TESTFORT

Case-Driven Automated Testing Guide

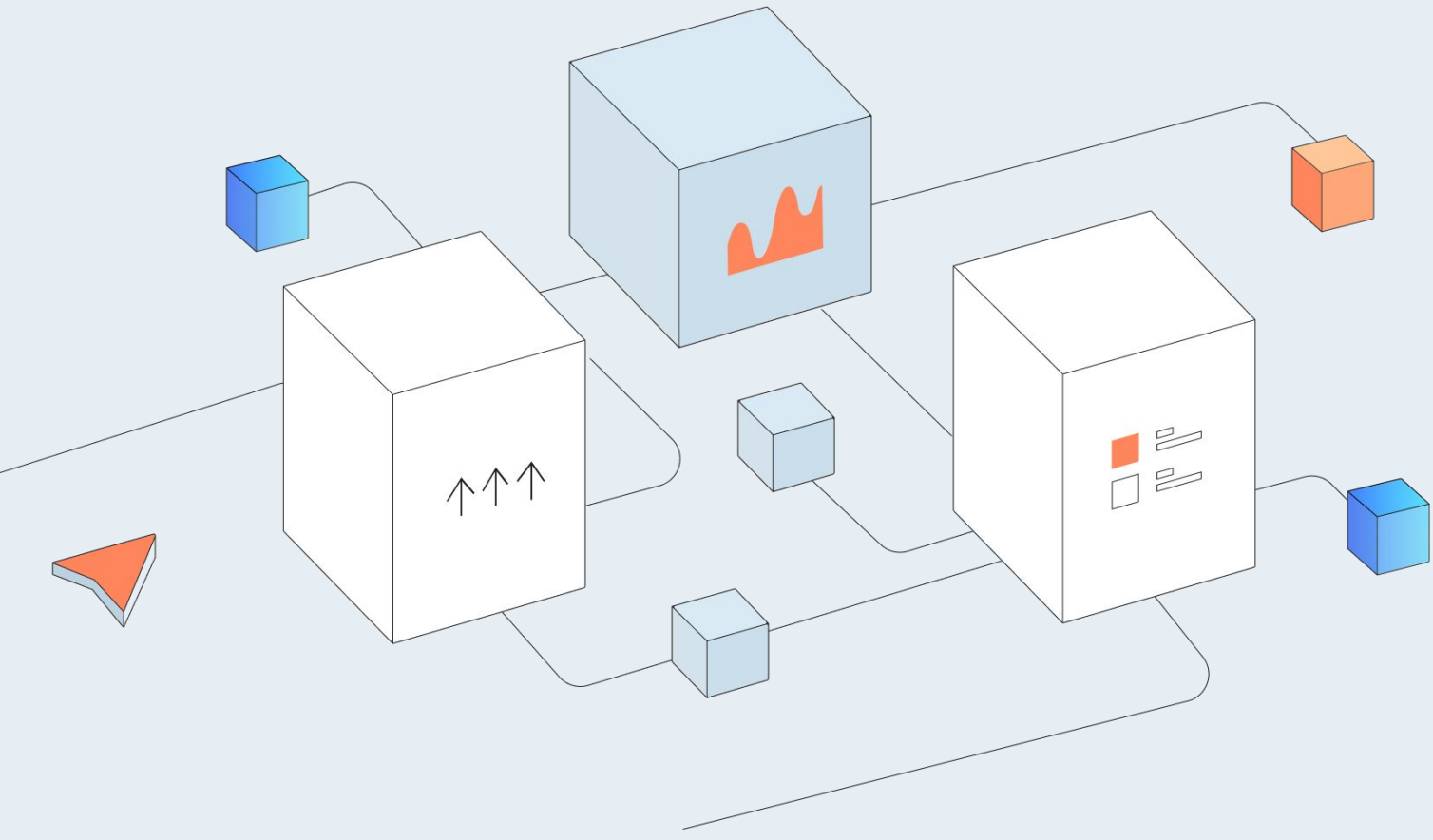


Table of Contents

01. Why bother with automated testing (advantages and trends)?

02. Choosing a Team

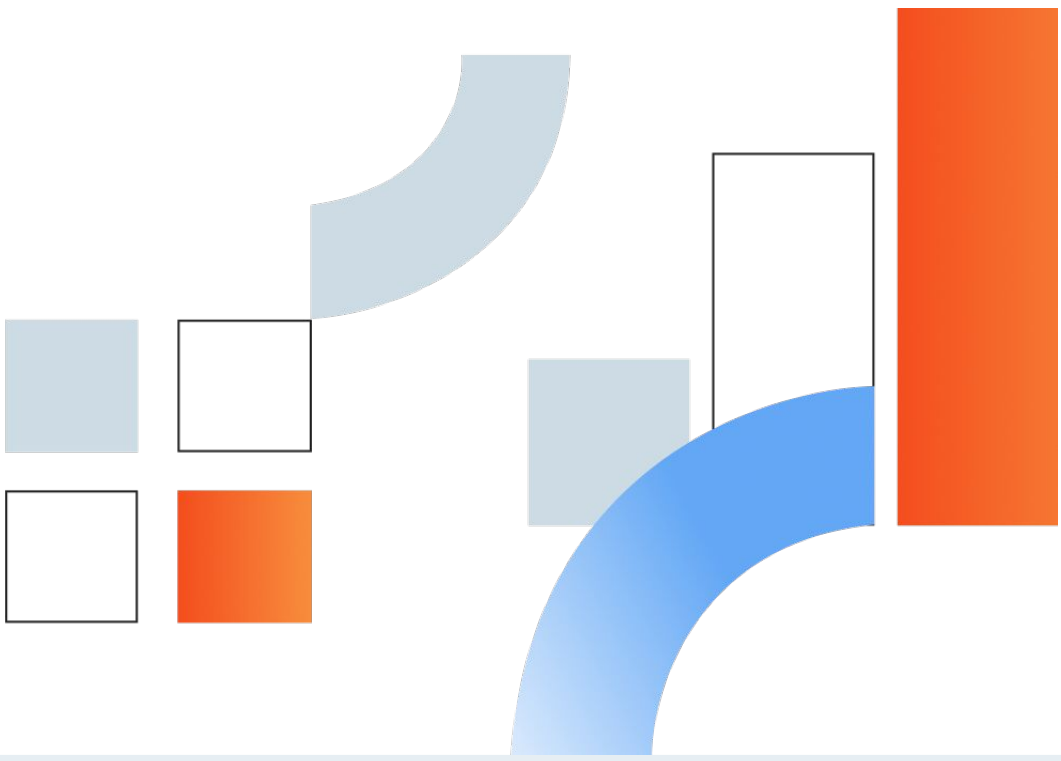
03. Developing a Strategy & Setting Objectives

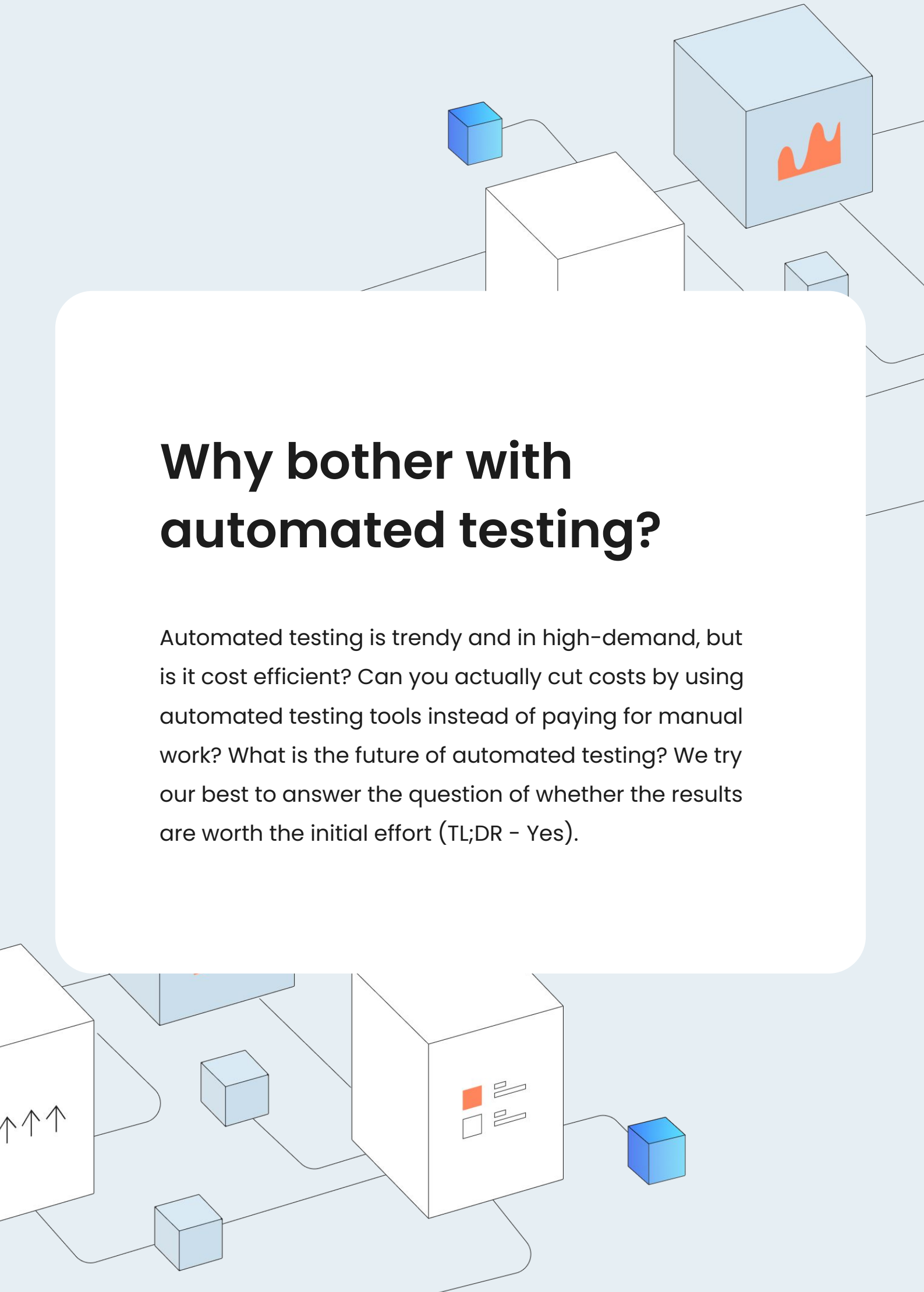
04. Choosing the Right Tools + Netflix Case Study

05. Challenges & Problem Solving

06. Stories of Success and Failure

07. Conclusion: Automated Testing Checklist





Why bother with automated testing?

