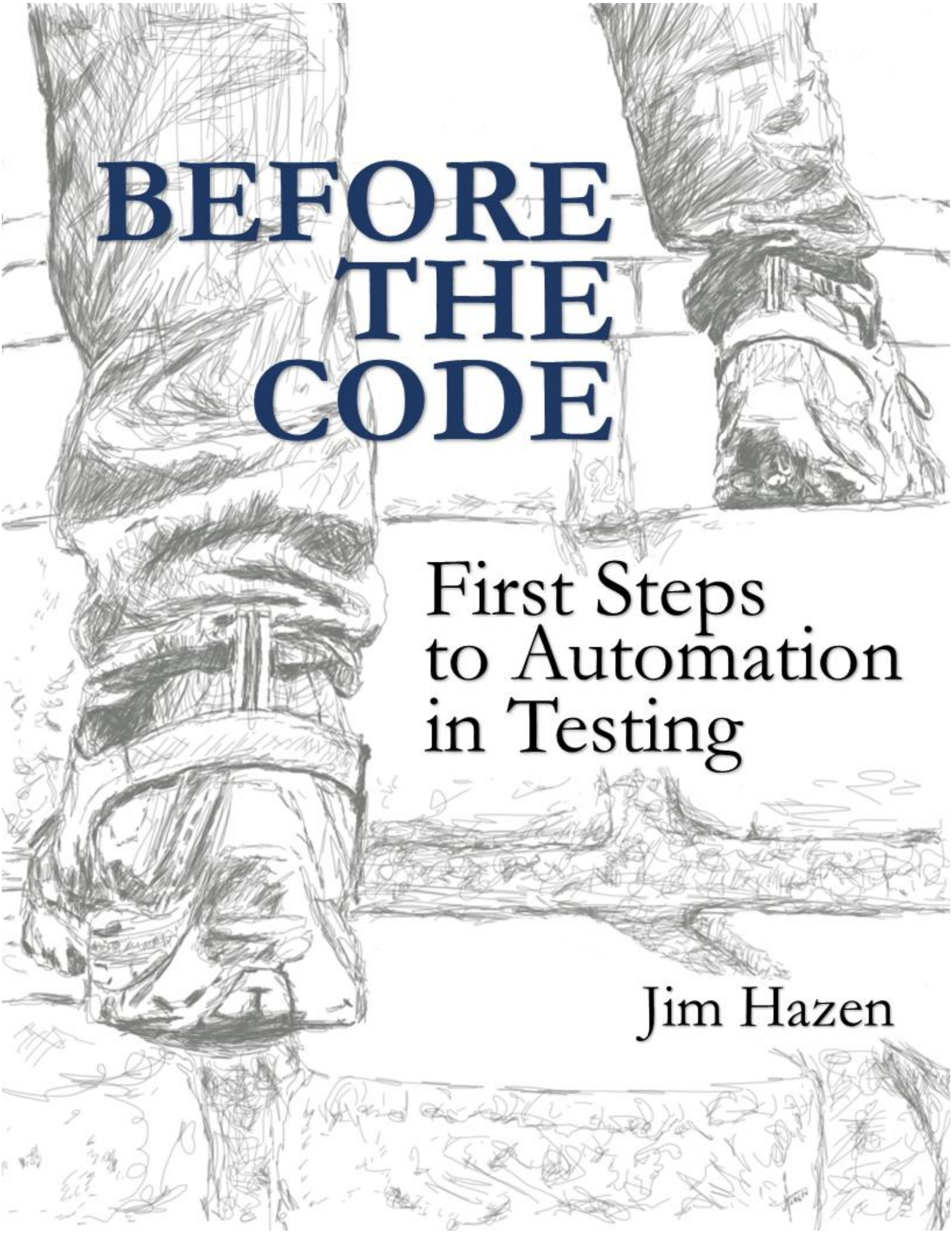

BEFORE THE CODE

First Steps to Automation in Testing

Jim Hazen

This version was published on 2019-03-07

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Cover art by Jason Hearn (thejasonhearn@gmail.com)

Table of Contents

ABOUT THE AUTHOR

Originally I started my career in computers as a programmer in the late 1980s and got into testing by accident like a lot of other people at that time. I was a good programmer but was better at testing software and figuring out how it worked, and why it didn't work. It all started at the company I worked for when I was recruited to help launch the Test department. After convincing management to send me to classes on software testing, one of which happened to be about test automation, I switched over full time. At that time most people would consider it is going in reverse, but I was good at it and felt I had found my niche in the software industry.

