Agile and Scrum for QA

Quality in Every Sprint

THE MODERN QA ENGINEER SKILLSET

Agile and Scrum for QA: Quality in Every Sprint

Sprint Planning, Ceremonies, and the QA Mindset

Yuri Syuganov

Modern QA Course

https://modernQAcourse.com

Agile and Scrum for QA: Quality in Every Sprint

Sprint Planning, Ceremonies, and the QA Mindset

Book 20 of 26 in the series *The Modern QA Engineer Skillset*

First Edition · Version 1.0.0 · 2026-07-11

Set in Charter, IBM Plex Sans, and IBM Plex Mono. Typeset with Pandoc and XeLaTeX.

Modern QA Course · https://modernQAcourse.com · support@modernQAcourse.com

Contents

Introduction

For QA engineers who want to become indispensable members of Agile teams.

About This Book

This book is a practical, hands-on guide to working as a QA engineer in Agile and Scrum environments. It is not a theoretical overview of Agile principles – there are plenty of those already. Instead, this book focuses on the day-to-day reality of being the person responsible for quality on an Agile team: what to say in sprint planning, how to estimate testing effort, when to push back on a story that is not ready, and how to build a culture where quality is everyone's concern.

Every chapter includes realistic scenarios drawn from actual team dynamics, templates you can bring to your next sprint, and exercises that build real skills. By the time you finish this book, you will not just understand Agile QA – you will be able to practice it with confidence.

Who This Book Is For

This book is written for:

- **QA engineers transitioning from waterfall to Agile** who need to understand how their role changes when testing is no longer a separate phase
- **Junior QA engineers joining Agile teams** who want to contribute meaningfully from their first sprint

- **Senior QA engineers looking to level up** from test execution to quality coaching and process leadership
- **Test automation engineers** who want to understand where their automation fits in the bigger Agile picture
- **Development team leads and Scrum Masters** who want to understand how to integrate QA effectively into their teams

Prerequisites

To get the most from this book, you should have:

- Basic understanding of software testing concepts (test cases, defects, test environments)
- Some experience working on a software development team, in any methodology
- Familiarity with common tools like Jira, Git, and CI/CD pipelines (helpful but not required)
- No prior Agile or Scrum experience is needed – we start from the fundamentals

How to Use This Book

Read it cover to cover if you are new to Agile QA. The chapters build on each other, starting with sprint mechanics and progressing to advanced practices.

Use it as a reference if you are already working in Agile. Jump to the chapter that addresses your current challenge – each chapter is self-contained enough to be useful on its own.

Do the exercises. Every chapter ends with hands-on exercises rated by difficulty. The exercises are designed to be done with your actual team on your actual project. Reading about Agile QA is not the same as doing it.

Bring the templates to your team. This book includes templates for Definition of Done, sprint checklists, retrospective formats, estimation guides, and more. These are designed to be used immediately – print them, share them, adapt them.

Complete the capstone project. The final chapter is a capstone project that ties everything together. It is the best way to verify that you have internalized the material.

Difficulty Ratings for Exercises

Throughout this book, exercises are rated on a three-level scale:

- **[Beginner]** – Can be completed individually with minimal team involvement
- **[Intermediate]** – Requires collaboration with your team or changes to your current process
- **[Advanced]** – Requires facilitation skills, organizational influence, or multi-sprint commitment

PART I

FOUNDATIONS – QA IN THE SPRINT

Part I covers the fundamental mechanics of working as a QA engineer in Scrum. You will learn your role in every ceremony, how to define "done" with your team, and how to estimate testing effort so it is never an afterthought.

CHAPTER 1

The Agile Mindset for QA

1.1 Why Agile Changed Everything for QA

The biggest shift Agile brought to QA was timing. In waterfall, testing happened after development was "done." Requirements were written, designs were approved, code was built, and then – often weeks or months later – the testing team received a finished product and began looking for defects. By that point, the cost of fixing any bug was enormous: the code was complete, the developers had moved on to other projects, and the architecture could not easily accommodate changes.

In Agile, testing is woven into every sprint, every story, every conversation. A QA engineer in an Agile team is not a gatekeeper who approves or rejects work at the end. They are a quality advocate who shapes how work is defined, built, and verified from the start.

This distinction matters. It is not just a change in process; it is a change in identity. The waterfall QA engineer asked: "Does this work correctly?" The Agile QA engineer asks: "How do we make sure this works correctly before we even start building it?"