Automated testing is trendy and in high-demand, but is it cost efficient? Can you actually cut costs by using automated testing tools instead of paying for manual work? What is the future of automated testing? We try our best to answer the question of whether the results are worth the initial effort (TL;DR - Yes).

A manual tester works 8 hours and goes home. Automated testing works all the time. But does it actually work?
We are not going to say that automated testing is or should be your number-one option. It doesn't work for each project, and anyone telling you otherwise is simply wrong. Human intuition is often everything since there are complex user scenarios to evaluate and predict.
Let's cross these intuitive types of testing of our list right away, before we get started with building a strategy and assembling a team.

Automated testing DOESN'T WORK for:

Interface evaluation

Robots are great, but there is a long way to go before algorithms reach the level of understanding high enough to determine whether something looks 'pretty'. Same goes for things like haptic feedback or sounds.
Strategic Development and User Patterns
When we create software for humans, it's only humans who can determine user behavior. How to predict what button a user will press? Will they find it in the first place? Algorithms might be smart, but they lack intuition. Humans know humans best.

Testing complex scenarios

It's not that it's impossible to write an automated test for complex user scenario, but even if you could, you really shouldn't. Teaching human logic in a complex action to a machine is a long and tiresome process. Al and machine learning isn't quite there yet.
Robots are no humans, and often it is a disadvantage. Not all the time though. Algorithms don't
stress out, never get bored, or lose focus. If a test case requires a number of dull repetitive actions,
wasting a perfectly good team of manual testers is unreasonable.
That being said, let's see where automated testing is not only helpful but absolutely necessary.

Automated testing WORKS for:

Regression testing

You know that situation when only one piece of code is changed but the functionality stops working altogether? To avoid that 'this-shouldn't-have-happened' situation, regression testing is performed. You verify that the system works fine after previously made edits, software patches, or setting changes. It's repetitive work which can and should be done by machines, not humans.

Functional Testing

Here testing team's task is to evaluate whether every software function works. It's a routine task, but a crucial one - every little operation needs to be performed, assessed and documented (sometimes multiple times).

GUI Functional Testing

It's not about evaluating the attractiveness of the interface but only about its functionality. Algorithms will assess whether GUI objects work the way they were designed to.

Smoke testing

Smoke testing is a type of functionality testing that verifies and assesses the most important system functionality. Test cases analyze main user scenarios that necessarily will be performed by actual users.
Every software is like a building. There are cornerstones which work as the foundation for the entire architecture. If these are broken, there is no point in moving on to smaller errors. That's why smoke testing is the first to be done.

Load and performance testing

This is the stage where you prepare your project for success. If it gets popular, you better be sure the product will survive the traffic load. For that, the automated testing team writes tests to create an environment as if 10,000, 100,000, 1,000,000 people are using the service at the same time.

Let's answer the most important question. Why automate? Couldn't it all just be performed by a bunch of manual testers?

Just because it could be, doesn't mean it should be. The golden rule of all tech progress ever achieved says: what could be automated, shouldn't be done manually. That's the reason we have technology in the first place. Why bore humans with dull repetitive actions if there are dozens of tools and frameworks to do the work for them?

Imagine yourself being a manual functional (for instance) tester Every day, you have dozens of functions to run, assess, and fix what went wrong. Hundreds of bugs, 8 hours a day, every day. The same bugs. The same functions. At this point, I wouldn't blame you if you started looking for the easiest path, the one requiring the least amount of effort on your part. I also wouldn't blame you if, from time

Algorithms, luckily, do not get tired or bored. This is what makes them not just a tech-savvy innovation but a necessity.

Will automated testing replace manual testing?

Not fully but likely, in most testing fields, teams will switch to automation. When you work with technology on daily basis, automation is a logical thing to do. Also, the number of testing market and testing cases grows every year.

According to Statista.com, in 2019, the proportion of budget allocated to quality assurances and testing as a percentage of IT spend will top 40%. Just for comparison, in 2013, it was only 18%.

First-chapter checklist

- [ ] Not every type of testing can be automated. If you hear or read it somewhere, don't buy it. There are complex user scenarios which only humans can create, assess, and understand. These are Interface Evaluation, Strategic Development, and User Patterns, or assessment of complex test cases.

- [ ] There are types of testing where automated testing is absolutely necessary such as Regression testing, Functional testing, GUI Functional testing, Smoke testing, Load and Performance testing.

- [ ] Automated testing is not a trend, it's a reality. With a steady growth of demand on QA, testers will switch to algorithms instead of performing manual work.

- [ ] Automated testing not only saves resources and money but also improves the work quality. When human testers perform repeated actions day to day, they get bored and stressed. The product is hardly better for it.

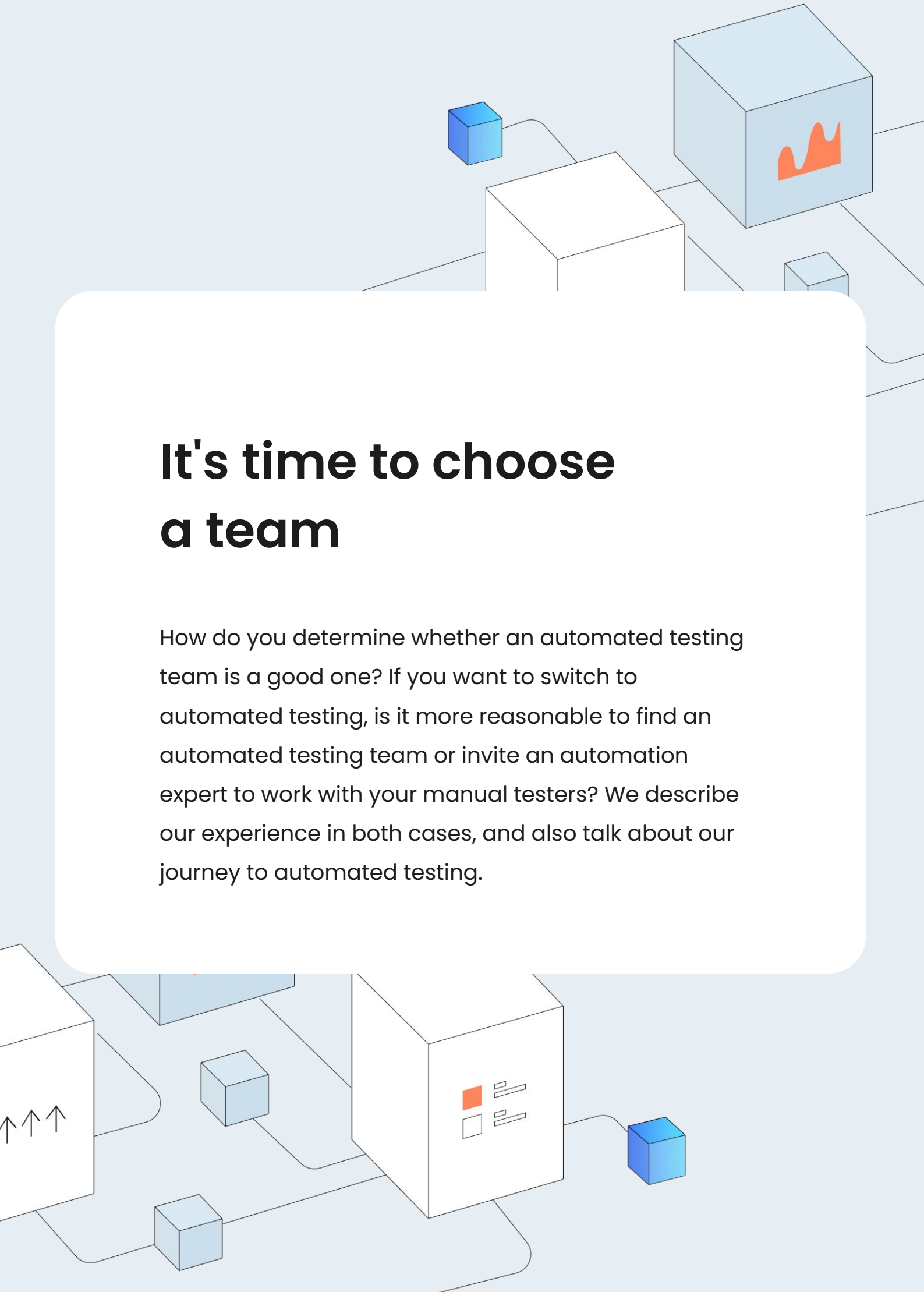
- [ ] Automated testing won't fully replace manual but automation will surpass manual work. The earlier you start implementing automation, the faster you'll adapt to new demands of the technology market.

Three takeaways from the chapter:

- [x] Take our list of types of testing that absolutely have to be automated and analyze your expenses on the manual testing. Sum it up — it'll help you to compare your possible expenses for automated testing to your current ones as we move further.