In the early 1990's I was lucky enough to work with the first generation of GUI (Graphical User Interface) test automation tools. One of which was for an OS/2 (IBM OS/2 operating system) Client/Server project for a large financial company. The automation tool itself was an OS/2 Client/Server application that allowed a central machine to connect to other computers to run tests in a distributed fashion. It was ahead of its time, and I did some cool work with it. Unfortunately, the tool fell off the radar when OS/2 failed to gain market share, and Microsoft Windows took over the PC platform. The tool vendor company didn't shift over to the Microsoft Windows platform quickly enough.

After that, I worked with Microsoft Windows PC based GUI tools over the next 8-10 years at various companies. In 2000 I went to work for a Test Consulting company that had private lab facilities that allowed me to work with multiple types of tools and technology stacks. I even got into Performance Testing and the tools for it. While working there, I presented at my first Software Testing conference in 2001 and continued doing so at a few other conferences for the next two years before a job change temporarily stopped my involvement in that arena.

From 2004 onward I continued to work as a consultant and contractor for testing tools and worked with multiple clients. Thus I've been able to work with many different projects and learned a lot from those experiences.

In 2010 I was again able to present at testing conferences and have stayed active to a reasonable degree. My speaking focuses mainly on automation in testing and the process of doing that line of work.

That is where this book comes into play; it is a culmination of those years of experience and conference presentations. A "brain dump" of sorts, in an organized form, of the last 25 plus years of my work with tools for testing and test automation. I've been there and done that so to speak. Learned some good lessons and had some hard ones as well. As my conference bio often says I'm a "Well-worn veteran of the software testing trenches," and have the experience to back it.

My intent with this book is to impart information of those experiences and help the reader, you, to avoid some of those hard lessons and have a fighting chance to be successful with your efforts relating to automation in testing.

Jim Hazen

ACKNOWLEDGMENTS

First off, I want to acknowledge and thank my co-workers over the years who have been sounding boards for ideas and work-related discussions. They are Igor Gershovich, Steve Romero and Don Smith. These three helped me to improve my knowledge and abilities as an automation developer, and are always great to work with on projects.

Second, I want to thank the following industry colleagues for their invaluable contributions to automation in testing and software testing in general. They are Joe Colantonio, Bas Djikstra, Elfriede Dustin, Dorothy Graham, Scott Barber, and Paul Grizzaffi.

Third, I want to thank Software Test Professionals (STPCon) for allowing me the opportunity over the years to be a presenter and instructor at their conferences. It has been a lot of fun to be part of their conferences and getting a chance to connect with industry friends I've known over the years and to make new ones.

Fourth, I want to thank the following people for reviewing this book and helping me to make it better. They are: Bas Djikstra, Damian Synadinos, Paul Grizzafi, Ali Khalid, Michael (Fritz) Fritzius, Igor Gershovich, Steve Romero, and Mike Lyles. Their feedback was invaluable.

Fifth, I want to thank Jason Hearn for the excellent job he did on the book cover. I really appreciate his efforts.

Finally, I want to thank Lori (my wife) and Grace (my daughter) for all of their love and support of me over the years that I have worked in the craziness called Software Testing. You both have helped to keep me grounded and sane. I dedicate this book to you both.

INTRODUCTION

One of the hottest topics today in software testing is automation. Why is that? Is it because everyone wants to make their work easier? Do testers want to do more extensive work? Is management demanding it? Or is it because the software industry as a whole is under so much pressure to produce products faster. Whatever the reason, the view is automation is a crucial part of the answer and solution.

You are probably asking yourself what makes this book different from others regarding automation in testing.

The main differentiator is the focus on the front-end processes — the work involved in implementing and maintaining test automation and emphasizing a tool agnostic mindset. All of this information is based on multiple years of work experience, using many different tools, and working with a wide range of application & development technologies. It comes from working in positions as a full-time employee or consultant, and at different levels as Tester or Lead or Manager. This information comes from my experiences in the Software Testing Trenches.

How to start learning about automation in testing is a good question and not an easy one to answer. By definition (Webster's) "automation" is "the method of making a machine, a process, or a system work without being directly controlled by a person".[1] In software testing's case it is both making a machine and system work without direct human control. Sound familiar? It should because that is what we do when we "program" a computer to perform a task. Be it a Spreadsheet, Word Processor, Database, Image Editing application or a complex business system for CRM/ERP. We "use" the "automation" to perform a task for us. We cause the behavior of some type to occur in the application we are automating.

Thus we first need to know two basic things in order to "automate" the process. First, how does the process operate; what are the steps to perform and the data needed for input, and then later validate regarding those steps and data inputs that have occurred. What types of outputs should we see? Second, how do we replicate those actions via non-human process? How do we "teach" the machine to replicate a human's action? The first one requires an understanding of "testing," or how to prove something via logic and Scientific Methodology. The second requires an understanding of how to tell the computer to do those actions, or how to "program" it.