1.2 From Gatekeeper to Quality Coach

The QA role has fundamentally shifted over the past decade. Understanding this evolution helps you position yourself for the modern version of the role and communicate your value effectively.

The Gatekeeper Model (Traditional)

In the traditional model, QA sat at the end of the development pipeline:

```
Developers --> --> --> --> QA --> --> --> --> Release
                           ^
                    "Is it good enough?"
```

The characteristics of the gatekeeper approach were:

- QA sits at the end of the pipeline
- Development "throws code over the wall" to QA
- QA finds bugs and "throws them back"
- Adversarial relationship: developers versus testers
- QA is a bottleneck by design
- Quality is QA's responsibility alone
- Success is measured by bugs found
- Testing happens in a separate "testing phase"

This model failed for several interconnected reasons. Bugs found late are expensive to fix. The adversarial dynamic hurts collaboration. QA becomes the bottleneck for every release. Developers do not learn from bugs because feedback comes too late. QA burns out from the pressure of being the last line of defense. And, most fundamentally, quality is seen as something that can be "tested in" rather than "built in."

The Quality Coach Model (Modern)

In the modern model, QA is embedded in the development team from day one:

```
      QA <-- Reviews requirements
       |
QA + Dev <-- Three amigos, pair testing
       |
 QA + Dev <-- Code review, test review
       |
   QA <-- Exploratory testing, automation
       |
  Team <-- Monitoring, continuous improvement
```

The quality coach approach looks fundamentally different:

- QA is embedded in the development team from the start
- QA helps developers write better tests, not just finds their bugs
- QA contributes to code reviews, architecture decisions, and CI/CD pipeline design
- Quality is everyone's responsibility; QA provides expertise and tooling
- QA focuses on preventing bugs through process improvement, not just detecting them through testing
- Success is measured by bugs prevented, not bugs found
- Testing is continuous, not a phase

What Quality Coaches Do

Upstream activities (before code is written):

Activity	Impact
Review requirements for testability	Catches ambiguous and untestable requirements
Three amigos sessions	Aligns team on expected behavior and edge cases
Define acceptance criteria	Creates clear, testable "done" criteria
Design test strategy for upcoming features	Ensures test infrastructure is ready when code is
Advocate for quality standards (DoD, coding standards)	Sets the quality bar for the team

During development:

Activity	Impact
Review PRs for testability	Catches missing test hooks, untested paths

continued on next page

Activity	Impact
Pair test with developers	Finds bugs during implementation, not after
Write and maintain automated tests	Builds the safety net
Monitor CI pipeline health	Keeps the feedback loop fast and reliable

After development:

Activity	Impact
Exploratory testing	Finds issues automation misses
Quality reporting	Makes quality visible to stakeholders
Retrospective contributions	Drives continuous quality improvement
Mentor developers on testing	Multiplies quality expertise across the team

1.3 What Differentiates Great QA Engineers

The best QA engineers in modern teams are not the ones who find the most bugs. They are the ones whose teams produce the fewest bugs, because they have built systems, processes, and culture that prevent defects from being created in the first place.

Good QA Engineer	Great QA Engineer
Finds bugs in testing	Prevents bugs in requirements
Writes tests after code	Writes test strategy before code
Reports on pass/fail	Reports on quality trends and risk
Uses test management tools	Builds testing into the development workflow

continued on next page

Good QA Engineer	Great QA Engineer
Works alone on testing	Coaches the team on quality practices
Reacts to flaky tests	Builds infrastructure that prevents flakiness
Tests what they are told to test	Identifies what needs testing and what does not

1.4 Common Misconceptions About Modern QA

Before we dive into the mechanics, let us address some misconceptions that can hold QA engineers back:

Misconception	Reality
"QA is being replaced by automation"	Automation replaces repetitive execution, not quality thinking
"Developers can do all the testing"	Developers are great at unit tests; QA brings different perspectives and skills
"AI will replace QA engineers"	AI amplifies QA capabilities but cannot replace judgment and creativity
"QA is less important in Agile"	QA is more important – quality must be built into every sprint, not bolted on
"If you can code, you should be a developer"	QA engineers who can code are the most valuable team members

> **Pro Tip:** When someone says "we don't need QA, developers can test their own code," respond with: "Developers are excellent at verifying that their code does what they intended. QA verifies that what they intended is what the user actually needs, and discovers what happens when users do things nobody intended." These are fundamentally different skills.