- [x] Look at the testing reports and glance through test cases your manual testers deal with. Find the ones that you think could be automated. Later we'll see how.

- [x] Analyze the types of testing that couldn't be automated. How often do you perform them? How many manual testers do you need for the process? Later we'll talk how manual and automated testers will cooperate so you will definitely need this data.



It's time to choose a team

How do you determine whether an automated testing team is a good one? If you want to switch to automated testing, is it more reasonable to find an automated testing team or invite an automation expert to work with your manual testers? We describe our experience in both cases, and also talk about our journey to automated testing.

A team is what makes difference between a great test automation and a poor one. No matter how you build a right strategy or set the objectives, it's them who will do the work.

When choosing automated testers, you have two routes to go

Route #1 — Educating your manual testers

We always like to work with people we know and trust. People who already know the project and have worked with the code are faster, don't ask redundant questions, and already believe in the product. There is only one issue: what if they have no experience in automation?

When to go this way?

As far as education usually goes, we all know it requires time and/or. Therefore, before making any decisions, think whether the investment won't be wasted. As we've seen from our own experience, going this way is reasonable if we talk about an in-house team. In this case, you are likely to benefit from their skills a lot more often than just this once.

If it's a remote team, making this investment is a huge risk. It might be best to find another team of experienced automated testers (we'll get to this later in the chapter).

Also, Route #1 is a long-run solution. If you need automated testing right here and right now —opt for hiring another team, dedicated specifically to automation. However, as a long-term investment, educating is truly reasonable.

The problem

You have a good team of manual testers. Maybe, these are in-house testers or an outsourcing team you've been working with for a long time. They know the project and the product, you have good communication and a well-designed management process. However, they have no automated testing background whatsoever. What to do?

Set a goal

You don't need all your manual testers to receive automation skills. Choose a few testers of the team and focus on giving them all the necessary knowledge.

How to start?

Before making investments, prepare the team. After all, it's them who will master a new skill and it's also their careers we are talking about. Therefore, take into account the following aspects:

Whole team commitment

Make sure you are not the only person interested in the process. You are about to give your testers new skills and knowledge, and they should be as excited about it, as you are.

Talk to the team, explain that learning process is a huge investment from both sides, and while you are going to provide all the resources, they have to put up extra hours and work longer days.

Assessing possibilities

Don't aim to give each of your manual testers the extensive automation background. You can form a team of volunteers who are willing to stay up late writing unit tests and experimenting with frameworks. You don't need every tester to be on board.

Interest in automation

Don't confuse this with commitment. Testers can be very committed to the team and its goals but that doesn't mean a good relationship with automation. Some people just don't have it in them — and it's okay. Talk to project manager and testers themselves. Maybe you'll find out some of them are happy working only manually.

You'll need a supervisor

The easiest and best-proven way to switch from manual to automated testing is to invite an automation expert and have them oversee the process. Invite an experienced automated tester and discuss the following issues:

- **Goals and strategy.** Most likely, you already have a project where you need automated testing skills. Invite your ‘supervisor’ to take a look at it and define most pressing issues.
- **Necessary instruments.** To switch to automation, you need tools and frameworks. Instead of experimenting with it on your own, better use the advice of an experienced automated tester.
- **Cost.** Switching to automation will take time and resources (you'll need a bunch of software and hardware to set up the environment as well as frameworks and drivers to automate the tests) so discuss it beforehand.
- **Management details.** Most likely, you can't freeze all your projects and tasks just so testers can learn. You still have to take care of many pressing issues and waiting is not an option. That's why we recommend to start the switching process with a few testers, then gradually increasing their number.

Develop the right metrics

How do you know the team is going the right way? How to determine whether all these efforts and investments really improve the process? There is only one way to know for sure and that is building metrics.

Metric ff1 — Test count
Quantity is important but it doesn't stand for quality. Control the interaction of every test. By the end of every week, you should get a report with color-tagged test results (green tag if everything went well, red — if a test failed).

Metric ff3 — Time and resource management
Evaluating the results is great but don't forget to compare them to expenses. How much you gained compared to what was spent? Is this number higher? Is there a chance it'll increase over time? If the answer to these questions is yes, you are going the right way.

Case in point: Microsoft Azure Automated Testing Strategy
Article “Software testing at scale to increase velocity” on Microsoft official blog describes the algorithms company used to minimize software testing expenses, including beforehand planning, measuring results and comparing outcomes. To minimize risks, the team maintained the comprehensive comparative report, using the criteria we described above.

Route #2 — Find an automated testing team

Turning manual testers to automated ones is not always reasonable. If you have one project that you'd like to automate, it's much easier to choose a 'ready-to-use' team.

However, if in the previous case you cooperate with testers you already know and trust, here you have the entire getting-to-know-each-other journey ahead. That's why it's crucial to put a lot of consideration before opting for an automated testing team.

These are 5 takeaways for picking an automated testing team. All of these insights were proven by our cases and experience, some of them — by mistakes and failures. To avoid those, learn our lessons.

Takeaway #1 — Hire a team with Automation Architect

A dedicated team needs guidance and control. For that, hire a team with 1-2 architects. They will be responsible for managing the process, building the strategy and evaluating the results. Even if you plan on hiring a small team of 5 testers, having an experienced automation architect makes a real difference.

Takeaway #2 — Look for a team with software development expertise

The core difference between manual and automated testing is that automation is almost software development. To design tools and frameworks, you need to follow best development practices, including regular code reviews, pattern design, constant support, and maintenance. Make sure your team knows and uses best development practices and applies them in automation.

Takeaway #3— Look for a team with software development expertise

Talking about software automation, we discuss the importance of the right instruments. The truth is, no tool is perfect. It might glitch and miss bugs, and it might not know an important for your project scenario. Every tool should be adapted specifically to your project and that is something your team will take care of. That's just another reason of why investing in testers matters more than frameworks and instruments.

Takeaway #4— Select a tool considering testers' capacities and your project's needs

If your team knows Java, they'll be more comfortable working with a tool that offers Java to write scripts. However, if your project is powered by C++, better pick a team that knows C++.

Takeaway #5— Product comes first

The priorities in automated testing are: project — team — tools. You start by analyzing your product, choose a team that can solve problems, and pick tools these testers will be comfortable with. We always share our experience of working with tools with our clients, it makes the tool selection easier.

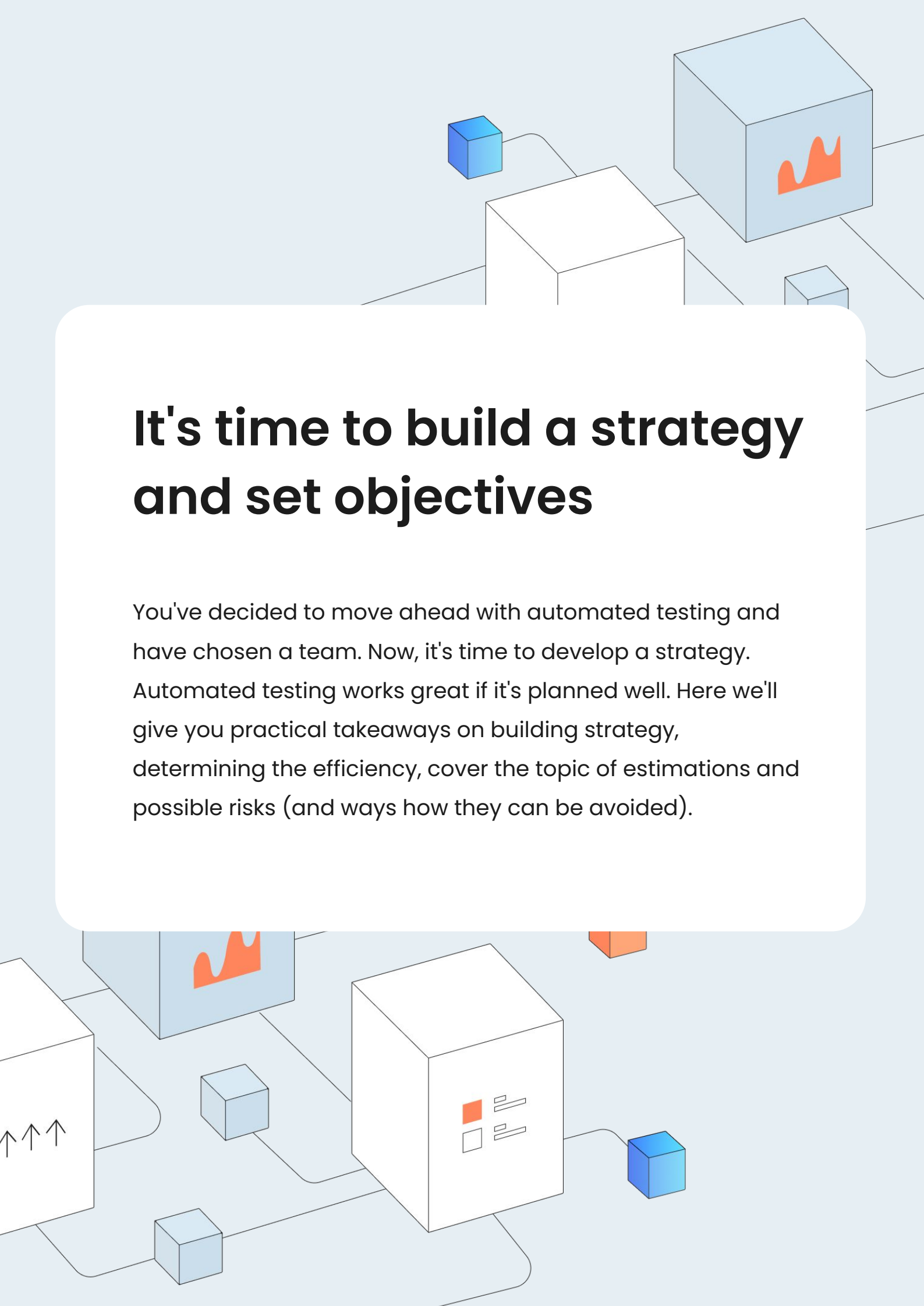
However, it all starts with a product. Only knowing it well, you can set the right goal, find capable testers, and choose tools. If it's a web page or an app, know what browsers you want it to support. If that's an application, think about functions and OSs. The more you know about technologies used in development, the easier it gets to build the testing strategy.

