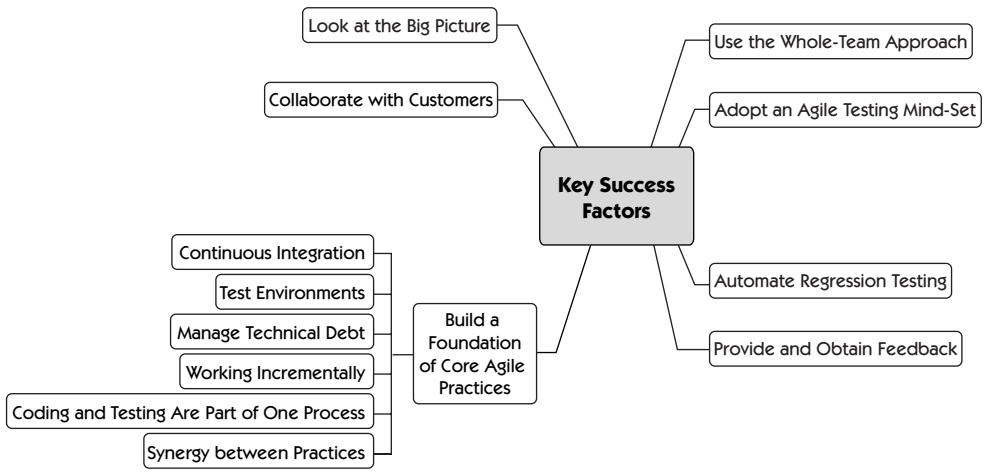
Chapter 21

KEY SUCCESS FACTORS



Having traveled through an iteration and beyond, following an agile tester as she engages in many activities, we can now pick out some key factors that help testers succeed on agile teams and help agile teams succeed at delivering a high-quality product. We think agile testers have something special to offer. "Agile-infected" testers learn how to apply agile practices and principles to help their whole team produce a better product. "Test-infected" programmers on agile teams learn how to use testing to produce better work. Lines between roles are blurred, but that's a good thing. Everyone is focused on quality.

We have gleaned some critical testing guidelines for agile teams and testers from our own trial and error as well as from teams with which we've worked. These guidelines are built on the agile testing matrix, on our experience of learning to overcome cultural and organizational obstacles, our adventures in performing the tester role on agile teams, and our experience of figuring out how best to use test automation. We like lucky numbers, so in this chapter we present seven key factors that help an agile tester succeed.

We asked a small group of people who were reviewing some of our chapters to suggest the order in which to present these success factors. The results varied quite a bit, although many (but not all) agreed on the top two. Pick the success factor that will give you the biggest return on investment, and start working on it today.

SUCCESS FACTOR 1: USE THE WHOLE-TEAM APPROACH

When the whole development team takes responsibility for testing and quality, you have a large variety of skill sets and experience levels taking on whatever testing issues might arrive. Test automation isn't a big problem to a group of skilled programmers. When testing is a team priority, and anyone can sign up for testing tasks, the team designs testable code.

Making testers truly part of the development team means giving them the support and training they need to adapt to the fast pace of agile development. They have time to acquire new skills in order to collaborate closely with members of both the development and customer teams.

If you manage an agile team, use the suggestions in Part II, "Organizational Challenges," to help your team adopt the whole-team approach. Remember that quality, not speed, is the goal of agile development. Your team needs testers to help customers clarify requirements, turn those into tests that guide development, and provide a unique viewpoint that will promote delivery of a solid product. Make sure the testers can transfer their skills and expertise to the rest of the team. Make sure they aren't pigeonholed in a role such as only doing manual testing. Make sure that when they ask for help (which may require considerable courage on their part), their team members give it. The reverse is true, too; a tester should step up whenever someone needs assistance that they can provide.

See Chapter 2, "Ten Principles for Agile Testers," for an example of how the "Power of Three" works.