Therefore in order to "learn automation," you would need first to understand the basic principles of Software Testing. And part of that is learning how to "model" the system and how to "test" it. Then you learn the basics of Programming itself, and a language to support that basic understanding. Both of these are quite large subjects, and many opinions abound around them regarding which test methods and programming languages to use. And they are covered in more detail in other source materials than what will be discussed here in this book. But they are foundational.

This book is going to focus on the other foundational subjects, specifically the ones that are not code and programming specific. These are the front-end tasks that need to occur before a line of automation code is written, and in some instances before the target application code itself is written. It will also cover items and tasks to consider during the writing of the automation code, and later on while maintaining it. It will also discuss some of the other hot topics surrounding automation in testing and how they may impact you and your work. So instead of just diving in head-first with a tool and programming automated test scripts, this book will show you the other

things to consider first. It will try to give you a broader foundation to work from and allow you to have a better chance at success with this type of work.

References

1) "Automation," Merriam-Webster Dictionary, www.merriam-webster.com/dictionary/automation

CHAPTER 1 – STARTING OFF

Where to Start

Isn't that the $1,000,000 question the majority of people and companies ask when they first start working on implementing automation for software testing? Typically, they get a tool and go for it, which sets them up for failure. The saying "A fool with a tool is still a fool" rings true in that case. So don't be foolish. To avoid that pitfall let's take a step back and get a better understanding of what we are talking about here.

What is test automation? It's a big topic to discuss and a bigger one than we think. To start, Webster's Dictionary provides one definition of Automation as "the technique of making an apparatus, a process, or a system operate automatically." [1] What does it mean to make "an apparatus, or process, or system operate automatically"? For our purposes, the meaning is setting up a test to run autonomously without human intervention. Using machines to offload recurring tasks, and to leverage economies of scale. Automation achieves efficiency gains and allows humans to focus on high-value tasks.

Automation is the use of a tool; specifically to aid a human in their work. For us, as testers, it is using software to support the process of testing other software. Automation is also using a machine to perform specific tasks in a repetitive manner so that we, the human tester, don't have to.

Unfortunately, in software testing, the use of automation tools for the execution of a test only is the preeminent line of thought by most people and companies. They see it as some "magic monkey" that will do their work for them. Execution of a test is only a small part of the process of testing. Other ways of using tools include test design, test data generation, scenario modeling, code coverage analysis, defect tracking, test execution management (along with test results analysis and reporting) and test environment management.

More appropriately is to say "Automation in Testing" [2] is the use of a tool, or tools, during any part of the process.

Now that we have cleared that up a bit let's switch back to the topic at hand: the steps to take first for implementing automation in and for testing. This book's focus is on the steps needed for the implementation of automated tests.

Initiation of Implementation

See a problem here?