Second-chapter checklist

- [ ] You have two options of cooperation with an automated testing team: 1) educating manual testers; 2) partnering a dedicated team of automated testers. If you have a strong in-house team, opt for the first option. In all other cases, go for the second one.
- [ ] Educating testers is a time-consuming process but it pays off in a long-term perspective. While switching from manual testing to automation, have an expert to oversee the process.
- [ ] If you look for a 'ready-to-use' automated testing team, make sure their skills correspond to your product needs. For example, if your app is powered by C++, make sure testers know C++ as well.
- [ ] Check out rating and awards. This way, you minimize the risk of partnering an untrustworthy team.

Three takeaways from the chapter:

- What ratings to use to find a trustworthy partner? Good testing teams are constantly featured in Clutch and international testing awards
- Do rise management When partnering and outsourcing testing team, analyze such factors as location, time and cultural differences, security challenges, and remote communication.
- Don't automate 4-5 types of testing at once. Start with one (like regression testing) then gradually automate new scenarios.



It's time to build a strategy and set objectives

You've decided to move ahead with automated testing and have chosen a team. Now, it's time to develop a strategy. Automated testing works great if it's planned well. Here we'll give you practical takeaways on building strategy, determining the efficiency, cover the topic of estimations and possible risks (and ways how they can be avoided).

Congrats, you decided to automate your testing and even gathered the team (well, almost). Now, you all have another thing to figure out, something that determines your successes and setbacks. Strategy.

There is no result without planning. It's not about predicting everything but rather defining challenges and finding solutions.

Let's start from the beginning.

- What defines successful automated testing?
- Well-built management strategy and its implementation The right selections of team, tools, software, and hardware
- Communication between team members and leaders.

Management matters

Fish rots from its head and failures in automated testing come from bad management. Inadequate communications, objectives, deadlines — any of these can lead to big loses.

Automation Goals

To achieve something, know what to achieve. All these vague objectives like ‘we need cheaper and faster’ won't cut it. Set specific goals like ‘automate X tests’, ‘run Y tests successfully’. Set goals that are measurable.

The goals for testing and automation are not the same

When building a strategy for automated testing, you have two types of goals. The first one is about testing — you need tests to be run successfully and work through as many scenarios as possible.

The second goal is regarding automation.

Automation doesn't equal automated testing. Why? Because it's not the testing itself, it's the way of doing testing. Here you need to focus on how many tests you automated, what tools were used in the process, how much time did the entire process take.

Goals for Automated Testing	Goals for Automation
The ultimate result we want to achieve (like find X of bugs)	What do we need to achieve those ultimate results (what tests do we need to automate)?
Another ultimate result we want to achieve	How many, in what time frame, with what resources?

Also, set specific objectives for each type of testing. Goals like ‘find lots of bugs’ won't work, for example, with Regression Testing because that's not how this type of testing works. In regression testing, you don't look for bugs, you make sure previous changes didn't affect the entire system. In most cases, there will be little or no bugs found, and it doesn't mean the test is wrong, that's just the way it works.

Guide and teach

We already discussed how important it is to have a supervising architect who will oversee the process, check the results of performance, talk to developers and the product owner. However, there is more to guiding management than just finding an experienced architect.

Providing time and resources

Automated testing is not done in a day. If you have never run automated tests before, it means, you'll need to pick tools, find the right hardware (if you are working with the outsourcing team, make sure they have it), and, most importantly, give time to build an optimize the process.

Build and maintain automation regime

You need a system, a schedule with planned tasks and defined objectives. Plan every week of the testing process, determining what tests you'll automate during these seven days and what are the expected objectives. If the outcomes are successful, you move onto new types of testing or more complicated user scenarios. If they are not, communicate with team managers and find out what went wrong.

Talk about your product and business overall

Testing always stands together with marketing. Your testing team works for the end result and directly influences the way users will perceive the product. Make sure all team understands the end benefit: what will their actions change not only in the code or architecture but at the bigger scale? How will it make your business stronger? Also, these are questions to discuss with a team manager. If there is no clear answer, it might be just time to consider switching to another vendor.

Don't rush the result

Automated testing is (supposed to be) faster. Not at the beginning. Give your team time to structure the process and adapt tools. Don't forget that automated testing is, first of all, a long-term investment, not 'I want it for yesterday' kind of approach.

Measure investments, measure results

There is a word in the automated testing community that tools determine 90% or even 99% of automation success. It's a popular misconception — don't buy it. Taking an Open Source tool just because it's free is not how you get results.

Profit requires investments. In automated testing, investing means experimenting with tools and frameworks until you find the instrument that suits your product best. If you put nothing, you get nothing.

But if you put everything, this alone guarantees you nothing as well. To make sure investments won't get wasted, it's important to manage all expenses and profits. You already have goals, now you have to evaluate the results — and make sure they comply with objectives.

Rule #1 - Estimate ROI at the beginning of automation
How to know whether you save more resources than you spend? The profit is usually calculated by hours: you compare the time spent on automated testing to the time taken to run these same tests manually. Then you count the cost of one hour and compare to previously made investments. Are you in the black? If yes, you are doing it right.

Rule #2 - Give your automation effort at least a month to be fully established
The truth is, if you count ROI at the first weeks of automated testing, it will be negative. At the beginning, running automated tests takes quite a while. It's only later, in a few months, you can see the true picture. Every case and every team has unique approach so consult your architects before starting automation. They should refer to previous experience and give you a timeframe.

Rule #3 - Compare ROI in stable intervals
If it's every week then it's always every week. If you are doing monthly reports, they should come out every month. The consistency in metrics gives you a clear picture.

Rule #4 - Don't confuse ROI with benefits
If you just put a check in your objectives list, it won't equal ROI. ROI means comparison between investments and profit, not just goals evaluation.

Rule #5 - At the beginning, ROI can be negative
Don't aim high right away. Automated testing can't be implemented in a day so expecting quick results is pointless.

Case in Point: IBM Testing Strategy Takeaways

When IBM analyzed the fundamentals of creating a successful testing strategy in their white paper, they identify four main problems that projects face:

- Non-extensible automation architecture. Developing a tool, testers and developers have to think not about just one particular user scenario but constantly keep the bigger picture in mind. If you have several test teams working on the project, take into account all their requirements.
- Duplication of efforts. If you have automated and manual testing teams working on the product simultaneously, the lack of proper coordination of efforts leads to the same actions performed twice. If you have already developed a similar solution, use already written tests.
- Writing tools from scratch. It's not a mistake by its nature yet now, when there are so many ready-to-go tools available, writing the new ones from the very beginning is most often a waste of resources. Monitor existing libraries and you'll most likely find something that suits your needs.
- The incapability to spread knowledge. We've already mentioned this in the ebook and IBM only proves what was said. It's crucial to manage and communicate every time and action or you will miss crucial tiny details that are not that tiny when it comes to maintenance.

Third-chapter checklist

- [ ] Management means measuring. Develop metrics at the beginning of automation. To check the efficiency of automation, check the number of automated tests.

- [ ] Evaluate clear goals. It's not just finding more bugs but find 100, 200 bugs, run 50 tests — and so on.
Keep hour count. Compare the number of hours spent on automated and manual testing. If your project fails at the beginning, it's okay. Give it a few months.

- [ ] Make sure your vendor is honest. Honesty means to talk about possible failures as well as about expected successes.

