Agile Testing by Lisa Crispin

Agile Testing – A Practical Guide for Testers and Agile Teams – Lisa Crispin and Janet Gregory

Contents

Short Summary

My **top 3** and 6 takeaways from the book were:

1. **The four quadrants of testing is the number one takeaway (Chapters 6, 7, 8, 9, 10, 11, 12). It explains all the different types of testing. You can find a description of the testing on a 2 x 2 of business facing v technology facing and critiquing the products vs supporting the team. The testing types we're familiar with are: (1) Q1 Tests - Unit tests and component tests, (2) Q2 Tests - Functional Tests, (3) Q3 Tests - UAT and (4) Q4 Tests - Performance, load, security and ility tests. We're not so familiar with (5) Q2 Tests - Examples, story tests, prototypes, simulations and (6) Q3 Tests - Exploratory testing, scenarios, usability testing and Alpha / Beta.**
2. **How you organise your team is super important. Need to balance Engineering with Business with Testing, otherwise known as the three amigos. (Chapter 4)**
3. **Measuring success is key including Defect tracking, Test planning, CMMI, ITIL, Documentation, System Test, Release to Prod (Chapter 5)**
4. Automation which generally gets harder before it gets easier. (Chapter 13, 14, and 15)
5. The importance of unit / component testing and catching bugs early. (Chapter 18)

6. Release timelines with UAT and with extended UAT where testers start working on next release whilst other testers finish UAT / Acceptance testing. (Chapter 20)

Interim step / General testing

This table shows:

1. Name of the test phase,
2. whether it is functional or non-functional
3. Description of what is tested by this phase

4. Description of when this testing is done in a fortnightly release cadence
5. Description of who does the testing.

Type of testing	**Functional / Non-Functional**	**Description**	**Done when in a fortnightly release cadence?**	**Done by who?**
1. Unit / component testing	Functional	Testing individual components or modules	Test driven development – Done before coding.	Developers
2. Integration testing	Functional	Testing the combined parts of the system	Once individual units are tested, integration testing can be done. Towards end of the first week. Or beginning of second week.	Developers and QA / Testers
3. System testing	Functional	Testing the entire system as a whole	Following integration testing. In second week.	QA / Testers
4. Acceptance testing	Functional	Ensuring the system meets the requirements of end users	Towards the end of the second week. Allows stakeholders to verify that the system meets their requirements before release	Testers + Business / BA + Engineers
5. Regression testing	Functional	Testing to ensure that changes have not affected existing funcationalit8ies	Performed throughout. Done intensively towards the end of the second week.	QA / Testers
6. Performance testing	Non-Functional	Assessing system performance under various conditions	Intermittently. More comprehensively towards the end of the release.	QA / Testers

Type of testing	Functional / Non-Functional	Description	Done when in a fortnightly release cadence?	Done by who?
7. Security testing	Non-Functional	Evaluating the system security feature and vulnerabilities	Intermittently. More comprehensively towards the end of the release.	QA / Testers
8. Usability testing	Non-Functional	Assessing the user friendliness and ease of use of the system.	Intermittently. More comprehensively towards the end of the release.	QA / Testers