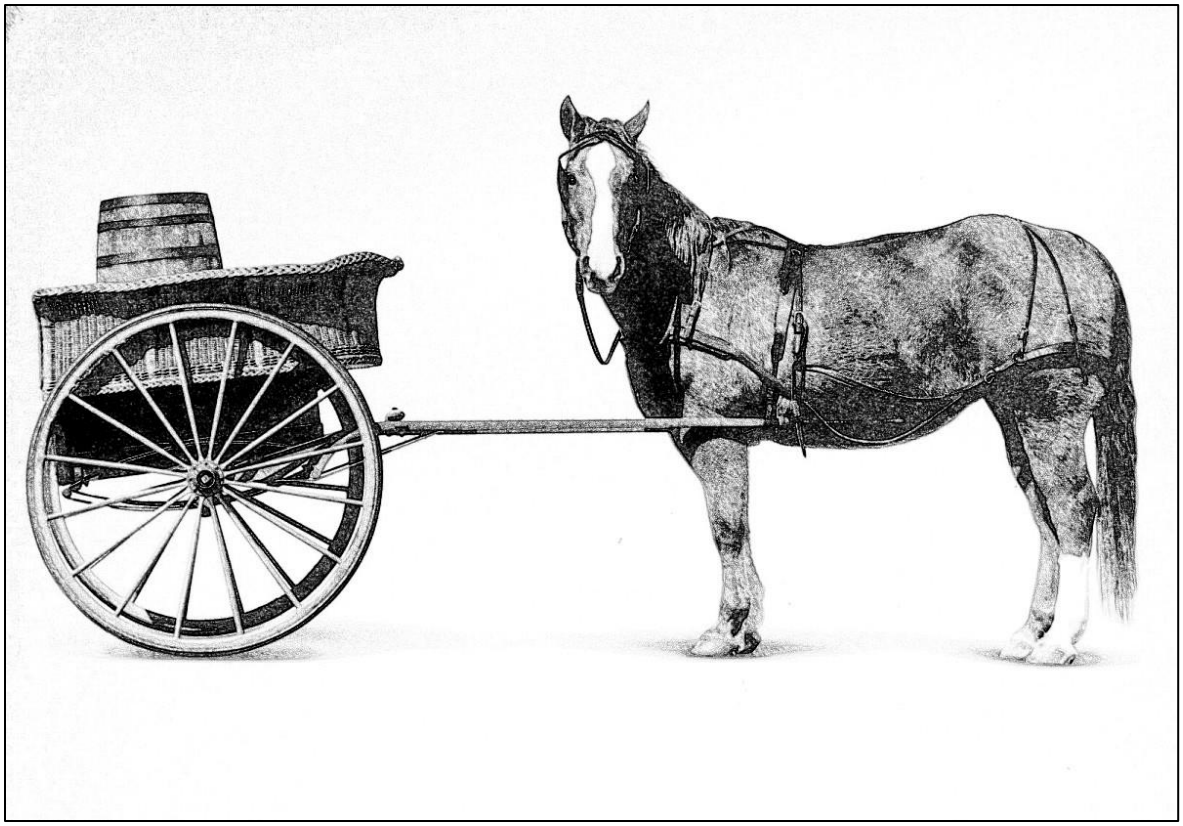

*Image adapted from “Cart before the Horse”[3]

Things to consider first (or putting the horse before the cart)

Do all parties involved understand the following questions?

- What is Test Automation in the first place?
- Why do they want to implement it?
- What is to be automated?
- How best to do it?
- Who should be doing the work?
- Where the automation work should take place?
- When the automation work should take place?

Answer the questions above first, then set proper expectations based on them and communicate them clearly.

Why Automate?

As stated before, there is an expectation that automation can achieve efficiency gains and allow humans to focus on high-value tasks. Using tools in the different parts of the testing process helps to reach that goal. Why use Excel spreadsheets or an Access database for either Defect Management or Test Management? That is inefficient. Use a tool that is specifically built to aid in those particular areas. By getting efficiency gains, you can get cost containment and savings. Also, you can improve the communication and feedback loops on the project. The more efficient you can become and the quicker you can turn around information, the happier other groups, including management, will be. The result ultimately will be delivering software of higher value to the end-user.

What to Automate?

Define the levels of tests to be automated such as Code, API/Services, and GUI. Define the types of tests to be automated, these include:

- Unit Tests for the code itself using Statement & Branch coverage and Function Call-Called Pairs
- Integration (Services/API interfaces)
- Build Validation (more commonly called Smoke Tests)
- Regression (critical path and in-depth functionality)
- Business scenarios
- Fault Handling and Error Injection
- End-to-End
- User Acceptance Test (UAT).

These can exercise the user interface, reporting, database, middle layer, back-end layer and integration/interaction with other systems outside of the one currently under test or anything else that is part of the system itself.

Load & Performance tests, along with Security Testing, needs to be considered as part of an overall automation strategy, too. The specific details on these particular items are best to discuss outside the realm of this book. There are numerous other and more appropriate sources available for them. This book's focus is the automation of testing for "functional" test execution.

Who builds the Automation?

People who build automation come in different forms with varied backgrounds. The common perceptions are technicians in white lab coats, the nerd with black horn-rimmed glasses, the hacker sitting in a cube drinking Red Bull and Mountain Dew while eating pizza, and even magicians. A lot of times though the magicians look like Seamus Finnigan, the character from the Harry Potter movies, when an experiment has blown up in his face.

In reality, people who build test automation end up looking like Coal Miners at the end of the day, all-in-all it is a dirty job. You have to get your hands into the application and work hard to get the tool to do what you want. The designing, writing, and testing of automation code is not easy. And can be especially true when your goal is to have reliable, robust, reusable and maintainable

automation code. The person building the automation code has to have the skills of both developer and tester along with the experience of the work.

The skills sets of these people vary, but a couple of core ones include: being able to program in the tool & language of choice and understanding how to get the tool to perform the actions of a test. These actions include navigation, data input/output, validation and verification of data and condition/state of the application under test, useful reporting on test status and how to go back to a known base state. They understand the mechanics of performing a test and how to get the tool to do it correctly.

Another important one they should possess is communication skills, both written and verbal. Being able to communicate effectively with management, development, business analyst, and testing groups is vital. They need to be able to gather information they need to do their job, but also to let people know what they are doing and why. They will collaborate tightly with both development and test to get the job done.

How to Automate?

How to implement automation is dependent upon many factors and has many implications. How is dependent upon the answer to the 'What' question. Meaning what level(s) to automate at and what types of tests to automate and against what kind of technology stack you are automating. Answering those previous questions will have an influence upon which tools and skill set you will need to employ on the project. There are many different approaches and methods associated with automation. For this book to go into detail on each one isn't practical. Other books and information sources that cover them specifically are more appropriate. This book takes a higher level approach to the overall process of how to implement automation.

Where to Automate?