1.5 Skills of the Modern QA Engineer

The skill set has expanded significantly from the traditional QA role:

Technical Skills:

- Test automation: not just writing tests, but designing test frameworks
- CI/CD pipelines: configuring, optimizing, and troubleshooting
- Programming: strong enough to write maintainable, well-structured code
- API testing: direct API interaction, contract testing
- Performance testing: load testing, profiling, budget enforcement
- Infrastructure: Docker, cloud services, monitoring tools

Process Skills:

- Agile practices: sprint ceremonies, estimation, continuous improvement
- Risk assessment: identifying what to test and what to skip
- Communication: reporting quality to different audiences
- Mentoring: teaching developers about testing best practices
- Collaboration: working with every role on the team

Strategic Skills:

- Test strategy: deciding what types of testing to invest in
- Quality metrics: measuring and reporting on quality trends
- Process improvement: identifying and fixing quality bottlenecks
- Tooling decisions: evaluating and implementing test tools
- AI integration: leveraging AI for test generation, prioritization, and analysis

1.6 Building a Quality Culture

The ultimate goal of a quality coach is to build a culture where quality is everyone's concern:

- Developers write tests because they value the safety net, not because QA forces them
- Product owners define testable criteria because they understand it leads to better outcomes
- Managers invest in test infrastructure because they see the ROI in fewer production incidents
- The team celebrates prevention (a sprint with zero escaped defects) as much as delivery (features shipped)

This culture does not happen overnight. It is built one sprint at a time, one conversation at a time, one demonstrated success at a time. The rest of this book gives you the tools to build it.

> **Common Mistake:** Trying to change your team's quality culture all at once. Introducing a 15-item Definition of Done, mandatory Three Amigos sessions, and a new test framework in the same sprint will overwhelm your team and create resistance. Pick one improvement per sprint. Demonstrate value. Then introduce the next one.

Self-Assessment Quiz: Chapter 1

1. What is the primary difference between the gatekeeper model and the quality coach model?
2. Name three upstream activities a QA engineer should perform before code is written.
3. Why does finding bugs late in the development process cost more than finding them early?
4. What is the key metric that differentiates a good QA engineer from a great one?
5. Name two misconceptions about QA in Agile and explain why they are wrong.

Answers:

1. The gatekeeper sits at the end of the pipeline and judges finished work; the quality coach is embedded in the team from the start and prevents defects through collaboration, process improvement, and early testing activities.
2. Any three of: review requirements for testability, conduct Three Amigos sessions, define acceptance criteria, design test strategy for upcoming features, advocate for quality standards.
3. Late-stage bugs require more rework – the code is already written, integrated, and potentially deployed. A requirement caught in review costs minutes to fix; the same issue in production can cost thousands of dollars and damage customer trust.
4. Bugs prevented, not bugs found. Great QA engineers build systems and processes that stop defects from being created.
5. Any two from the misconceptions table, with explanations matching the "Reality" column.

Exercises: Chapter 1

[Beginner] Exercise 1.1: Assess your current role. Make two columns: one for gatekeeper activities you currently do, and one for quality coach activities. Which column is longer? Identify one activity from the coach column you could start doing this sprint.

[Beginner] Exercise 1.2: Write a one-paragraph description of your role as a QA engineer using the quality coach framing. Focus on prevention, collaboration, and continuous improvement rather than finding bugs.

[Intermediate] Exercise 1.3: Propose one process change to your team that would shift testing earlier in the workflow. Write a brief pitch (3-5 sentences) explaining the change and its expected benefit.

[Advanced] Exercise 1.4: Track the "bugs found by phase" metric for your next three sprints. Categorize each bug by when it was found: requirements, development, testing, or production. Is the distribution shifting left over time?

Career Translation

Resume phrasing

- Championed the transition from gatekeeper-style QA to an embedded quality coach model, reducing escaped defects by 40% within four sprints.
- Led adoption of upstream quality practices (requirements review, acceptance criteria refinement) that cut average defect fix cost by 60%.
- Designed and delivered QA onboarding materials that shortened new-hire ramp-up from 6 weeks to 3 weeks.

Cover letter framing

I approach quality as a team-wide discipline rather than a downstream phase. By embedding QA thinking into how requirements are written, how stories are estimated, and how code is reviewed, I consistently help teams prevent defects rather than simply detect them. This upstream focus translates directly into faster delivery cycles and fewer production incidents, which is the kind of business outcome I aim to deliver.