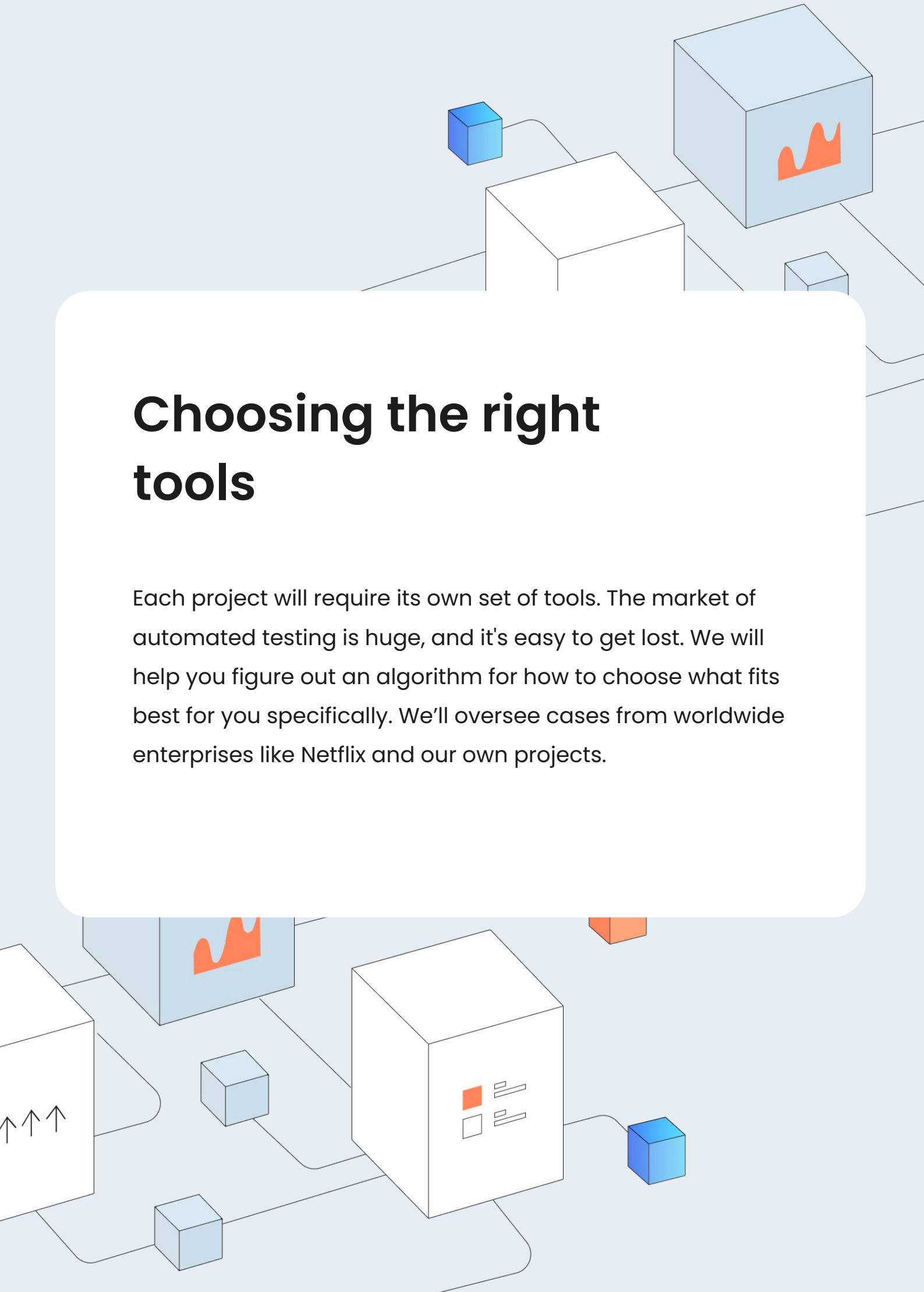
- [ ] Don't automate urgent projects if you are only at the beginning of automation. It won't be as quick as you'll need it to be.

Three takeaways from the chapter:

- [x] Make a spreadsheet with separate goals for automation and automated testing. Remember to avoid confusing these two.

- [x] Develop separate goals for each type of testing.

- [x] Communicate business benefits to your team. They should understand the project not only from a technical perspective but also its marketing objectives.



Choosing the right tools

Each project will require its own set of tools. The market of automated testing is huge, and it's easy to get lost. We will help you figure out an algorithm for how to choose what fits best for you specifically. We'll oversee cases from worldwide enterprises like Netflix and our own projects.

Choosing automated testing tool is important. The entire purpose of automation depends on how well we combine testing tools to achieve the defined objectives.

Choosing automated testing tool is important. The entire purpose of automation depends on how well we combine testing tools to achieve the defined objectives.

Testing tools are supposed to exclude the chance of human error. No mistakes can be made. Do you know what happens when something goes wrong?

This happened in November of 2000 at the National Cancer Institute. Therapy planning software miscalculated the dosage of radiation for cancer patients. Doctors found a 'hack' that allowed them to make program work faster and repeatedly gave patients twice more radiation. Why did it happen? Because software development testers didn't expect that the platform functionality could be used the wrong way, neither did they see a crucial architectural flaw that led to miscalculations. The mistake was made, and people died because of it.
Of course, stakes are not always this high. But testers can make mistakes and the task of automated tools is to make sure these accidents don't happen.

The tool alone doesn't guarantee you anything
It's not just about having a professional team (as we already discussed). It's also about adapting the instrument to your project, making sure it analyzes all possible scenarios and prevents such mistakes.

Having a right tool makes optimization easier. The question is, how can we know which one is right?

All of them seem equally fit for the project unless you know what you look for. So, the first task is to define the objectives for automated tools. What do you want it to do?

The goals can be:

- Manage tests, requirements, incidents, and defects
- Reporting and monitoring test execution
- Improve the speed and reduce the cost in X times
- Support manual testing in test planning, design, and reporting.

Don't expect one tool to cover all these objectives
Gartner says, that companies usually use tools from 3-4 different companies. Each tool falls into one of the following categories:

- It supports a specific technology
- It's responsible for a particular type of testing
- It reduces costs and makes testing faster.

Don't look for that one awesome tool that will do it all. It doesn't exist. Start with analyzing possible benefits and risks.

Possible Risks
The main enemy of automated testing are unrealistic expectations. Don't expect testing tools to do more that they can. Write clear goals and identify your needs (you already know how to do this if you've read this far) so you won't expect the impossible from the tool.
This is the list of risks identified by ISTQB (and we agree completely: these are indeed the most common concerns):

- Underestimating the time, cost and effort for the initial introduction of a tool;
- Underestimating the time and effort needed to achieve significant and continuing benefits of
- the tool;
- Underestimating the effort required to maintain the test assets generated by the tool;
- Over reliance on the tool.

What do you expect from the tool?
It all comes from the project. Project — team — tool, remember? Hence, before choosing a tool, analyze your project.

1. Is it an application or a web site? Mobile, desktop, web?
2. What technology and language did you use?
3. What kind of testing do you plan to perform?

Write it down.

Type of the project	Specifics	Language	Type of testing
Application	Desktop fintech platform	Javascript	Unit Testing

Now, come back to your strategy and remember what types of testing you were planning on automating. You can start with just one (let's say, unit testing) or do a few simultaneously. If you go for the second option, add it to your table.

Type of the project	Specifics	Language	Type of testing
Application	Desktop fintech platform	Javascript	Unit Testing
Application	Desktop fintech platform	Javascript	Regression Testing

Now, it'll be way easier for you to find a suitable tool. You know exactly what you need.
A testing tool for application, a desktop fintech platform, written in Java Script, for Unit and Regression testing.

When you know exactly what you are looking for, you can ask your architect and he'll easily give you the answers. Heck, you can even google it (not the safest approach yet possible). If we worked on such project, we'd recommend JS Unit or Jasmine for Unit Testing and Test Complete for Regression Testing.
What if I develop the next project with another language? Would I need a new tool?

No, not necessarily, If you know that you have other projects written in different languages that you'd like to automate, look for tools that are optimized for the languages you use.
For example, Test Complete that we recommended for regression testing, works not only with JS but also with C++ Script, C//Script, VB Script, Python, JScript, and DelphiScript.
You don't need another tool for other languages. Just plan it all beforehand and find a universal solution.

How to choose between 2-3 tools?
As you see, for one language and testing type there are dozens of tools. After you defined 2-5 options (the way we showed you just now), compare the options. You need to analyze the main features and put it all together.

Features	TestComplete	Selenium	UFT
Record and Playback	Yes	Limited	Yes
Scripting Languages	Python, VBScript, JScript, C++, C#, and Delphi	Java, C#, Ruby, Python, Perl, PHP , Javascript	VBScript
IDE Integrations	VisualStudio and RADStudio	Intellij, Eclipse, Visual Studio	None
Unit Testing Support	PyUnit, Ruby, PHPUnit, JUnit, NUnit, and TestNG	Java, C#, Ruby, Python,Perl, PHP , Javascript	VBScript
BDD/TDD Support	Yes	Manual	None
Data-Driven Testing	Yes	Manual	Yes
Source Control Management	Git, Subversion, Visual SourceSafe, CVS, Team Coherence	Manual	Git and Subversion

What are the most important functions to pay attention to?

- Support of Windows desktop, Web, RIA, and mobile applications
- Support not only of different operating systems but also of various OS versions Mobile devices' support
- The ability to work with different types of testing (so you don't need a new tool for each type of testing). One tool should cover 2-4 types of testing.
- The simplicity of writing tests. You don't always have developers nearby or an experienced architect around to oversee every test.
- Automated test scripting. Scripts allow writing more powerful tests much faster. Make sure the tool uses standard languages like JScript. That makes it easier for a team to get the hang of the tool as soon as possible.

- Recording test results and documenting them in brief clear reports. When a tool has a built-in documentation feature, it's easier to control and evaluate the process.
- Creating cross-browser and cross-platform tests. Writing unique scenarios for each OS and browser is a great pain in the neck — and a hole in the budget. Working with a tool that automatically adapts one test to different platforms is a more comfortable solution.

Of course, these goals depend entirely on your project nature. If you don't build cross-platform products, this feature is not your priority and go from there.

Case in Point: Netflix and its Open Source strategy

With an extensive growth of the Open Source community, companies start using assistance of
code-sharing platforms. That's precisely how Netflix strengthened its tech power by allowing testers and developers worldwide to contribute to their automated tools.