Where to start implementing automation is determined by risk and need for the project. Will it happen at the Unit Test level or API/Services level or GUI/Acceptance Test level? That needs to be determined as soon as possible. You'll need to balance the desire for quickness versus sustainability over the long term. Be sure to balance across all levels of testing and don't allow one type or level to become lopsided. Also, the categories of tests to automate need to be determined early on. Is the focus on Smoke/Build Validation tests first or something else? You will need to be more focused (surgical) than broad (shotgun) in your approach and strategy.

Another category of this question is where physically the implementation will take place. Will you use test staff personal machines to build and run the tests, or will you build out a physical lab or a virtualized one, or will you go with a hosted environment? Determining this early on will help with scheduling and budgeting.

When to Automate?

As part of what some people are calling "Shift-Left" practice in relation to testing, in general, it's best to start the process as soon as possible. By doing so, you get the Developers and Management involved early and gain their support. It doesn't mean you start building the framework right off the bat, but that you get buy-in and support for the work as soon as possible.

Your job is to "sell" the automation implementation and start getting what you need sooner rather than later. Part of this is to dispel the myths (discussed in the next chapter) and correct misconceptions regarding automation and how to implement it correctly. Determining when will help you to keep from being painted into a corner. Early interaction and involvement will give you the opportunity to build credibility with other groups, and help you in the long run.

Planning to Not Fail

To support the implementation process, you need to outline your strategy and schedule its tasks accordingly. Implementation of a project plan and budget comes into play here. It will help to keep you on track. A project plan will prevent other people from sidetracking you by showing what you are trying to do and by when. And it is especially true for management because they want to know how you are spending their money. Remember, time is money too.

Implementation Plan

What is it? It is the game plan that explains the Who, What, When, Where and How for the automation stream within a project. It can be a detailed document, outline, schedule or a combination of them all. It is a roadmap to follow during the implementation journey, and beyond. Why create one? To better organize and manage the tasks related to this line of work. You can't do everything all at the same time; implementation is a distance race, not a sprint. Who uses it? The automation project team primarily, and other outside groups as well. It is a needed communication tool and will keep everyone informed, so you don't go "dark."

The plan needs to comprise 3 or 4 main phases. These include:

- Initial planning and discovery
  - Develop the project plan itself
  - Interview the appropriate stakeholders to get & set their expectations
  - Review test cases for appropriateness and clarity
  - Reviewing the system for "Testability"
- Implementation
- Knowledge Transfer and Training
- Maintenance and Improvement

Use a Proof of Concept during the review for testability sub-phase to both evaluate the target application under test and select tools to be used with it. Do this to ensure you have the correct tool, or set of tools, in place to do the work ahead of you. The outputs from this phase include a detailed schedule and budget. These will be needed to show management what is to be done and the associated costs for the initial implementation.

The next phase is the actual implementation itself. Within this, you will set up your test environment where the automation will be built and run, implement the tool(s), construct the framework for the automation, and begin the initial test automation development. After this phase, the output should be an initial set of automated tests that can be used for build validation, or better known as Smoke Tests. By doing so, you produce something useful for the project and can show an initial return on investment by creating a way to get feedback more quickly to the development team, and management, regarding the health of the software. It is a great little win and can help to further the automation effort.

The third phase is knowledge transfer for other staff; you want to ensure that other people can understand and maintain the automation code as implemented. The fourth phase is the maintenance and improvement of the initial work.

Building these last two phases into the project plan provides the time to do the training and the maintenance work. Otherwise, the automation may become shelf-ware very quickly.

Automation is a form of software development unto itself. The process never stops, and there is always a need for improvement. Implementing automation in testing means creating "Testware," and as such it will involve following the same practices of software development regarding maintenance and enhancements to those assets.

Now that the tasks and schedule are defined the next step is to get funding.

Implementation Budget

It is the finances behind the game plan. Nothing is free; there is always a cost associated. Even the "free" open-source tools have a price which mainly comes in the time and effort to build out functionality and capabilities similar to a commercial tool. Thus, causing a more extended startup period for the project and company. As was said before, automation is a software development project unto itself and needs proper funding. Creating a budget allows you to show the costs. Management, the money people, is going to ask for it before they will give you a single penny. So be prepared to explain it and defend it. It gives everyone involved an idea of the type of investment needed and why. It is used throughout the work by the Test Manager and Automation Lead along with management. Within the budget, you need to show the cost of staff, equipment, and software required to get the implementation done. Be sure to give a couple of examples of the types of teams to be used to show the potential differences in cost. Do this for a group of all full-time employees (FTE), a team of consultants/contractors only, or a mix of them. Always include the hardware & software costs with each group for a total amount. Examples of high-level budgets are shown below.

