Building Automation Engineers From Scratch

Jenny Bramble

Jenny.bramble@gmail.com

Abstract

Creating automation engineers from manual testers is hard. Even if testers are willing, they have a lot of hurdles to get over to feel like the same kind of subject matter experts in automation as they are in manual testing.

As a career-long manual tester making the leap to automation, Jenny Bramble has experience to explain frustrations and provide solutions. She will discuss managing the expectations of testers and their managers (what's the time frame? Why isn't this working?), techniques for teaching (such as games! Pair/mob programming! Software fundamentals!), and how to know when testers have made it (what should manual testers be aiming for when they start?).

Biography

Jenny Bramble came up through support and DevOps, cutting her teeth on that interesting role that acts as the 'translator' between customer requests from support and the development team. Her love of support and the human side of problems lets her find a sweet spot between empathy for the user and empathy for the team. She's done testing, support, or human interfacing for most of her career and is excited about the future of automation.

1 Introduction

One of my favorite bosses asked me if I wanted to consider becoming an automation engineer. This would mean moving away from my decade-long manual testing career into the realm of coding—reflecting the degree that I'd been working at off and on during that time.

I told him no.

Honestly, I was perfectly happy as a manual tester! I got to work with the applications, find the mindsets of users, and prevent major issues from reaching production. I felt valuable, engaged, and excited about my job. There was no need, I felt, to take the leap into automation so I stayed deep in the trenches of manual testing. Sure, that meant that I spent eight hours one day running login tests to make sure that every permutation on every device was covered when we made a change. And maybe there was the weekly smoke test that took 12 hours of my life. But this was what I wanted to do—I wanted to be the user and advocate for them!

Looking back, I recognize that I was blocking myself off from an entire suite of tools that I need to be the most successful tester possible. There's a lot to be gained by adding automation to our abilities--being able to take some of the more trivial things we do and clear them from our plates so that we can focus on the interesting, complex, and sometimes purely confounding edge cases or UX testing that humans are best at doing.

I'm going to address these things over the next few pages. We will follow my journey as a manual tester making this transition into an automated tester and some of the expectations I've had to adjust as well as the challenges I've had to meet.

Adding automation skills to the toolbox of a manual tester is not a trivial undertaking. There are barriers that include perception of ability, velocity, and success that can be enhanced by the concern that their previous skills are no longer needed. You can reset these expectations and meet these challenges by creating a framework for success that speaks to learning styles and support. In the end, the life of a software tester is changing rapidly and she needs to be able to continuously learn and add new tools and abilities as often as she can.

2 Assumptions

I am going to make a few assumptions about the resources and humans you have available. There are a lot of artic les and presentations that talk about some of the things I'm making assumptions about. As such, this paper is going to gloss over them and concentrate on some of the things we don't talk about as much.

2.1 Resources

To begin with, you need people: either you yourself, or a group of manual testers who are interested in moving towards automation. A passing interest, a fancy is all that it takes to start down the path of becoming an automation engineer! But the spark needs to be there.

Second, you need to have time and resources. I'll discuss more in detail what this means later, but for now, you need to be able to dedicate time to this adventure. There are several talks and papers that focus on the mechanics of finding time, so this topic is considered out of scope of this one.

Next, a support system is vital for anyone adding a new skill to their resume. This can look like a supportive boss, a dedicated mentor, a group who is also working towards the same goals, or a team that is ready to support you. Finding a mentor or a support system is somewhat outside of this paper, though I will discuss how you can support someone moving into automation.

Finally--and most importantly--you are willing to make and execute a plan. This whole endeavor hinges on your willingness to step outside of your comfort zone into a challenging new phase of your career or to help others move into a new phase of their career. It's not a trivial undertaking because it requires confronting some long-held expectations for yourself and for the people around you as well as learning how to apply a set of skills that have been honed over a long career to a new way of thinking and operating.

When I started out, I found my favorite developer and asked him to teach me Ruby. He became a huge source of support for me and nurtured my budding interest in test automation and coding in general.

2.2 Humans

There are also humans involved in every step of the plan and I will be making several assumptions about them.

Every tester is an SME in their field. No matter what you call them, anyone who has made their career in testing software is an expert in testing software. We have spent years honing and fine tuning our ability to detect places in applications that may not meet the expectations of everyone around us and that's very valuable.

Related to that, manual testing is still very important. There will always be elements of a system that cannot or should not be tested in an automation fashion. There will always be points in the process that need the eyes of a human.

The next points are also related. First: not everyone wants to be an automation engineer. Second: change is scary.

Spending your day writing code is not something that everyone wants to do. However, I feel very strongly that every tester should have automation in her arsenal--it doesn't make sense to limit ourselves when we could have these skills in addition to all our others. At the same time, change can be terrifying. You're asking yourself or a group of people to step away from something they're secure in and have known for years to something new. That needs to be recognized.