If you're a tester on an agile team, and there are planning meetings and design discussions happening that don't include you, or the business users are struggling to define their stories and requirements alone, it's time to get up and go talk to the rest of the team. Sit with the programmers, invite yourself to meetings, and propose trying the "Power of Three" by involving a tester, a programmer, and a business expert. Be useful, giving feedback and helping the customers provide examples. Make your problems the team's problems, and make their problems yours. Ask your teammates to adopt a whole-team approach.

SUCCESS FACTOR 2: ADOPT AN AGILE TESTING MIND-SET

In Chapter 2, "Ten Principles for Agile Testers," we cautioned agile testers to lose any "Quality Police" mind-set they might have brought with them. You're on an agile team now, where programmers test and testers do whatever they can think of to help the team deliver the best possible product. As

we emphasized in Chapter 2, an agile testing attitude is proactive, creative, open to new ideas, and willing to take on any task. The agile tester constantly hones her craft, is always ready to collaborate, trusts her instincts, and is passionate about helping the team and the business succeed.

We don't mean that you should put on your Super Tester cape and go protect the world from bugs. There's no room for big egos on agile teams. Your teammates share your passion for quality. Focus on the team's goals and do what you can to help everyone do their best work.

Use agile principles and values to guide you. Always try the simplest approach to meeting a testing need. Be courageous in seeking help and experimenting with new ideas. Focus on delivering value. Communicate as directly and as often as possible. Be flexible in responding to change. Remember that agile development is people-centric, and that we should all enjoy our work. When in doubt, go back to the values and principles to decide what to do.

See Chapter 2, "Ten Principles for Agile Testers," for more about the agile testing mindset.

An important component of the agile testing mind-set is the drive to continually find better ways to work. A successful agile tester constantly polishes her craft. Read good books, blogs, and articles to get new ideas and skills. Attend local user group meetings. Participate in mailing list discussions to get feedback on problems or new ideas. If your company won't pay for you to attend a good conference, put what you've learned into an experience report to exchange for a free conference registration. Giving back to your testing and agile development communities will help you, too.

Experiment with new practices, tools, and techniques. Encourage your team to try new approaches. Short iterations are ideally suited to experimentation. You might fail, but it'll be fast, and you can try something else.

If you manage agile testers or an agile team, give them time to learn and provide support for the training they need. Remove obstacles so that they can do their best work.

When you're faced with problems that impact testing, bring those problems to the team. Ask the team to brainstorm ways to overcome these obstacles. Retrospectives are one place to talk about issues and how to resolve them. Keep an impediment backlog and address one or two in every iteration. Use big visible charts, or their virtual equivalents, to ensure that everyone is aware of problems that arise and that everyone can track the progress of coding and testing.

SUCCESS FACTOR 3: AUTOMATE REGRESSION TESTING

Can an agile team succeed with no test automation? Maybe, but the successful teams that we know rely on automated regression tests. As we've said often in this book, if you're spending all your time doing manual regression testing, you'll never have time for the important exploratory testing that will ferret out the damaging behaviors lurking in the code.

Agile development uses tests to guide development. In order to write code to make a test pass, you need a quick and easy way to run the test. Without the short feedback cycle and safety net regression that suites provide, your team will soon become mired in technical debt, with a growing defect queue and ever-slowing velocity.

See Part II for more on the Agile Testing Quadrants.

Automating regression tests is a team effort. The whole team should choose appropriate tools for each type of test. Thinking about tests up front will let programmers design code for ease of test automation. Use the Agile Testing Quadrants and test automation pyramid to help you automate different types of tests effectively.

See Chapter 14, "Automation Strategy," for more on the test automation pyramid.

Remember to start simply. You'll be surprised at how much value some basic automated smoke tests or automated unit tests can provide.

See the bibliography for resources on promoting change.

Test automation is a team effort. It's also hard, at least at first. There's often a big "hump of pain" to overcome. If you manage a development or testing team, make sure you're providing enough support in the form of time, training, and motivation. If you're a tester on a team with no automation, and the programmers are too frantic trying to write production code to stop and think about testing, you have a big challenge ahead of you. Experiment with different ways of getting support from management and from team members to start some tiny automation effort.