Overall Cost – All FTE Team

Item	**Type**	**Quantity**	**Cost**
Automation Architect	FTE	1	$145,000.00
Sr. Automation Developer	FTE	1	$133,000.00
Automation Developer	FTE	2	$232,000.00
Laptop	Resource	4	$8,400.00
Virtualization Server (10 VM's)	Resource	1	$8,000.00
Automation Tool License	Resource	10	$85,000.00
Total			**$611,800.00**

Overall Cost – Mixed Team

Item	Type	Quantity	Cost
Automation Architect	Contract	1	$192,000.00
Sr. Automation Developer	FTE	1	$133,000.00
Automation Developer	Contract	1	$153,000.00
Automation Developer	FTE	1	$116,000.00
Laptop	Resource	4	$8,400.00
Virtualization Server (10 VM's)	Resource	1	$8,000.00
Automation Tool License	Resource	10	$85,000.00
Total			**$696,400.00**

Notice the staff costs are for a yearly amount, and that it is the most significant part of the overall budget. Machines and software are cheap in the grand scheme of things. Something management needs to see clearly up front.

Breaking things down further in the following tables are the individual costs for both staff and other expenses.

Staffing Cost

Position – FTE	***Base Cost – Salary***	***Benefits***	***Total***
Automation Architect	$125,000	16%	$145,000.00
Sr. Automation Developer	$115,000	16%	$133,000.00
Automation Developer	$100,000	16%	$116,000.00
Position – Contract (Commercial Tool)	***Base Cost – Contract***	***Benefits***	***Total***
Automation Architect	$100/hr	0%	$192,000.00
Sr. Automation Developer	$90/hr	0%	$172,800.00
Automation Developer	$80/hr	0%	$153,600.00
Position – Contract (Open-source Tool)	***Base Cost – Contract***	***Benefits***	***Total***
Automation Architect	$125/hr	0%	$240,000.00
Sr. Automation Developer	$110/hr	0%	$211,200.00
Automation Developer	$100/hr	0%	$192,000.00

Equipment Cost

Item	***Price***
Laptop – 15.6", i7, 500 GB HD, 8 GB RAM, Windows 10 Pro	$1,000.00
MS Office Pro	$325.00
Keyboard & Mouse	$75.00
Docking Station	$225.00
Monitor – 27 inch LED (x2)	$400.00
Total	$2,025.00
Automation Tool – Concurrent License	$8,500.00
Overall Total per Workstation	**$10,525.00**
Virtualization Server (10 VM's supported)	$8,000.00

Summary

As you can see there is a lot to consider up-front when implementing test automation; you do not just grab a tool and go for it. You'll have to understand the Who, What, When, Where, and Why questions along with the How to set yourself up for success. There is a real need to make sure you understand the “costs," primarily a monetary one, which comes with the work. But also the effects it can have on the testing effort along with the overall software development process. It means a change in the skills and knowledge of test staff along with others on the project. It means making sure other groups, especially management, understand them as well. Mapping things out and planning will go a long way on the road of success. Doing this helps to ensure you'll be part of the small percentage of first-time implementations that don’t fail.

As my friend Paul Grizzaffi says “With Great Judgment Comes Great Responsibility”[4]. Meaning you will have the responsibility, and accountability, to others as well as yourself in the judgments and decisions you make regarding test automation. And what was discussed here in this chapter is just scratching the tip of the iceberg, there is a lot more we will explore in this book.

References

1) "Automation," Merriam Webster Dictionary online, www.merriam-webster.com/dictionary/automation.
2) "Automation in Testing," Richard Bradshaw, Automation in Testing, automationintesting.com/about/, 2018.
3) https://www.phrases.org.uk/images/cart-before-the-horse.jpg
4) “With Great Judgment Comes Great Responsibility," Paul Grizzaffi, Software Test Professional Conference Keynote, September 28th, 2017.

CHAPTER 2 - AUTOMAGIC

Automation Myths & Misconceptions

At this time it is worthwhile to discuss the many myths and misconceptions regarding automation in testing. Some old ones keep popping up, and some new ones that make a person stop and slap their forehead in disgust.

Let's start with some of the oldies but goodies.

- *You can automate 100% of your tests*.
- *Automation will "solve" all problems with testing*.
- *Record & Playback is all you need*.
- *Automation allows Management to reduce test staff*.
- *One tool is all you need*.
- *Automation will speed up the testing process*.

Now for some of the new myths that have appeared over the last few years.

- *You don't have to know how to program*.
- *Anyone (untrained/unskilled) can do automation*.
- *Open-source tools are "free*."
- *Commercial tools are "expensive*."
- *GUI level automation is a waste of time*.
- *"Codeless" and "Scriptless" tools* can do all the work.
- *Automation can be created quickly*.
- *Commercial tools can't support agile*.
- *Developers only need to focus on Unit Tests*.
- *Only hire a Software Design/Development Engineer in Test (SDET)*.
- *Automation provides an immediate return on investment (ROI)*.