Detailed Summary

Contents

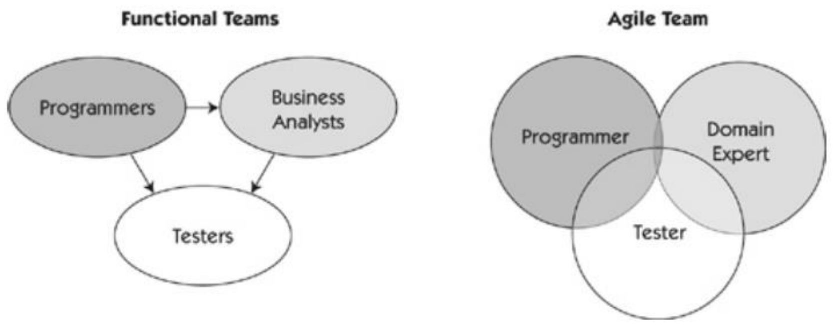
Summary of book

- Part I. Introduction
    - Chapter 1 What is agile testing, anyway?
        - Agile Manifesto / Manifesto for agile software development
            - We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
                - Individuals and interactions over process and tools
                - Working software over comprehensive documentation
                - Customer collaboration over contract negotiation
                - Responding to change over following a plan
            - That is while there is value in the items on the right, we value the items on the left more.
        - This book isn't about unit testing or component testing, although this is critical to the success of a project.
        - Roles in an agile team
            - Customer team
            - Developer team
            - Interaction between customer and developer teams
            - Programmer v Domain expert v Tester
        - Traditional / waterfall testing:
            - Requirements
            - Specifications
            - Code
            - Testing
            - Release
        - Agile has a whole team approach. This means everyone is involved.
    - Chapter 2 Ten principles for agile testers
        - 10 Principles
            - Provide continuous feedback
            - Deliver value to the customer
            - Enable face to face communication
            - Have courage – this is a core value in XP.
                - Developers have courage to make changes and refactor the code base because they have the safety net of an automated regression suite.
            - Keep it simple
                - Again from XP. The best thing to do is the simplest thing you possibly can.
            - Practice continuous improvement
            - Respond to change

            - Self organise
            - Focus on people
            - Enjoy
        - Adding value – The 10 values bring business value. In agile development, the whole team takes responsibility for delivering high quality software that delights customers and makes the business more profitable.
            - Team members may wear many hats
            - Agile testers not only think about the system from the viewpoint of stakeholders who will live with the solution but they also have a grasp of technical constraints and implementation details that face the development team.
            - Peril – you're not really part of the team – if you're a tester and you're not invited to attend planning sessions, stand-ups or design meetings, you might be in a situation where testers are viewed as somehow part from the development team. If this is the case, the team is at risk and this should be addressed.
        - An agile testing mindset is customer focused, results oriented, craftsman like, collaborative, creative, eager to learn and passionate about delivering business value in a timely manner
        - Attitude is important and it blurs the lines between testers, programmers and other roles on an agile team.
        - Agile testers apply agile values and principles such as feedback, communication, courage, simplicity, enjoyment and delivering value to help the team identify and deliver the customer requirements for each story
        - Agile testers add value to their teams and their organisation with their unique viewpoint and team oriented approach
- Part II. Organisational challenges
    - Chapter 3. Cultural challenges
        - Quality philosophy – what is an acceptable level of software quality
            - Peril – Quality policy mentality. There are two issues with this
                - Quality folks nit pic over everything including coding standards.
                - Team relies on the quality folks as a safety net and don't build quality in to their development.
                - Ways to avoid this peril:
                    - Whole team ownership of quality
                    - Skills and adaptability
                    - Factors that help
            - Sustainable pace
            - Customer relationships
            - Organization size
                - Communication challenges
                - Conflicting cultures within the organisation

    - Advanced planning
    - Act now, apologise later
    - Empower your team
- Barriers to successful agile adoption by test / qa teams
    - Loss of identity
    - Additional roles
    - Lack of training
    - Not understanding agile concepts
    - Past experience / attitude
    - Cultural differences among roles
- Introducing change
    - Talk about fears
    - Give team ownership
    - Celebrate success
- Management expectations
    - Cultural changes for managers
    - Speaking the managers language
- Change doesn’t come easy
    - Be patient
    - Let them feel pain
    - Build your credibility
    - Work on your own professional development
    - Beware the quality police mentality
    - Vote with your feet
- Summary
    - Consider organisation culture before making any kind of change
    - Testers have an easier time integrating into agile teams when their whole organisation values quality, but testers with a ‘quality police’ mindset will struggle
    - Some testers might have trouble adjusting to the whole team ownership of quality
    - Customer teams and developer teams must work closely together
    - Large organisation s that tend to have more isolated specialist teams face particular cultural challenges in areas such as comms and collab
    - Major barriers to success for testers for agile adoption include fear, loss of identity, lack of training, previous negative experiences with new development process and cultural differences among roles