3 Adjusting Your Expectations

Everyone has expectations. They permeate every part of our careers and lives. When you're building up automation engineers from scratch, it's vital to get out in front of the expectations that people have of themselves and that others have of them.

Let's discuss some common expectations and how we can adjust our thinking.

3.1 Expectation: Perception of Ability

I'm an expert manual tester. I've been testing applications for a long time on desktops, mobile, embedded systems, and more. During this time, I've learned to stretch and grow into different types of users—super users, musicians, digital advertisers, clinical research associates, blind users, patients in hospitals, music reviewers, and more. I pride myself on my adaptability and the way that I can navigate an application with the user's mindset.

When I started out dipping my toes into automation, I hadn't coded seriously for five years. I'dd fallen behind—no one had Visual Basic or Assembly or C tests. It was all languages that hadn't been taught when I was in school or that I'd picked up in my tenure as a Linux support tech. It felt like everything I'd learned was now useless and I was looking at starting over.

If you have been an expert in manual or human testing for a while, it can be very jarring to suddenly be put in a place you no longer feel like an expert. There's a picture you have in your mind of yourself--or

that you hope other people have of you--that can be hard to reconcile with being faced with an entirely new field of knowledge that you do not possess.

This perception can be a difficult force to combat because it is very subtle. Not often will people come out and directly say 'I'm stressed because I no longer feel like I am in my element.' Instead, look for signs such as timidity in moving forward or always taking on the parts of the jobs they have succeeded in before while leaving little room for the new parts that may involve coding.

3.1.1 Adjustment: Respect the manual

To adjust for the perception of ability, start by acknowledging and respecting that manual testers have valuable skills. Be careful not to disparage the skills people have built up over years and years of work. Automation complements and does not replace the manual tester.

One of the best ways to respect a manual tester's skills is by starting conversations with 'what are we going to test' instead of moving straight into 'how are we going to test this?' That gives everyone a chance to be heard and to have their skills reinforced. It will also teach other people on the team more about testing and what they should be looking for when they are passing a feature along.

I've found it incredibly valuable to sit with my developers and walk through PRs together—especially ones that involve unit tests. Not only does this give me a chance to look more deeply into the application code, but I can often find places that aren't covered in a way that I would cover when I apply my manual tester's skills.

3.1.2 Adjustment: Reset your mindset

Speaking of mindsets, you need to reset your thinking and reset your tester's mindsets. We have a natural tendency to assume we will be just as good at one thing as we are at something tangentially related.

Don't expect someone to be just as good at writing code as they have been at manual testing.

Do expect them to apply their stellar manual skills to the business of writing code. They won't churn out feature tests as quickly but no one knows the system from a user's perspective like someone in the trenches who's seen all the bug reports and handled all the defects. Even if you don't know the code, you know the paths users take in the app.

3.2 Expectation: perception of velocity

Let's talk more in depth about velocity.

Velocity is one of the metrics that is widely used—everyone wants to know how fast something will appear in the Done column. We talk about points, sizing, and commitments weekly if not more often. We press each other in stand up to make sure that things are moving.

When you're first learning something entirely new, it's hard to estimate how long it will take you. In fact, you don't know enough to even guess at what your velocity might look like. Using other people's velocity can undermine you as they have a much deeper toolbox to use than your new automation engineers.

3.2.1 Adjustment: velocity is not objective

We can see here that an objective view of velocity is not beneficial to your automation engineers. Holding people to expectations they cannot meet is a toxic situation.

Work with your team members to determine what they think they can complete in a sprint and give them the ability to make adjustments as they learn and become better at estimating.

Above all, don't fall into the trap of comparison. It's often easy to think that a group will have the same velocity, but as people ramp up, their velocity will need to change.

3.3 Expectation: perception of success

Success is a funny thing—there's a lot of expectations around what success looks like. Everyone in the process has an idea of what the end goal is and often these ideas are not aligned.

One of my favorite conversations always starts with 'are you done?'

Done with what? What does it manes to be done with a whatever thing I'm being asked about? Often times, the person asking isn't even sure what I'm doing, much lesswhat done would look like when I reach it. This puts me in a mush deeper conversation than anyone had expected starting out.

3.3.1 Adjustment: define your success metrics

Success or 'done' looks like something different every sprint and every week. If we don't sit down and define some success metrics for our adventures, we're going to have a hard time reaching success.

These metrics can be anything that makes sense for your team. Each team is different and in a different place, but there's a few constants.

As you're pulling together metrics to use, remember that they should be reevaluated often. There is also a chance for people to game the metrics or to use them as weapons against other people. You can help prevent this by checking in often with the team and making sure that your metrics still fit your needs.

Here's some metrics I've seen used.