During such cooperation, the company developed the following tools:

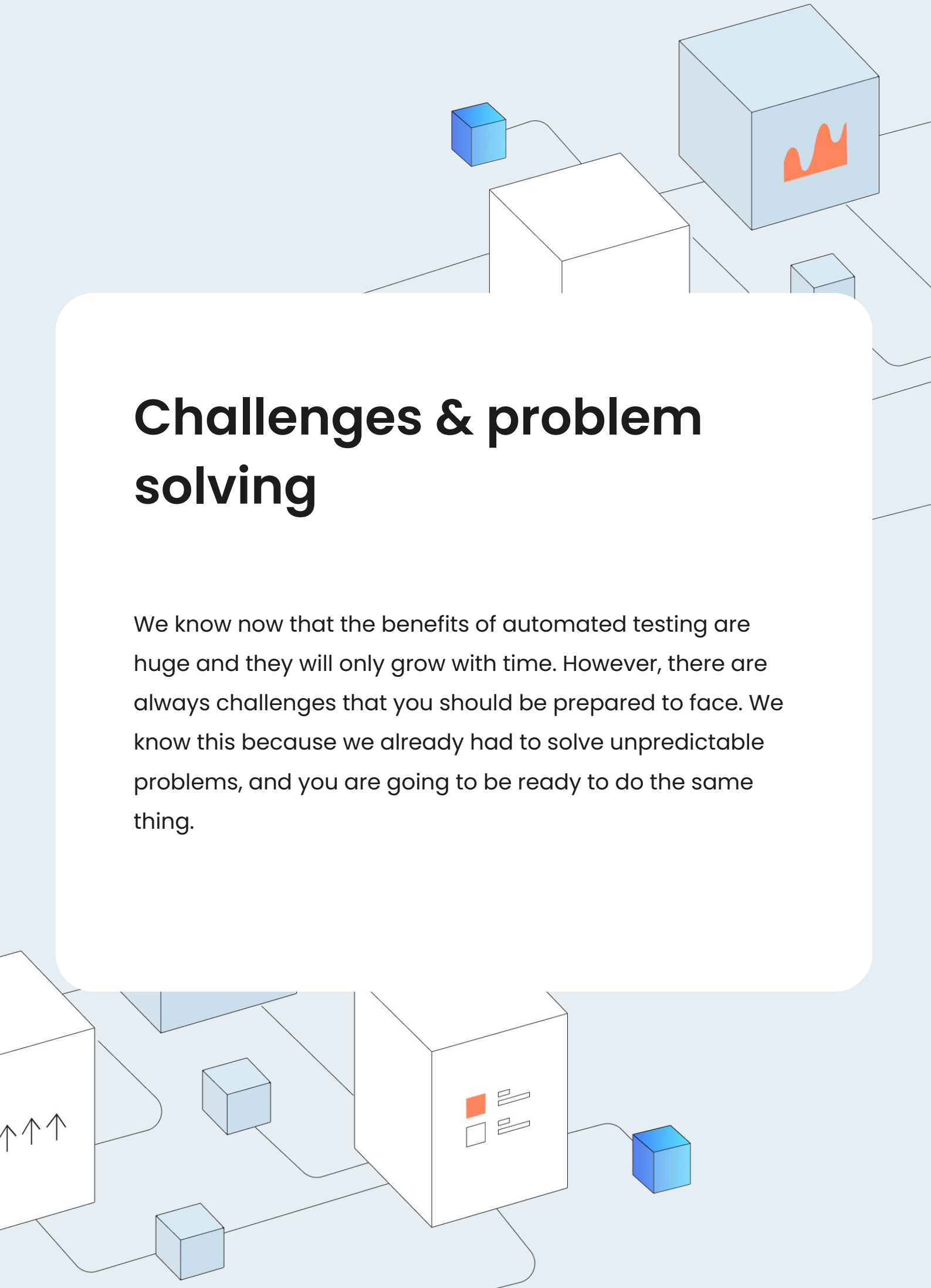
- Genie is a dynamic, REST-based abstraction to company's different data processing frameworks.
- Security Monkey identifies system weaknesses in large AWS-based environments.
- Stethoscope is a web application that collects information from existing systems management tools (e.g., JAMF or LANDESK} on a given employee's devices and gives them clear and specific recommendations for securing their systems.

Fourth-chapter checklist

- [ ] Choosing tools is a responsible task. Mistakes in software testing costs a lot as proven by many cases.
- [ ] Don't expect to find one goal which will fulfill your needs. Most testing teams use 3-5 tools for different types of testing, OSs, and programming languages.
- [ ] Put clear goals for your tool. What is the project? What are the expectation in terms of time and resources? Figure it out before investing in a tool.
- [ ] Don't trust your tool too much. It matters but the biggest influence on the project success has your strategy and teamwork.
- [ ] Determine 3-4 tools that fit the need of your projects and compare their features.
- [ ] Give tools time for installation and optimization. It can take a week or two. But don't take your team's word for it until you saw the reports and the data to prove it.
- [ ] The first tool you choose might not be the best for your business. It's okay, and we'll talk about handling such situations in the following chapters.

Three takeaways from the chapter:

- [x] Make tables like the ones we did in the chapter to collect all goals for the tool and compare possible solutions.
- [x] Use Gartner and Forrester to stay on top of all latest automation trends and insights. Also, follow Testfort blog.
- [x] If it comes to choosing, always prefer the tool that your team already worked with. It will make process faster and easier.



Challenges & problem solving

We know now that the benefits of automated testing are huge and they will only grow with time. However, there are always challenges that you should be prepared to face. We know this because we already had to solve unpredictable problems, and you are going to be ready to do the same thing.

Don't expect automated testing to be easy. It can be effective, production, profitable, it can be many things, but easy is not one of them. There are always challenges, there will be less as you get experience, but they are never going to disappear.

Each testing case is different but as we've seen from our experience, there are particular issues that always repeat. Let's go over them.

Challenge #1 — Team roles

When you are transferring from manual to automated testing, it's important to keep in mind that requirements for a test automator are different than those for manual software testers.

What is the role of a test automaton?

Testing automators are the bridge between testers and testing tools. They make sure the team's actions and the tools' algorithms are perfectly synchronized. Basically, a test automator oversees writing, designing, and maintaining the automation software, while also being in charge of scripts, additional tools, and the end results of the testing.

That said, a test automator performs functions of both testers and developers. On one hand — they support testers by coordinating their actions, solving technical issues, supporting new requirements, etc.

On the other hand, the test automator has direct responsibility for a tool's efficiency. If code needs to be adapted for a particular project, it's test automator who handles it. It's essential for them to have good programming skills.

When it comes to automated testing, a common mistake is confusing testers and automators. See, not every tester can and should be an automator. We talked about this earlier in the eBook. Of course, there are frequent cases when automators perform functions of testers and vice versa, but mixing these roles often leads to confusion.

Responding to the challenge: define who performs functions of testers and who is an automator. There can be several automators if they know how to cooperate with each other. Should we force a non-technical tester to obtain development skills? No. These experiments usually end with losing a tester and not getting a good programmer. Let testers stay testers.

Remember what we discussed at the beginning of the book? If testers don't feel comfortable with automation challenges, give them time to ease into it.

Challenge #2 — Cooperation with developers

Successful automated testing teams consist of three groups: testers, test automators, and developers. We already discussed the approaches to testers and test automators and why they are different. What about developers?

There are few golden rules to follow when it comes to tester-developer relationships in automated testing.

Golden Rule #1 — Develop with automated testing in mind

You know why it's best to implement automation as early as possible? Because you start preparing your project for automation from the very first lines of code written for your software solution.

When the testing team has a say in development, they can say things like “Please use standard controls to make further automated testing easier."

This is a small detail. It's easy to implement for most developers. However, when the time comes for automated testing, these small details (if not implemented) become a huge pain in the neck for testing automators.

What to do if the project is already developed?

At this point there is usually not much you can change in the project's architecture. Sure, there are always little code adjustments that can be made, but nothing significant can be done anymore. Is the hope lost for such projects?

Of course it's not. The test automation is still possible, but be prepared for it to be more complicated and take longer. Here's what you can do to ease the pain:

- Make sure testers have access to all necessary information about the project: what technologies were used, what are the possible complications, what functions might cause more trouble than others. Developers usually have these things documented (if they don't, they should) and testers will benefit from the insights.

- Schedule daily meetings between testers and developers so testing team can ask for necessary details before starting their workday.
- When there are significant changes of functionality to be made, both by developers and testers, this should be discussed beforehand. Why-haven't-you-asked issues are expensive to tackle and easy to avoid if you communicate.

Golden Rule #2 — Responsibility

Make sure both parties take responsibilities for the project, not passing the blame onto the other party. In fact, a love-hate relationship between development and testing team can be tackled if they work on same strategic issues:

- Collect customers insights and explore user journey
- Build stories and discuss project functionality
- Get clear definition of 'done' to avoid later criticism
- Adjust definitions of done as testing progresses.

With DevOps around, conversations about sharing responsibility for all project stages can become overwhelming. But the truth is, successful automated testing teams have worked with DevOps for a long time. In automated testing, the concept of DevOps is so natural and so logical that it's impossible to imagine a strong team working non-DevOps.

Challenge #3 - Choosing a country for outsourcing

As we said, you can either switch some of your manual testers to automation or outsource the process to an offshore team. Overall it's an easier method because you work with well-qualified automa ed testers which reduces the time of adaptation. All of a sudden, the question of training an internal manual tester in a new workflow and technology is no longer an issue.

Cost comparison of in-house and outsourced QA

Criterion	In-house QA	Outsourced QA
Number of programmers		
Man hours needed	100	100
Middle QA expert hourly rate	30 $	10 8
Taxes, bonuses, training	+ 30 to the salary	
Travel cost		
Estimated testing cost	$ 12k minimum	$ 3k + travel cost

The main advantage of outsourcing testing is lower cost, of course

However, there are challenges and important decisions to make. The most pressing one, as we think, is how to pick a country. It's no secret that there are plenty of international teams who offer their services as outsourcing testers. Whatever country you choose, there are dozens of companies.

Why does country even matter?

It's not ffte country vre are choosing. It's testing learn so its qualification should 6e the criteria, not the location. Right?

Right and also wrong.

Yes, you are choosing the team, not the country. Yes, making location the main search criteria is not the wisest choice. But location matters. It should not be your first priority but it definitely has to make it to top-3 or top-5 of your concerns. Why so?

1. You don't just choose a country, you choose an entire market. This will affect prices, qualifications, and availability of resources. A country is not just a location, it's the conditions of your testing team training. Things like stable internet uptime, time-zones, and language barriers can become a problem, which leads to:
2. Communication plays the key part in automated testing. If you have manual and automated testing teams working simultaneously, they should interact with each other. And don't forget about developers! Consider time zones and language differences.
3. Costs. Looking for outsourcing companies in the US is expensive due to high costs of human resources. The cheapest markets are Eastern Europe, India, and China, each with varying prices, different qualification levels, and various levels of technical expertise.