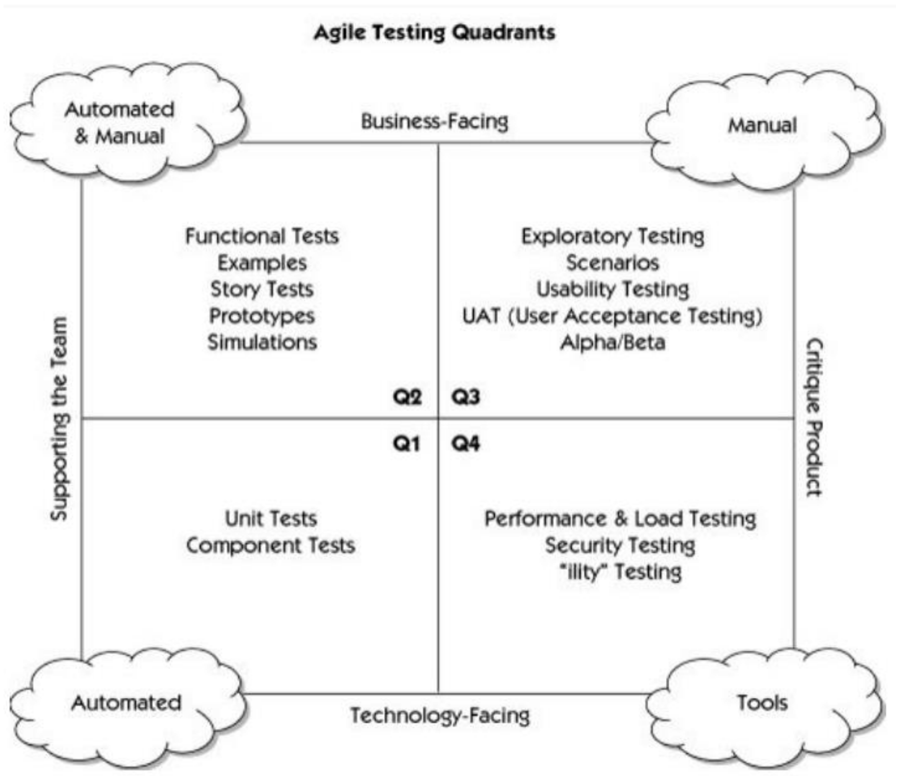
        - To help introduce change and promote communication, we suggest encouraging team members to discuss fears and celebrating every success, no matter how small
        - Guidelines such as a ‘tester bill of rights’ give testers confidence to raise issues and help them feel safe as they learn and try new ideas
        - Managers face their own cultural challenges and they need to provide support and training to help testers success on agile teams
        - Testers can help teams accommodate manager expectations by providing the information managers need to track progress and determine ROI
          change doesn’t come easy so be patient and work on improving your own skills wo you can help your team
- Chapter 4. Team Logistics
    - Team structure
        - Independent QA teams
        - Integrating testers into agile project
        - Agile project teams



    - Physical logistics
    - Resources
        - Tester developer ration
        - Hiring an agile tester
    - Building a team
        - Self organising teams
        - Involving other teams
        - Every team member has equal value
        - Performance and rewards
        - What you can do
    - Summary
        - Consider the importance of team structure; while testers might need an independent mindset, putting them on a separate team can be counterproductive,.
        - Testers need access to a larger community of testers for learning and trying out new ideas. QA teams might be able to create this community within their organisation.