- Everyone on the team interacted with a pull request
    - Putting one in
    - Commenting on a PR
    - Being part of an in person review
    - Merging something
- Golden Path or happy path is automated
- A person in particular has learned how to do something of note
    - Scrolling and tapping on iPhone
    - Using intents in Espresso
    - Creating a new page
- There are 15+ tests
- We got tests running in CI
- No one cried this week

The important thing is to define success. This will sometimes be driven by business, but really should be done by the test engineers and team at large. Start out the project with everyone's individual goals as well as team wide goals. This will let people feel like they are succeeding on both levels,

4 Set up a framework for success: how can you actually make this work?

4.1 What do you know?

I've said before that I'm an expert manual tester. This will never change. As long as I'm testing, I'll be improving that expertise and sharpening my skills.

While it can feel that way, automation is not an entirely new skillset. It is an augmentation to the skillset I already have. It's a tool in my toolbox that lets me apply the skills I've already gained and honed in a different way.

This is a skill set built on top of my existing skills. I know I say 'scratch' in the title, but no one really starts from absolute zero. You don't wholesale drop your other skills to pick up automation. Everything you've done in your career and in your life is a stepping stone to make your automation better.

On top of knowing how to test, you also know how to use tools and language. Think about the last time you picked up a new test case management system—how long did it take you to start really utlizing it? I've switched between three or four test case management systems in the past year. It's difficult, but we're used to ramping up quickly and becoming useful.

We also know how to use language. I often say that test engineers are the morale center of teams in large part because we do a significant amount of the communication on teams. We discuss defects with developers, product, and sometimes end users. We spend most of our days doing some sort of communication within our teams. This is an incredible skill when it comes to automation—coding is, after all, communicating what tasks you'd like a computer to perform for you. Once you get the hang of where to start, using your natural communication skills will help you far more than you can imagine.

Another of the top things a test engineer has is her logic skills. We puzzle through defects and convoluted systems. We spend our days asking questions and digesting the answers. This ability to work through an issue is often hard won by many years of work and is invaluable when you sit down to learn to code. Not only do you have to tell the computer how to step through a task, but you often have to find creative and interesting work arounds. Using your deduction and logic skills will let you have a much easier time with these tasks.

The final thing we'll discuss is your ability to learn.

No successful test engineer shies away from learning. Every new feature, use case, situation, or application that we run up against is an opportunity for us to learn. We have to learn in order to proceed through the situation—if we completely stopped learning, we'd be unable to test once we had a new feature. We have to learn what ig does, how it can be used, and who would be using it in order to do our jobs.

Adding automation to our skillset is not much different—it feels different, but in the end, we're learning a new way to communicate and to do our jobs better

4.2 What do you need to know?

When you're setting out to become an automation engineer, the first thing you think about doing is learning to automate. Don't fall into this trap.

Instead, learn to *code*.

When we talk about writing code as a test engineer, we often start by talking about automation frameworks and how to get our tests working. Instead, we should start by talking about how we can create quality code on the same levels as our developers. This isn't to say that we should expect ourselves to code at the same level as our coworkers who have been exclusively coding for their entire careers; more to say that we should aspire to have clean, efficient code that follows the standards of your platform and company.

In that same vein, the software development lifecycle at your company is another vital component. As test engineers, we often feel like we've got a really good grasp on the SDLC—but when you look at it

from a developer-centric perspective, it can look really different. Talk to your developers about how they move from tickets to code reviews and do your best to follow those procedures.

One group that I work with creates UI automation subtasks for each story. These subtasks are groomed, pointed, and slotted into sprints just like any other work. We move them through in progress to code review to testing. When they hit testing, they're usually been running on our build server for a few builds, so we're not looking at the code as it's written so much as we're double checking the manual testing that isn't covered by our automation. If we determine that there is more automation to do, we'll send the ticket back to the start of the workflow. This means that if a developer picks up a UI automation ticket, they're using a process that they are familiar with. It makes it much easier to share tasks.

And remember—automation code is production code. It tests production and is a vital part of the pipline for getting features to users. You have a responsibility to yourself and your users to write clean, maintainable code that is treated just as seriously as application code. If you wouldn't do it in prod, don't do it in your test suite.

5 What if you don't want to step away from what you've always done?

That's okay! Sometimes, the job we want is exactly the job we have.

However, the world of software is changing rapidly around us. Software testing is wildly different than it was years ago and will be wildly different in several years' time. If we expect to still be working in this field and making meaningful contributions to our teams, then we need to take advantage of every opportunity and advancement. This is why we go to conferences, read blog posts, and network. Humans want to advance, and we want to be better than we were. This is not a profession in which you can let yourself grow stagnant and still succeed. This is a profession of growth.

Automation is one way we grow. It is an incredibly valuable tool to add to your toolbox of skills and we are shorting ourselves and our commitment to quality if we refuse to investigate this tool. When we prepare ourselves for the challenges that we're going to see in the next 5-10 years, then we also prepare our software for those challenges.

Software development is not going to slow down and it's certainly never going to stop changing around us.

Building automation engineers is one of the ways we rise to meet those challenges.