SUCCESS FACTOR 4: PROVIDE AND OBTAIN FEEDBACK

Feedback is a core agile value. The short iterations of agile are designed to provide constant feedback in order to keep the team on track. Testers are in a unique position to help provide feedback in the form of automated test results, discoveries made during exploratory testing, and observations of actual users of the system.

Agile Is All about Feedback

Bret Pettichord, CTO of WatirCraft and co-author of *Lessons Learned in Software Testing*, shared these thoughts on the importance of feedback to agile development.

> Agile methods allow your team to get feedback regarding the software you are building. That's the point. The feedback works on several levels. Pair programming gives developers instant feedback on their code. Stories represent units of work where testers and analysts can give feedback to developers. Iteration releases facilitate feedback from outside the team. Most agile practices are valuable because they create feedback loops that allow teams to adapt.
>
> A lot of teams adopt Agile with a grab-bag approach without quite realizing the point of the practices. They pair-program without discussion or changing drivers. They send code to QA that the testers can't test because the story boundaries are arbitrary; they can't tell whether they found a bug or just the end of the story. Iterations become schedule milestones rather than real opportunities to improve alignment and adjust objectives.
>
> The reason Agile teams can do with less planning is because feedback allows you to make sure that you are on course. If you don't have meaningful feedback, then you're not agile. You're just in a new form of chaos.
>
> On my last project, we defined our stories so that they made sense to everyone on the team. Our analysts, testers, and developers could all understand and review individual stories. But we found that we had to create a larger grouping, which we called features, to facilitate meaningful review from outside our team. We made sure all the stories in a feature were complete before soliciting feedback from outside the team.
>
> Being able to give and receive meaningful feedback is often a challenge for people. Yet it is crucial to success with Agile.
>
> Agile teams get into terrible binds when executives or clients hand them a list of requirements at the start, tell them to use Agile (because it's faster), and then don't want to participate in the feedback process.
>
> Agile isn't faster all by itself. Agile is only a benefit in a world that acknowledges the value of adapting. And that adaptability needs to go all the way to whoever is funding the project. It is not enough for the team to be agile. The sponsors need to be agile too. Are all of the requirements really required? Do we know exactly what the software needs to look like from the start?

> Agile is faster because feedback allows you to find and focus on the most valuable features. If you are certain you know what needs to be built, don't use Agile. If you don't have time to gather and act on feedback from customers, then don't use Agile. If you are sure that everyone understands exactly what needs to be done from the start, then don't use Agile.
>
> Agile practices build a technical and organizational infrastructure to facilitate getting and acting on feedback. If you aren't going to adapt to feedback, then this infrastructure is waste that will only slow you down.
>
> To us, the value of agile development isn't that it's faster but that it delivers enough value quickly enough to help the business grow and succeed. Testers play a key role in providing the feedback that allows that to happen.

Testers need feedback too. How do you know that you have the right examples of desired behavior from the customers? How do you know if the test cases you wrote reflected these examples correctly? Can the programmers understand what to code by looking at the examples you've captured and the tests you've created?

One of the most valuable skills you can learn is how to ask for feedback on your own work. Ask the programmers if they get enough information to understand requirements and whether that information guides their coding. Ask customers if they feel their quality criteria are being met. Take time in both the iteration planning meetings and retrospectives to talk about these issues and suggest ways to improve.

SUCCESS FACTOR 5: BUILD A FOUNDATION OF CORE PRACTICES

An old saying in the testing business is, "You can't test quality into the product." This is, of course, true of agile development as well. We feel you can't deliver high-quality software without following some fundamental practices. While we think of these as agile practices, they've been around longer than the term "agile development," and they're simply core practices of successful software development.

Continuous Integration

Every development team needs source code management and continuous integration to be successful. You can't test effectively if you don't know exactly

what you're testing, and you can't test at all if you have no code you can deploy. All team members need to check in their work at least once a day. Every integration must be verified by an automated build that includes tests to provide rapid feedback about the state of the software.