            - It is important for the whole team to be located together, to foster collaboration; if the team is distributed, provide tools to promote communication
            - Hire for attitude
            - There is no right tester / developer ratio. The right answer is ‘it depends on your situation’
            - Teams need to self organise, identify and find solutions to their own problems and look for ways to improve. They can’t wait for someone to tell them what to do.
            - Management should reward performance in a way that promotes the teams effort to deliver business value but not penalize good individual performance if the team is struggling
            - Testers can use agile principles to improve their own skills and increase their value to the team. They need to be proactive and find ways that they can contribute.
    - Chapter 5. Transitioning typical processes
        - Seeking lightweight processes
        - Metric
            - Lean measurements
            - Why do we need metrics
            - What not to do with metrics
            - Communicating
            - Metrics ROI
        - Defect tracking
            - Why use a defect tracking system (DTS)
            - Why not
            - Defect tracking tools
        - Test planning
            - Test strategy vs test plan
            - Traceability
        - Existing processes
            - Audits
            - Frameworks, models and standards
                - Capability maturity model integration (CMMI) – helps organisations improve their process but doesn’t dictate specific development practices to accomplish the improvements.
                - Information technology infrastructure library (ITIL) is a set of best practices for IT service management intended to help organisations develop an effective quality process
                - Documenting the test strategy:
                    - Project initiation – get an understanding of the project

            - Release planning – participate in sizing stories, create test plans
            - Each iteration – sprint planning, estimation, write and execute story tests, pair test with other testers, developers. Business validation. Automate new functional test cases. Run automated regression test cases. Run project load tests. Demo to the stakeholders
        - End Game (System Test) – Release management tests, mock deploy on staging, smoke test on staging, perform load test, complete regression test, business testers perform UAT, participate in release readiness
        - Release to prod / support – participate in release to production. Participate in retrospectives.
    - Summary
        - The right metrics can help you to make sure your team is on track to achieve its goals and provide a good return on your investment in them.
        - Metrics should be visible, providing necessary milestones upon which to make decisions
        - The reasons to use a defect tracking system include for convenience, for use as a knowledge base and for traceability
        - Defect tracking systems are too often used as a communication tool and entering and tracking unnecessary bugs can be considered wasteful
        - All tools including the DTS need to be used by the whole team, so consider all perspectives when choosing a tool
        - A test strategy is a long term overall test approach that can be put in a static document; a test plan should be unique to the project
        - Think about alternative before blindly accepting the need for specific documents. For example the agile approach to developing in small, incremental chunks, working closely together might remove the need for formal traceability documents. Linking the source code control system comments to tests might be another way.
        - Traditional quality processes and process improvement models such as SAS 70 audits and CMMI standards, can coexist with agile development and testing. Teams need to be open to thinking outside the box and work together to solve their problems.
- Part III. The agile testing quadrants
    - Chapter 6. The purpose of testing
        - Overview of quadrants

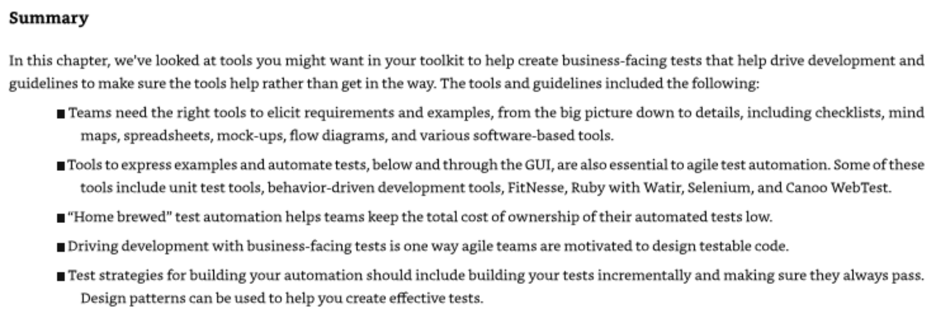


        - Tests that support the team
        - Tests that critique the product
    - Knowing when we're done
        - Shared responsibility
        - Fitting all types into doneness
    - Managing technical debt
    - Context driven
    - Summary
        - Tests that support the team can be used to derive requirements
        - Tests that critique the product help us think about all facets of application quality
        - Use the quadrant to know when you're done, and ensure the whole team shares responsibility for covering the four quadrants of the matrix
        - Manging technical debt is an essential foundation for any software development team. Use the quadrant to think about the different dimensions
        - Context should always guide our testing efforts.
- Chapter 7. Technology Facing tests that support the team
    - Foundation of agile testing
        - Purpose of unit tests
        - Supporting infrastructure
    - Why these tests
        - Go faster do more
        - Value to testers

