour,** because this is where you get the most valuable reuse.

Each layer breaks down and describes the behaviour of the next layer down, right down to where you actually interact with the system. This way you **maximise ease of maintenance, reuse and stability.**

**This is the key to writing tests fast, too. One team of testers I know used this approach to automate almost 800 rock solid business scenarios in under a month.**

```
Feature: Joining the Frequent Flyer Programme

  In order to encourage more regular travellers to fly more often with us
  As an airline manager
  I would like travellers to join our loyalty programme

  Scenario: Registering online for a new Frequent Flyer account
    Given Jane is not a Frequent Flyer member
    When Jane registers for a new account
    Then she should be sent a confirmation email
    And she should receive 500 bonus points
```

```
@When("{actor} registers for a new account")
public void registersForANewAccount(Actor theTraveller) {
    theTraveller.attemptsTo(
            Navigate.toTheFrequentFlyerRegistrationPage(),
            EnterRegistrationDetails.using(travellerDetails),
            Confirms.termsAndConditions()
    );
}
```

```
public class Navigate {
    public static Performable toTheFrequentFlyerRegistrationPage() {
        return Task.where("{0} opens the Frequent Flyer registration page",
                Open.url("https://frequent-flyer.flying-high.com"),
                Click.on(MenuBar.REGISTER)
        );
    }
}
```

## Don't be a slave to automation

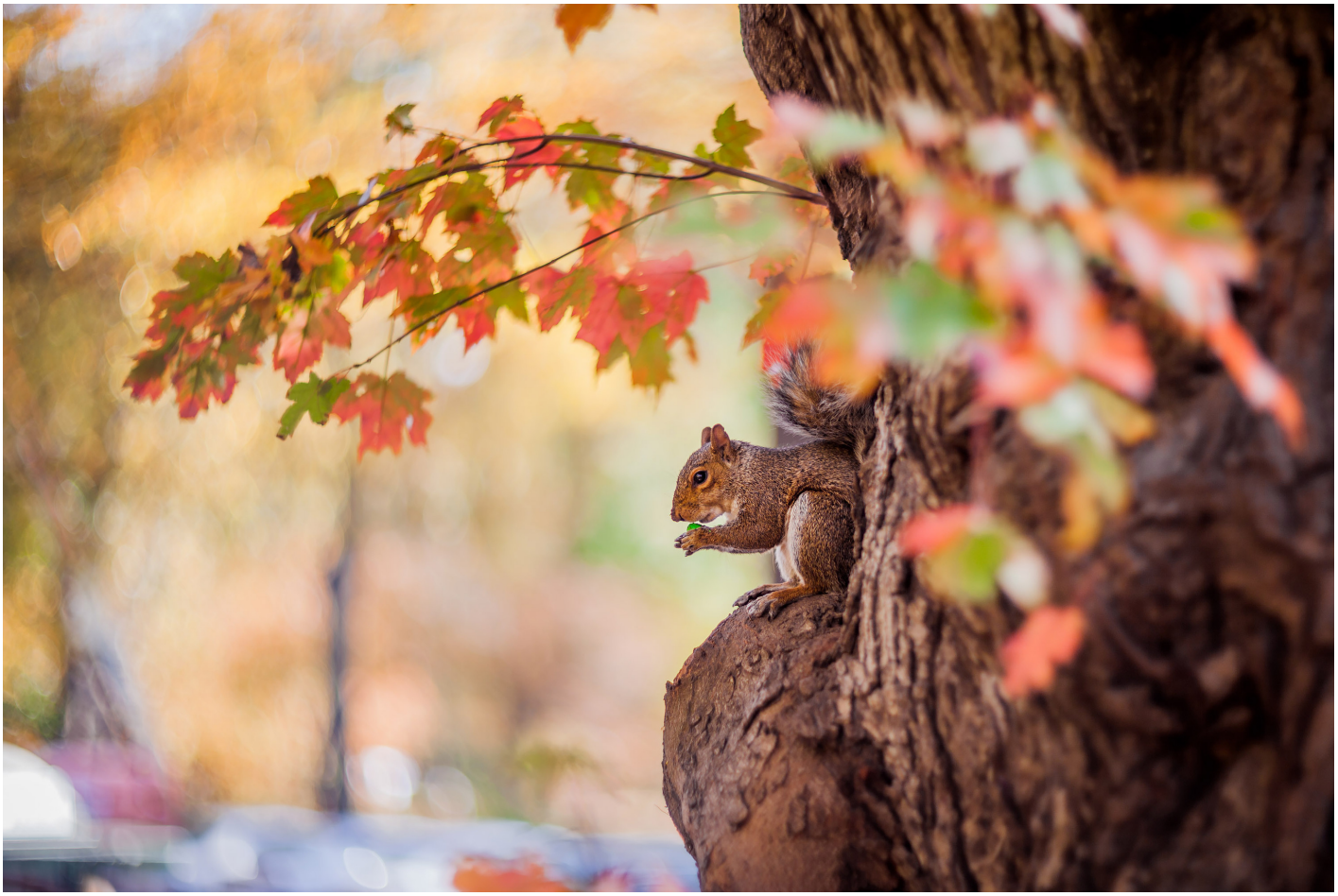
Test Automation is **essential** in an agile project. As a tester you **need to learn it,** and you need to learn **how to do it well.**

But test automation shouldn't mean an **endless struggle** with looming deadlines and falling automated tests that nobody believes anymore.

Automation should **free testers** up to do deeper, smarter exploratory testing. It should **provide automated evidence** on the state of the application, in terms that the business understands and trusts.

And the Flipped Testing approach that I've outlined in this document is a great way to deliver automation that truly **makes life easier for you as a tester**, and also **accelerates the development process as a whole.**

## Build a test automation framework that lasts.
## A framework you can be proud of.



# WHAT'S NEXT?

It's our hope that this document will give you some insight into the "Flipped Testing" approach.

We'd love to continue the conversation! If you want to learn more about modern agile test automation, **come join our LinkedIn Group** on the following URL:

**https://bit.ly/agile-test-automation-secrets**

If you're ready to talk about more tailored help in your test automation career, we can do that too. At Serenity Dojo we have helped hundreds of testers from all background become **high performing agile test automation engineers**, with a solid understanding both of the automation tools, and of the best way to use the tools in modern projects.

**I can't make test automation easy. It's not. But I can give you the skills and mindset to make it easy for you.**

So if you'd like to speak with us on how you too can learn agile test automation, we'd love to talk to you. There is never any pressure on our clarity call sessions. Our objective is to figure out what's stopping you becoming an awesome agile test automation engineer, and then help you figure out what the best plan of action is.

You can book a call with us on the URL below:

**https://www.serenity-dojo.com/apply**

Feel free to send us an email (if you need anything) or if you just want to say 'thanks for the guide'. (support@serenity-dojo.com).

Looking forward to talking soon!

John Ferguson Smart
Founder
Serenity Dojo