Do I choose the cheapest market?

Cost matters but it definitely shouldn’t be the main concern. It's best to combine all the described factors, assess their importance for the project — and go on.

Criteria	How does it matter for the project (on scale from 1 to 10)?	Countries that fit best
Communication and management	If you develop a security-sensitive project (like a banking platform), that's the most pressing issue. If that's a single-purposed mobile app, let's put 3/10.	For big projects: closer to you. If you are located in the US, choose a company that has an office or at least competent representatives in the US For smaller ones, where communication is not that high on the list, Eastern Europe will do just fine.
Market	Look at the technology you are interested in. Let's say, you need AI.	For AI, the big three are India, China and the U.S, according to CNN. You can follow the same procedure for any other technology.
GHOStS	Developers have different salaries on various markets. The lower is the salary, the lower will be the testing cost.	India, Eastern Europe

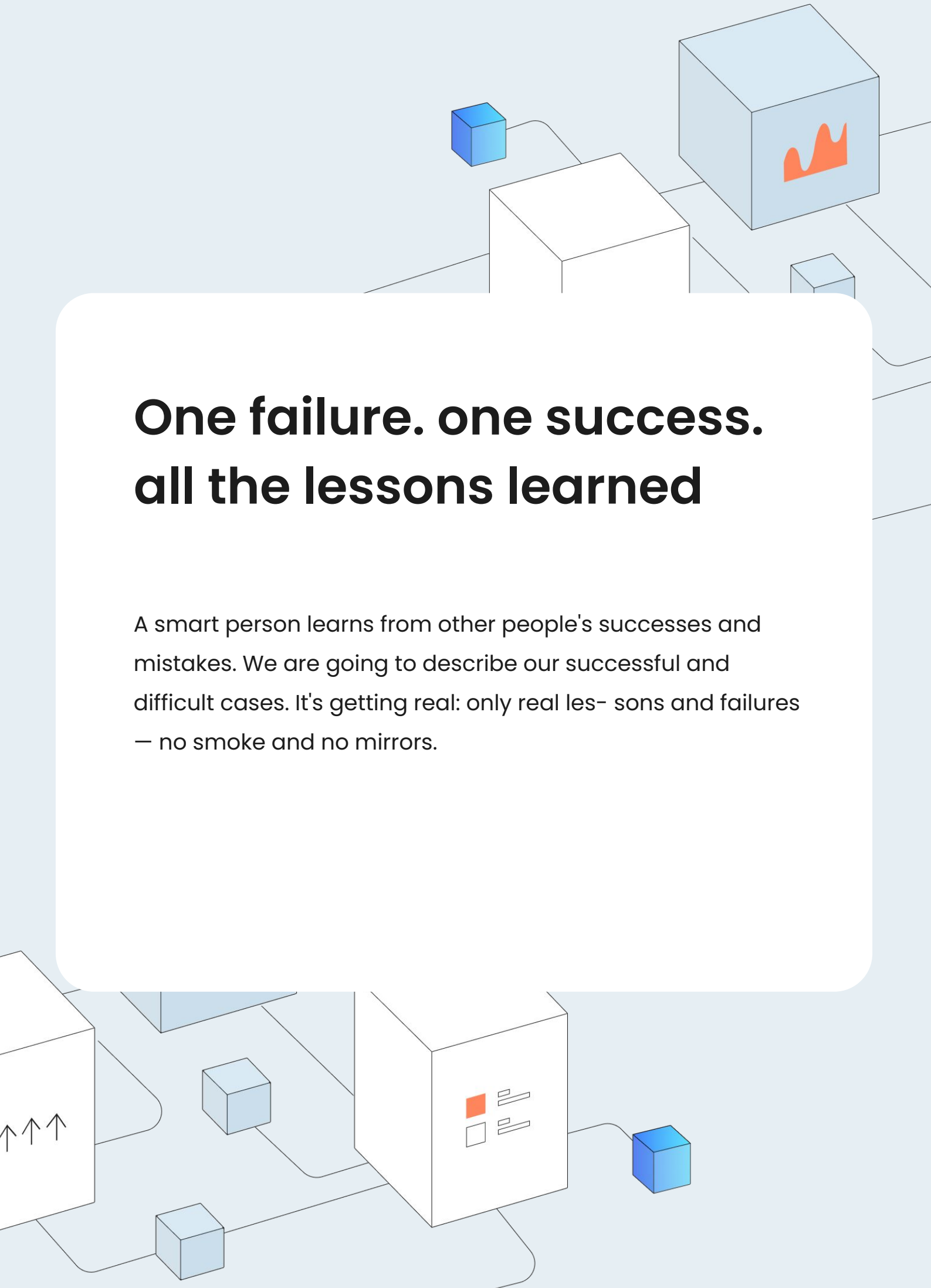
For global project, communication and qualification matter most. For smaller ones, it's okay to put cost as your top priority (doesn't mean you shouldn't aim for good quality though).

Fifth-Chapter Checklist:

- [ ] If you outsource, base your choice on three criteria. Assess the market, evaluate communication challenges, and calculate the cost.
- [ ] Make sure testers and developers cooperate. It's best to start at the very beginning of development so the product is made with automated testing in mind.
- [ ] Differentiate the team roles. Test automators should have both strong testing and development background. When it comes to testers, they shouldn't necessarily have programming skills so don't force them into it.
- [ ] Share responsibility. All members of your team, both testing and development, should participate in planning and strategy building. To avoid conflicts, set a clear definition of done but update it as the project moves forward.

Three takeaways from the chapter:

- Automated testing is only successful if it applies principles of Agile and DevOps. Be sure to apply these methodologies. Take a look at our 5 Tips for Agile Test Automation and a comprehensive analysis of why DevOps Need Test Automation.
- Make testing backlog available for developers — and vice versa. That will increase transparency and simplify cooperation.
- Take a look at Deloitte outsourcing survey. It will help you to define the criteria when choosing an offshore team.



One failure. one success. all the lessons learned

A smart person learns from other people's successes and mistakes. We are going to describe our successful and difficult cases. It's getting real: only real les- sons and failures — no smoke and no mirrors.

We've already discussed what makes the most difference between a successful automated testing project and a failed one. To make an entire picture even more clear, we decided to share our own experience of automated testing, demonstrating what had the biggest influence on the end outcome.

Due to confidentiality reasons, we can't share specific client's name, but we are going to go even further. Instead of taking just one project, we've analyzed a few of our failures and successes and drew a common denominator to determine a pattern of automated testing victories.

A story of a lost case

As any testing team, we did not come to the success right away. If there is one thing you need to re- member from this book, it's a simple thought: automated testing might not be done perfectly right away, if not managed by an experienced team. If you don't have thorough guidance, be ready to face shortcomings and frustrations.

Ten years ago, we started exploring automated testing. Not just for the easiest testing cases, but actually building complex strategies and managing big teams of automated testers. We started the experiment with our own tool for project management — and good thing it was our own product, and we were the only ones to bear the responsibilities of the failure.

Overall, the process of transferring from manual to automated testing, building strategy and managing the project took us 6 months. We invested in hiring a professional testing architect and even added a couple of professional automated testers to the team. It seemed that all preparation is done right... just to see, six months later, that it wasn't.

Here we are, ten years later, with hundreds of successful cases of automation under our belt, and with a clear understanding of what was done wrong. Now, a quick warning: while reading this chapter, it might seem to you that the mistakes are obvious. They're not. Don't jump to conclusions, because it's the little "obvious" things that topple your testing efforts.

A few quick details about the case:

Time spent: 6 months

Tested product: a project-management application, specifically designed for custom software development by our in-house team

Goals: to automate the entire testing process, reducing the number of manual testers to the lowest number possible

What went wrong?

Back then it seemed as if we are doing everything by the book. That's why, when six months later we encountered a sad failure, we were confused: what happened. However, as time passed and we got more experience in automated testing, we began to understand what factors can either contribute to a great success or become a reason of a failure.

Factor #1 — Unqualified leadership

As was already mentioned, it was the first complex automated testing project for our team where we were fully in charge of the process, starting from setting goals and building strategy, finishing with the full execution of automated tools and algorithms.

There's a reason we started this eBook with advice on having your entire team on board.

Our first challenge was communicating the purpose of automation. Our QA architect was one of the very few people on the team who really understood the necessity of automation. However, team management failed to communicate these goals to testers, and the team saw that the hustle is just not worth the result.

As our testers explained later, they all thought that the product will be faster tested manually because we wouldn't have to waste time on choosing tools, writing algorithms, and testing them.

Of course, from a short-term perspective they were completely right, but we were looking for long-term results that we failed to communicate to the rest of the team. That's why at the end we ended up with an unmotivated team and unclear goals for the entire automation strategy.

Factor #2 — No cooperation with manual testers

Even though our goal was initially to switch from manual testing to automation, we completely ignored the perspective of automated testers. We hired an experienced automated testing architect who had spoken to manual testers as if they already have experience in automation. Instead of using the experience that manual testing already had with a project, we forced them to retrain, completely ignoring their previous background.

Factor #3 — Lack of patience

The major problem we faced while working on the project was an underestimation of the challenges we had ahead. For a complete automation from scratch, six months are a pretty short period of time as it is, and we planned to do everything in 3 months! Imagine the team disappointment when we realized that this deadlines aren't going to cut it.

A story of a successful case

Even though the first serious failure really pulled the rug from under our confidence, we understood that mastering automation is a logical step of our QA Lab development. Of course, we couldn't stop on one unlucky outcome. After a thorough analysis of what went wrong, we started again.