See the bibliography for more information about continuous integration.

Implementing a continuous integration process should be one of the first priorities of any software development team. If your team doesn't have at least a daily verified build, stop what you're doing and get one started. It's that important. It doesn't have to be perfect to start with. If you have a huge system to integrate, it's definitely more challenging. In general, though, it's not that difficult. There's a plethora of outstanding tools, both open source and commercial, available for this purpose.

Test Environments

You can't test productively without a test environment that you control. You need to know what build is deployed, what database schema is being used, whether anyone else is updating that schema, and what other processes are running on the machine.

Hardware is getting less expensive all the time, and more open source software is available that can be used for test environments. Your team must make the investment so that you can effectively conduct automated and manual exploratory tests quickly and efficiently. If there's a problem with the test environment, speak up and let it be a problem for the team to solve creatively.

Manage Technical Debt

Even good software development teams, feeling time pressure, neglect refactoring or resort to quick fixes and hacks to solve a problem quickly. As the code becomes more confusing and hard to maintain, more bugs creep in, and it doesn't take long before the team's velocity is consumed by bug fixes and trying to make sense out of the code in order to add new features. Your team must constantly evaluate the amount of technical debt dragging it down and work on reducing and preventing it.

People often say, "Our management won't give us time to do things right, we don't have time to refactor, and we're under tight deadlines." However, it's not hard to make a clear business case showing what growing technical debt is costing the company. There are many ways to measure code and defect rates that can translate technical debt into its impact on the bottom line. Merely pointing to your decreasing velocity may be enough. Businesses need

their software development teams to remain consistently productive. They may have to reduce the scope of their desired features in order to allow enough time for good, test-guided code design and good practices such as continual small refactoring.

Good coverage from automated regression tests is key to minimizing technical debt. If these are lacking, budget time in each iteration to build up the automated tests, plan a "refactoring iteration" to upgrade or add necessary tools, and write tests and do major refactoring efforts. In every iteration, take the time to guide code with tests, refactor the code you're touching as needed, and add automated tests where they're missing. Increase your estimates to account for this work. In the long run, the team will be able to go much faster.

Working Incrementally

One reason agile teams are able to create a quality product is that they work on a small scale. Stories represent a few days of work, and each story may be broken into several thin slices or steel threads and built step-by-step. This allows testing each small piece and then incrementally testing as the pieces are put together.

Read more about small chunks and thin slices in Chapter 8, "Business-Facing Tests that Support the Team."

If your team members are tempted to take on a large chunk of functionality at once, encourage them to look at a stepwise approach. Ask questions: "What's the central business value in this story? What's the most basic path through this piece of code? What would come next?" Suggest writing task cards to code and test the small pieces, get a proof of concept for your design, and confirm your test and test automation strategy.

Coding and Testing Are Part of One Process

People who are new to agile often ask agile testers, "What do you do until all the stories are finished and you can test?" Experienced agile practitioners say, "Testers must be involved throughout the whole iteration, the whole development process. Otherwise it doesn't work."

Testers write tests, based on examples provided by customers, to help programmers understand the story and get started. Tests and examples provide a common language that everyone involved in producing the software understands. Testers and programmers collaborate closely as coding proceeds, and they both also collaborate closely with the customers. Programmers show testers the functionality they've written, and testers show programmers the unexpected behaviors they've found. Testers write more tests as coding proceeds, program-

mers make them pass, and testers do more exploratory testing to learn whether the right value has been delivered. Each agile iteration consists of dozens of constant, quick, incremental test-code-test-code-test iterations.

When this collaboration and feedback cycle is disturbed, and testing is separated from development, bad things happen. If a story is tested in the iteration after which it was coded and bugs are found, the programmer has to stop working on the new story, remember how the code worked for the last iteration's story, fix it, and wait for someone to test the fix. There are few facts in software development, but we know for sure that bugs are cheaper to fix the sooner they're found.

Read more about coding and testing in Chapter 18, "Coding and Testing."