Interview framing

"I approach quality by positioning myself as a coach rather than a gatekeeper. In practice, that means I am active before any code is written – reviewing requirements for ambiguity, pushing for testable acceptance criteria, and making sure edge cases surface early. I optimize for defect prevention over defect detection, because every bug caught in a requirements review saves an order of magnitude more time than one caught in testing. The trade-off is that upstream involvement requires strong communication skills and the willingness to slow a planning session down when something is not clear, but teams quickly see the payoff in fewer late-sprint surprises."

What not to say

- "I find bugs" – Reduces your value to a single, narrow activity. Hiring managers want prevention, not just detection.

- "I make sure developers write good code" – Sounds adversarial and implies policing rather than coaching.
- "I run test cases" – Positions you as an executor, not a thinker. Nobody hires senior QA for execution alone.
- "QA is the last line of defense" – Reinforces the outdated gatekeeper model and signals you have not internalized shift-left thinking.
- "I am detail-oriented" – Generic, overused, and reveals nothing about how you actually improve quality.

Interview Depth Check

Question 1

Prompt: Your new team has no formal QA process. Developers test their own code and bugs are reported by users. The engineering manager says, "We move fast – adding QA will slow us down." How do you respond, and what do you implement first?

What a strong answer should cover:

- Acknowledgment that speed matters – you are not there to add bureaucracy
- Data-driven framing: escaped defects cost more than prevention
- A phased introduction plan (not everything at once)
- Starting with something that delivers immediate, visible value (e.g., a lightweight DoD or requirements review)

Example answer:

- I would first agree that velocity matters and clarify that my goal is to increase sustainable speed, not add process overhead.
- I would gather data on production incidents from the last month – their severity, time to resolve, and customer impact – to quantify the cost of the current approach.
- My first action would be proposing a 4-item starter Definition of Done (code review, existing tests pass, no critical bugs, deployed to staging) because it is lightweight, team-owned, and immediately prevents the most expensive category of bugs.

- Within 2-3 sprints, I would show the before-and-after escaped defect rate to demonstrate that quality investment accelerates delivery rather than slowing it.

Question 2

Prompt: Describe the difference between the gatekeeper model and the quality coach model. Give a concrete example of how the same situation is handled differently in each model.

What a strong answer should cover:

- Clear articulation of the philosophical shift (testing phase vs. continuous quality)
- A specific, realistic scenario (not abstract theory)
- Measurable differences in outcome

Example answer:

- In the gatekeeper model, QA receives finished code, tests it, and either approves or rejects it. Quality is QA's responsibility alone, and the relationship with developers is often adversarial.
- In the quality coach model, QA is embedded from the start – reviewing requirements, pairing with developers, and building quality into the process rather than inspecting it at the end.
- Concrete example: A developer marks a story as "done." In the gatekeeper model, QA finds 5 bugs and sends it back, frustrating both sides. In the coach model, QA participated in the Three Amigos session, reviewed the PR during development, and paired on complex edge cases – so by the time the story reaches final testing, there are 0-1 minor issues instead of 5, and the developer feels supported rather than policed.

Question 3

Prompt: A developer on your team says, "We do not need QA. Developers can test their own code." How do you respond without being defensive?

What a strong answer should cover:

- Agreement that developers should test their own code (unit tests, code review)
- Explanation of what QA adds beyond developer self-testing (different perspective, risk assessment, cross-feature regression)
- A non-confrontational tone that invites collaboration

Example answer:

- I would agree wholeheartedly that developers should write thorough unit tests and verify their own work – that is a baseline expectation on any strong team.
- Then I would explain the complementary value QA brings: developers verify that code does what they intended, while QA verifies that what they intended is what the user actually needs, and discovers what happens when users do things nobody intended.
- I would offer a concrete example: "Last sprint, exploratory testing found that the checkout flow crashed when a user applied a coupon to an empty cart – a scenario no unit test covered because nobody thought of it. That is the gap QA fills."
- The framing is additive, not competitive: QA does not replace developer testing, it covers the blind spots that are invisible from inside the implementation.

Question 4

Prompt: How do you measure whether you are an effective QA engineer? What metrics matter and what metrics are misleading?