So how do we go about addressing some of these? Let's go back over the previous items listed above.

- You can automate 100% of your tests – There are two parts to this.
  - First, some tests cannot or should not be automated. An example is a test that needs to determine if a color is correct on the screen, another is testing for intuitive navigation through the application itself. Because of scenarios like these automation is blind to a large degree. Automation scripts become complicated to create and are unreliable for these types of scenarios. Even tools that claim to be visually based will have issues and can produce flaky tests.
  - Second, because of time limitations, it is an impossible task to try to automate all tests. By applying the 80/20 Rule (Pareto Principle[1]), automating the 20% of functionality used 80% of the time keeps the focus on what is important. To help with this process use functionality maps, categorization and prioritization, and critical

path as criteria to narrow the scope to stay focused on what is essential. Another method to consider is User Community Modeling Language (UCML™)[2]. It is a way of modeling usage of the system under test. Its creator, Scott Barber, stated it is a method "…performance testers often employ to determine what activities are to be included in a test and with what frequency they'll occur." This method of determining usage and frequency level can be a great aid in determining what falls in the 20% used 80% of the time.

- Automation will "solve" all problems with testing – It helps, but it creates new problems too. Specifically around reuse and maintenance. Automation isn't a silver bullet, and we're not hunting werewolves, so we don't need them. Some people put too much trust in it as a way to solve testing problems, and project level problems as well, that are beyond the operational level alone.
- Record/Playback is all you need – Proven time & time again to be false, and to fail. That was quickly shot down because of the brittleness of strictly recorded scripts and issues with hard-coded data and difficulties with maintenance. For building quick skeleton code or capturing objects it can be of benefit, but don't rely on it exclusively. Also, if you know the test is "throwaway," it might make sense to use it. But never use it as a long-term solution.
- One tool is all you need – "There isn't a one-size fits all tool." No tool can solve every problem in every situation. Just like a surgeon will use different instruments during surgery a tester will use various tools to evaluate software. You need a tool or set of tools that are appropriate for the job at hand.
- Automation allows Management to reduce test staff - You still need the human and their brain to determine what to automate, how to translate it into automation, to analyze the results from the test run and then act on them. Additionally, there is still the Exploratory Testing work that must be done by humans. Automation allows your testers to focus on that type of work, and to find the bugs that automation cannot.
- Automation will speed up the testing process – This is what I like to call the "Illusion of Speed" problem. Yes, a computer may execute a test quicker, but the effort to get to that point and then afterward is all made by a human, and that takes time. You can get a perceived increase in speed by running the automated tests in parallel in a distributed fashion. But the cumulative effort and time are still the same. Even automation is dependent upon synchronization and other timing issues in software, or it may require latencies (delays/waits put in them for various reasons) to keep the machine from getting ahead of itself. Also, a large number of automated scripts will not run extremely fast if there are dependencies upon run order or data or a combination of them. And if the workload stack is only run on a couple, or single, machine, you are throttled by the processing capacity of the setup. Put the workload stack on its side and divide up the workload and then distribute to multiple machines to run in parallel. Use "Economies of Scale" to get "speed." You get efficiencies in execution by having multiple machines and running tests in parallel (and eventually unattended & off-hours). In a later chapter, setting up distributed test execution via automation will be discussed.
- You don't need to know how to program – Anything you do with a computer involves some form of programming. You are giving the machine "instructions" to do its work. You need to understand process and logic to instruct a computer to do its job. Artificial Intelligence for testing doesn't exist yet, and if it did you better hope it doesn't become self-aware and start Judgment Day.