We learned our lessons

Now, when we look back at this project, it's apparent that failures we experienced during our first case contributed most to the success of the next one. Even though we didn't know for sure what to do to make it right, we had a clear idea what not to do. Therefore, we weren't just blindly experiment- ing like the last time but already knew what could the possible roadblocks be.

During our preparational team meeting, we went through all the documentation from the previous

case to determine what factors caused that outcome. It also proved to us one more time just how important QA documentation is. If you have a successful case, you might want to know how you did it — so it can be

repeated again. If you mess up, well, at least you can come back and see what really went wrong.

That's what we did, and that's what helped us identify critical issues in our last project. We wrote down what test cases were the biggest success and what became the most disappointing failure. We have also saved the records of team performance — this way it was clear who should be the part of our next project, and who shouldn't.

Communication played a crucial part

When you are set out to have a successful project, you want to keep track of all main automation processes. If in the failed project we mostly depended on team meeting where discussed performed tasks and achieved objectives, this time our priority was to collect a set of comprehensive testing reports from each group of testers, who were responsible for a particular type of testing.

Every weak project manager received these documentation to comply it into a comprehensive report. Putting two hours per week on this might seem uncalled for but realistically, these two hours save you days of later struggle.

After you have specific documentation of your hands, the efficiency of collective meetings doubles. You'll have real data to discuss with your testers and management, and that improved the teamwork drastically.

If you read the entire book carefully, these insights hardly were totally new for you — we already discussed most of them throughout the text. Let's just go through main lessons again and make sure they really stick.

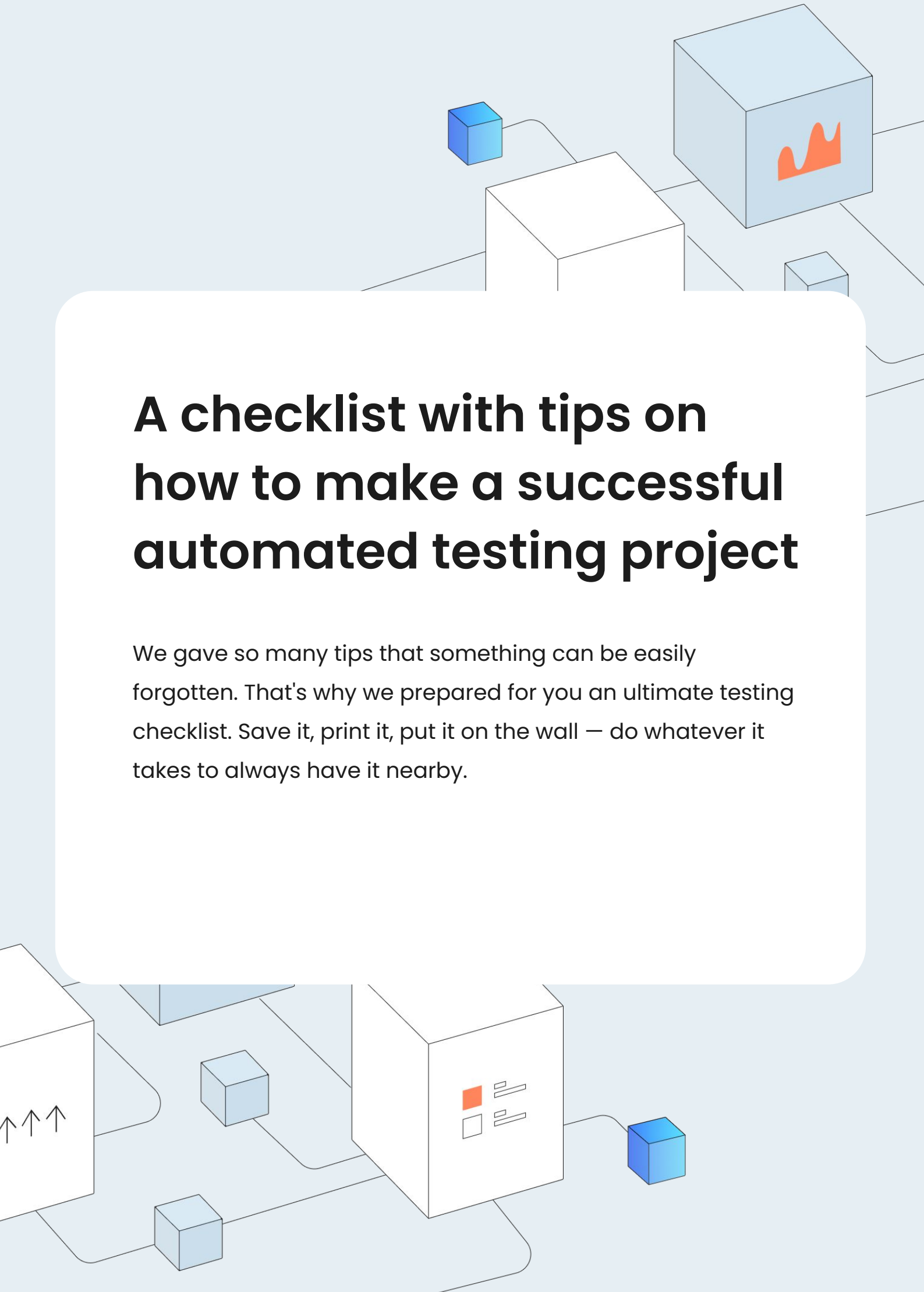
Crucial factors	Failure	Success
Management and communication	No experience in managing complex automation processes, building strategy, and identifying goals	Obtaining guidance of professional QA automation architect along with the co- operation with manual testes.
Testing and development practices	Weak cooperation with developers, random choice of testing tool. We didn't consult development team before writing automation algorithms.	We built reusable libraries of functions with developers' assistance and consult- ed them every step along the way.
Strategy	Unrealistic goals lead to the unrealistic planning	We understood that automated tools requires testing. Management and communication issues are not to be ruled out as well.

Sixth-chapter checklist

- [ ] Don't rush automation. Make sure both your team and project are ready for it.
- [ ] Check whether your testing team coordinates their effort with manual testers. Could be, they are working on the same cases without even knowing about it.
- [ ] Don't let your expectations jump too high. Ask your team manager about possible risks and delays. If you get the everything-is-fine answer, there is something the team is not telling you.
- [ ] Give tools time for installation and optimization. It can take a week or two. But don't take your team's word for it until you saw the reports and the data to prove it.
- [ ] The first tool you choose might not be the best for your business. It's okay, and we'll talk about handling such situations in the following chapters.

Three takeaways from the chapter:

- [x] Make tables like the ones we did in the chapter to collect all goals for the tool and compare possible solutions.
- [x] Use Gartner and Forrester to stay on top of all latest automation trends and insights. Also, follow Testfort blog.
- [x] If it comes to choosing, always prefer the tool that your team already worked with. It will make process faster and easier.



A checklist with tips on how to make a successful automated testing project

We gave so many tips that something can be easily forgotten. That's why we prepared for you an ultimate testing checklist. Save it, print it, put it on the wall — do whatever it takes to always have it nearby.

You are past six chapters — congratulations! Now you know how to approach test automation from the very beginning to the final stages of the process.

There is one thing left. We prepared a checklist for you so you can repeat and summarize all the key takeaways.

The success of automated testing depends on putting the right objectives. These most often are:

- Shorter delivery delay
- Reduction of the workload
- Improving testing coverage and flexibility
- Increasing motivation and productivity of testers by making automated execution of repetitive tasks.

The testing process has its own lifecycle:

Planning

- [ ] Making Automation Test Plan
- [ ] Analyzing previous projects and identifying possible risks
- [ ] Assessing team resources, their skills, and possibilities
- [ ] Choosing tools based on technical project specifics
- [ ] Determine the automation framework
- [ ] Evaluate existing manual tests and scripts so the duplication of efforts can be avoided.

Planning

- [ ] Identify the scenarios that your testing team will be automating
- [ ] Prepare traceability matrix
- [ ] Create and customize the automation framework
- [ ] Specify the description of all testing processes

Planning

- [ ] Prepare automation scripts
- [ ] Test and debug automation scripts
- [ ] Review the reports after scripts were created and tested, ensure all most important user scenarios are covered.
- [ ] Make changes to traceability matrix along the way.

Execution

- [ ] Prepare test environment
- [ ] Perform Test Run
- [ ] Update Automated Framework if required Automatic Replay

Completion

- [ ] Record and evaluate test results
- [ ] Compare results with the initially planned workload and expected cost-efficiency (it's called baseline)
- [ ] Automate the generation of testing documentation with result evaluation.

Test infrastructure

- [ ] Setup the necessary infrastructure and develop it in the process
- [ ] Make an automation lab for big projects
- [ ] Automate the configuration of test means

Control

- [ ] Measure test creation and execution with color-tagged heatmaps and weekly result reports
- [ ] Monitor and report the results.

There you have it, the entire process of software testing automation. Each of them matters and impacts the end outcome greatly. Yet no matter how important all these stages are, the most crucial one is to start automation.

To start automating, contact an experienced QA team that will share with your their knowledge, insights, and resources. Like TestFort, a testing laboratory with award-winning testers and QA engineers. With their expertise and this knowledge combined, you will easily keep track of the process of automation.

To get more insights on automated testing, go to Testfort blog. There we publish regular insights on planning and executing automated testing. Basically, it's a logical continuation of this ebook — more takeaways, more expertise, more knowledge.

Your QA process deserves more than guesswork. Let's build it right together.

Start with intro talk with our Head of Testing Department.

Plan a call

Testfort is a QA and software development company with 23+ years of experience in the market. Our team doesn't just write code or run tests — we think like product owners. We offer flexible engagement models because we know one size doesn't fit all. Our team is focused on creating impact-driven solutions with your end goal in mind. We're not here to just tick boxes; we're here to make your product succeed.