What a strong answer should cover:

- Prevention-focused metrics (escaped defects, bugs by phase) over volume-focused metrics (bugs found, test cases executed)
- Recognition that "bugs found" can be a perverse incentive
- Business-aligned metrics (incident frequency, customer-reported defects, release confidence)

Example answer:

- The most important metric is escaped defects – bugs that reach production. If that number is trending down, quality is improving.
- I also track defect distribution by phase: what percentage are caught in requirements review, during development, in testing, and in production. A healthy shift-left shows more caught early and fewer caught late.
- Misleading metrics include "bugs found" (a high count could mean bad quality or good testing – it is ambiguous) and "test cases executed" (quantity says nothing about effectiveness).
- I complement these with business metrics: production incident frequency, mean time to recovery, and whether stakeholders feel confident enough to release without demanding extra manual regression cycles.

About This Sample

You have just read **Chapter 1 of Agile and Scrum for QA: Quality in Every Sprint** — the free sample. The complete book continues with every remaining chapter, the interview Q&A, and all of the code.

Get the full book

Sample readers get a discount at
https://leanpub.com/modernqa-20-agile-scrum-qa/c/SAMPLE10

Read online first

Chapter one of **every** book in the series is free to read at
https://modernqacourse.com/library

About the Author

Yuri Syuganov is a software quality engineer with more than twenty years of hands-on testing experience, spanning the arc from manual test design through Selenium-era automation to today's AI-augmented, agent-driven frameworks. He is the author of The Modern QA Engineer Skillset, a 26-book series built on a blunt analysis of what companies actually expect from QA engineers, and the creator of modernQAcourse.com, where the course behind the series is free and readers can bring real testing problems to experienced practitioners. He revises the series as the industry changes — because it does.

Also in This Series

The Modern QA Engineer Skillset — a 26-book course in modern software quality assurance.

AI-Augmented QA (Books 1–10)

- **Book 1:** Agent Skills for Browser Automation: The Complete Guide
- **Book 2:** AI-Augmented Test Design: From Prompts to Test Suites
- **Book 3:** Agentic Testing Architectures: Building Autonomous QA Systems
- **Book 4:** API and Contract Testing with AI: Schema to Shield
- **Book 5:** Performance and Chaos Engineering: Breaking Things on Purpose
- **Book 6:** Observability-Driven Testing: Testing in Production
- **Book 7:** Security Testing for AI Applications: The New Attack Surface
- **Book 8:** Infrastructure as Code Testing: Terraform to Kubernetes
- **Book 9:** Mobile and Cross-Platform Testing: 24,000 Devices, One Strategy
- **Book 10:** Visual and Accessibility Testing: Pixels and People

Technical Foundations (Books 11–19)

- **Book 11:** Manual Testing Fundamentals: The Bedrock
- **Book 12:** Programming for QA: Python, JavaScript, and TypeScript from Zero to Fluent
- **Book 13:** Browser Test Automation: Playwright, Patterns, and the Death of Fragile Tests

- **Book 14:** API Testing Fundamentals: Postman to Pytest
- **Book 15:** SQL and Database Testing: Query Your Way to Quality
- **Book 16:** CI/CD Pipelines: From Commit to Confidence
- **Book 17:** Git and Version Control: Beyond Push and Pull
- **Book 18:** Test Management Tools: Making Quality Visible
- **Book 19:** Linux and Command Line: Your Production Survival Kit

Professional Skills (Books 20–26)

- **Book 20:** Agile and Scrum for QA: Quality in Every Sprint
- **Book 21:** Communication and Stakeholder Management for QA: Influence Without Authority
- **Book 22:** Test Strategy and Quality Metrics: Measuring What Matters
- **Book 23:** QA Leadership and Mentoring: Multiplying Your Impact
- **Book 24:** Technical Writing for QA: Documents That Drive Action
- **Book 25:** Interview Preparation and Career Strategy: Land the Job
- **Book 26:** Testing Like a Senior: Patterns & Anti-Patterns Across Software QA Career Levels

Visit https://modernQAcourse.com for the full course catalog, updates, and companion resources.

Stuck on a Real Problem?

The full course behind this series is free to read at https://modernQAcourse.com — and it stays free.

When a chapter meets the real world and something breaks, post it on the **Help Board** at https://modernQAcourse.com. Asking is free, and experienced practitioners answer right in the thread. Need hands-on help? Book a 1:1 screen-share session with a QA expert.