- Anyone (untrained/unskilled) can do automation –Not everyone has the 'mindset,' it is both learned and inherent (logic). Not anyone can be a doctor or lawyer. Automation requires skills, knowledge, and experience. It is a craft and necessitates people with craftsmanship.
- Open-source tools are "free" – In reality, no they are not. There are still costs due to ramp up time and employing skilled staff to implement it.[3] Initially the cost of the tool itself is zero, but the overall cost to implement and use them can be quite significant due to time to build the framework and its functionality to match commercial tools. And the price of staff can be higher due to the scarcity of resource or skills to do the work which can result in significant hidden costs.
- Commercial tools are "expensive" – Not really in the grand scheme of things.[3] Yes, there is the initial license cost, but you get a tool that has all of the needed functionality built-in. This allows you to jumpstart the automation implementation without having to string disparate open-source tools together to provide the same capabilities as a commercial tool.
- GUI level automation is a waste of time - I don't think so, and in some cases, it is quite beneficial to use. Business scenario & End-to-End tests are created this way. Also, if you follow the Automation Pyramid model, you still have some GUI tests to build. For example, if GUI automation should only be 10% of your automation effort and you have automated 10,000 tests that means you will have 1,000 GUI tests. Even the percentages attached to the Automation Pyramid are a misconception and will be talked about in more detail in Chapter 3.
- "Codeless" and "Scriptless" tools can do all the work - It is just a different type of UI for the end-user. It is adding on a layer of abstraction on top of the actual code. There is still "code" running underneath it all. In some instances, you will need to use the tools "code" module to create custom code to perform a task. So in some way, shape or form, there is going to be code. And when you build a "test" in these tools, you are creating "code," because the code is a set of instructions for the machine to use to perform a task. And this requires "programming" to do this. So again, you need to understand the mechanics of programming to get the job done correctly. Finally, all of the programming and code gets stored in a "script" in some form.
- Automation can be quickly created – A lot of time is required to write/create an automated test initially, and then additional time to maintain it. Expect a higher initial time investment than in comparison to manual testing. The investment will pay back over time, but it is not instantaneous. Because automation is its own form of software development, it will be constrained by the same rules as most other types of application development work.
- Commercial tools can't support agile - It isn't the tool that does agile, it is the person using it.
- Developers only need to focus on Unit Tests – In the Automation Pyramid you have Services and API tests, and the tools that drive this are more technical and require someone to build interfaces for them. Developers are best suited to this job or a skilled tester with programming skills/knowledge who can do the same. In the end, it comes down to finding the right person for the job.
- Only hire a Software Design/Development Engineer in Test (SDET)[4,5] is all you need - This one is a pendulum swing to the other extreme so to speak. If you only hire an SDET, you run the risk of having people who do not have the experience and knowledge of testing

that is needed to be successful in automating testing itself. Harry Robinson stated in regards to SDET's and Model-Based Testing (MBT)[6] "A particularly odd problem is that many people who end up in test teams have not had the opportunity to study software testing." And paraphrasing a couple of other points he made Testers do a different kind of programming than regular developers. It is a much simpler form of code, algorithms and data structures. This difference in focus is beneficial as "testers are less susceptible to common mode failure." Pradeep Soundararajan states "…a lot of fresh talent out of college is being hired to create the overgrowing need of SDET. Most of these SDETs will know a lot of programming and scripting instead of testing. In a few years, we will have an overload of people writing script who don't know about testing but dictating how testing will be done in organizations."[7] Therefore, there needs to be a balance of roles (SDET & Automation Developer/Tester) in place to keep things in equilibrium. Remember that too much of one thing isn't always the best to have.

- Automation provides an immediate return on investment (ROI) - This is the typical hyperbole by a Salesperson or Consultant. The reality of the situation is this whole thing will take time. If you follow the common practices of software development to implement automation you'll see it is going to take time. Implementing automation is like a marathon and not a sprint. Proper planning, pacing, and patience over the long run will have payoffs in different ways. There are ways to get an immediate ROI with quick wins like constructing the Build Validation/Smoke Tests first and incorporating them into the Continuous Integration process. But, it is unrealistic to think automation of testing will occur overnight by magic.

Summary

So now you see there are many myths and misconceptions to deal with even before writing one line of code. Addressing these up front will allow you to open lines of communication and change perceptions with different groups. Doing so will help to keep you from getting painted into a corner and having to live up to unrealistic expectations. And this will be especially true when it comes to educating management because it is your job to do so and to help them protect their investment in automation in testing.

References

1) “Pareto Principle”, Juran’s Quality Control Handbook, Seventh Edition, Joseph M. Juran, 2016.
2) “User Community Modeling Language (UCML™)," Scott Barber, www.perftestplus.com/articles/ucml.pdf, 2003.
3) "Test Automation - Let's Talk Business," Igor Gershovich, Connected Testing, 2006.
4) “Software Development Engineer in Test (SDET)," Microsoft, blogs.msdn.microsoft.com/seliot/2010/04/18/what-is-an-sdet/, 2010.
5) “Software Design Engineer in Test (SDET)," Microsoft, blogs.msdn.microsoft.com/chappell/2004/10/06/difference-between-ste-and-sdet-roles-at-microsoft/, 2004.
6) "Interview with Harry Robinson," Model-Based Testing Community, http://model-based-testing.info/2012/03/12/interview-with-harry-robinson/comment-page-1/, October 2011.
7) “The rise of SDET and related problems in Software Testing," Moolya, moolya.com/sdet-software-testing-challenges/, February.

Additional Resources

1) "Seven Steps to Test Automation Success," Bret Pettichord, www.io.com/~wazmo/papers/seven_steps.html, 2001.
2) "Success with Test Automation," Bret Pettichord, www.io.com/~wazmo/succpap.htm, 2001.
3) “Test Automation Snake Oil," James Bach, www.satisfice.com/articles/test_automation_snake_oil.pdf, 1996.

This page intentionally left blank.