When coding is constantly guided by tests, and testing happens alongside coding, we're much more likely to achieve the behavior and provide the value that the customer wanted. Testing is a team responsibility. If your team doesn't share this view, ask everyone to think about their focus on quality, their desire to deliver the best possible product, and what steps they can take to ensure that the team achieves its goals.

Synergy between Practices

A single agile development practice such as continuous integration can make a difference, but the combination of multiple agile practices is greater than the sum of the parts. Test-driven design, collective code ownership, and continuous integration together deliver rapid feedback, continually improving code design and the ability to deliver business value quickly. Automating tests is good, but using automated tests to drive development, followed up by exploratory testing to detect gaps or weaknesses, is many levels of magnitude better.

Some practices don't work well in isolation. Refactoring is impossible without automated tests. It's possible to do small releases in a mini-waterfall fashion and avoid all benefits of agile development. If your on-site customer isn't empowered to make decisions, her value to the team is limited.

Agile practices were designed to complement each other. Take time to understand the purpose of each one, consider what is needed to take full advantage of each practice, and make thoughtful decisions about what works for your team.

SUCCESS FACTOR 6: COLLABORATE WITH CUSTOMERS

Some of the greatest value that testers contribute to agile teams is helping customers clarify and prioritize requirements, illustrating the requirements

with concrete examples of desired behavior and user scenarios, and turning those examples into executable tests. Testers speak the domain language of the business and the technical language of the development team. We make good facilitators and translators.

Never get in the way of direct communication between programmers and customers. Do encourage as much direct communication as possible. Use the "Power of Three." When requirements are missed or misunderstood, a customer, programmer, and tester need to work together to get questions answered. Get the customers talking in front of a whiteboard or its virtual equivalent as often as necessary. If customers are scattered around the campus, the country, or the globe, use every tool you can find to enhance communication and collaboration. Teleconferences, instant messages, and wikis aren't an ideal replacement for face-to-face conversation, but they beat sending emails or not talking at all.

SUCCESS FACTOR 7: LOOK AT THE BIG PICTURE

This is a generalization, of course, but we've found that testers tend to look at the big picture, and usually from a customer point of view. Programmers usually have to focus on delivering the story they're working on now, and while they may be using tests to guide them, they have to focus on the technical implementation of the requirements.

This big-picture viewpoint is a huge contribution to the team. Test-driven development, done well, delivers solid code that may, in isolation, be free of defects. What if that new feature causes some apparently unrelated part of the application to break? Someone has to consider the impact to the larger system and bring that to the team's attention. What if we've overlooked some little detail that will irritate the customers? The new UI may be flawlessly coded, but if the background color makes the text hard to read, that's what the end user's going to notice.

Part III explains how to use the Agile Testing Quadrants.

Use the Agile Testing Quadrants as a guide to help you plan testing that will cover all the angles. Use the test pyramid idea to ensure good ROI from your test automation. Guiding development with tests helps make sure you don't miss something big, but it's not perfect. Use exploratory testing to learn more about how the application should work, and what direction your testing needs to take. Make your test environments as similar as possible to production, using data that reflects the real world. Be diligent about re-creating a production-style situation for activities such as load testing.

It's easy for everyone on the team to narrowly focus only on the task or story at hand. That's a drawback of working on small chunks of functionality at a time. Help your team take a step back now and then to evaluate how your current stories fit into the grand scheme of the business. Keep asking yourselves how you can do a better job of delivering real value.

SUMMARY

Testing and quality are the responsibility of the whole team, but testers bring a special viewpoint and unique skills. As a tester, your passion for delivering a product that delights your customers will carry you through the frustrations you and your team may encounter. Don't be afraid to be an agent for continual improvement. Let agile principles and values guide you as you work with the customer and development teams, adding value throughout each iteration.

In this concluding chapter, we looked at seven key factors for successful agile testing:

1. Use the whole-team approach.
2. Adopt an agile testing mind-set.
3. Automate regression testing.
4. Provide and obtain feedback.
5. Build a foundation of core practices.
6. Collaborate with customers.
7. Look at the big picture.