    - Designing for testing
    - Timely feedback
- Quadrant 1 tests vs quadrant 2
    - Where does one stop and the other start
- What if you don’t do them
    - What testers can do
    - What managers can do
    - Team approach
- Toolkit
    - Source code control
    - IDEs
    - Build tools
    - Build automation tools
    - Unit test tools
- Summary
    - Technology facing tests that support programming let the team produce the highest quality code possible; they form the foundation for all other types of testing
    - The benefits of this quadrants tests include going faster and doing more, but speed and quantity should never be the ultimate goal
    - Programmers write technology facing tests that support the team and provide great value to testers by enhancing the internal quality and testability of the system
    - Teams that fail to implement the core practices related to agile development are likely to struggle
    - Legacy systems usually present the biggest obstacle to test driven development, but these problems can be overcome with incremental approaches.
    - If your team doesn’t now do these tests, you can help them get started by engaging other team members and getting support from management
    - There can be some overlap between technology facing tests and business facing tests that support the team. However when faced with a choice, push tests to the lowest level in order to maximise ROI
    - Teams should set up continuous integration, build and test processes in order to provide feedback as quickly as possible
    - Agile teams acquire tools for tasks such as source code control, test automation, IDEs and build management to facilitate technology facing tests that support the team.
- Summary

    - In agile development, examples and business facing tests, rather than traditional requirements documents, tell the team what code to write
    - Working on thin slices of functionality, in short iterations, gives customers the opportunity to see and use the application and adjust their requirements as needed
    - An important area where testers contribute is helping customers express satisfaction conditions and create examples of desired, and undesired behaviour for each story
    - Ask open ended questions to help the customer think of all the desired functionality and to prevent hiding important assumptions
    - Help the customer achieve consensus on desired behaviour for stories that accommodate the various viewpoints of different parts of the bushiness
    - Help customers develop tools (e.g. a story checklist) to express information such as business satisfaction conditions
    - The development and customers teams should think through all the parts of the application that a given story affects, keeping the overall system functionality in mind.
    - Work with your team to break feature sets into small, manageable stories and paths within stories
    - Follow a pattern of 'write test – write code – run tests – learn' in a step by step manner building on each pass through the functionality
    - Use tests and examples to mitigate risks of missing functionality or losing sight of the big picture
    - Driving code with business facing tests makes the development team constantly aware of the need to implement a testable application
    - Business facing tests that support the team must be automated for quick and easy feedback so that teams can deliver value in short iterations.
- Chapter 8. Business facing tests that support the team
  - Driving development with business facing tests
  - The requirements quandary
    - Common language
    - Eliciting requirements
    - Advance clarity
    - Conditions of satisfaction
    - Ripple effect
    - Customer availability
  - Thin slices, small chunks
  - Knowing when we're done
  - Test mitigate risks
    - Post conditions

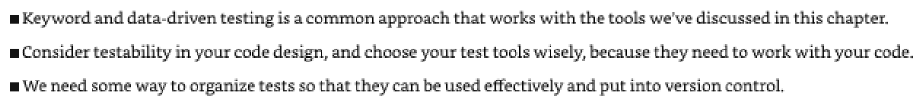
        - Peril: Forgetting the big picture
    - Testability and automation
- Chapter 9. Toolkit for business facing tests that support the team
    - Business facing tool strategy introduction
    - Tools to elicit examples and requirements
        - Checklists
        - Mind maps
        - Spreadsheets
        - Mock-ups
        - Flow diagrams
        - Software based tools
    - Tools for automating tests based on examples
        - Below the GUI / API
        - Using the GUI
        - Home brewed frameworks
    - Strategies for writing tests
        - Build tests incrementally
        - Keep the tests passing
        - Design patterns
        - Keyword and data driven tests
    - Testability
        - Code design and test
        - Automated vs manual tests
    - Test Management
    - Summary:



Summary

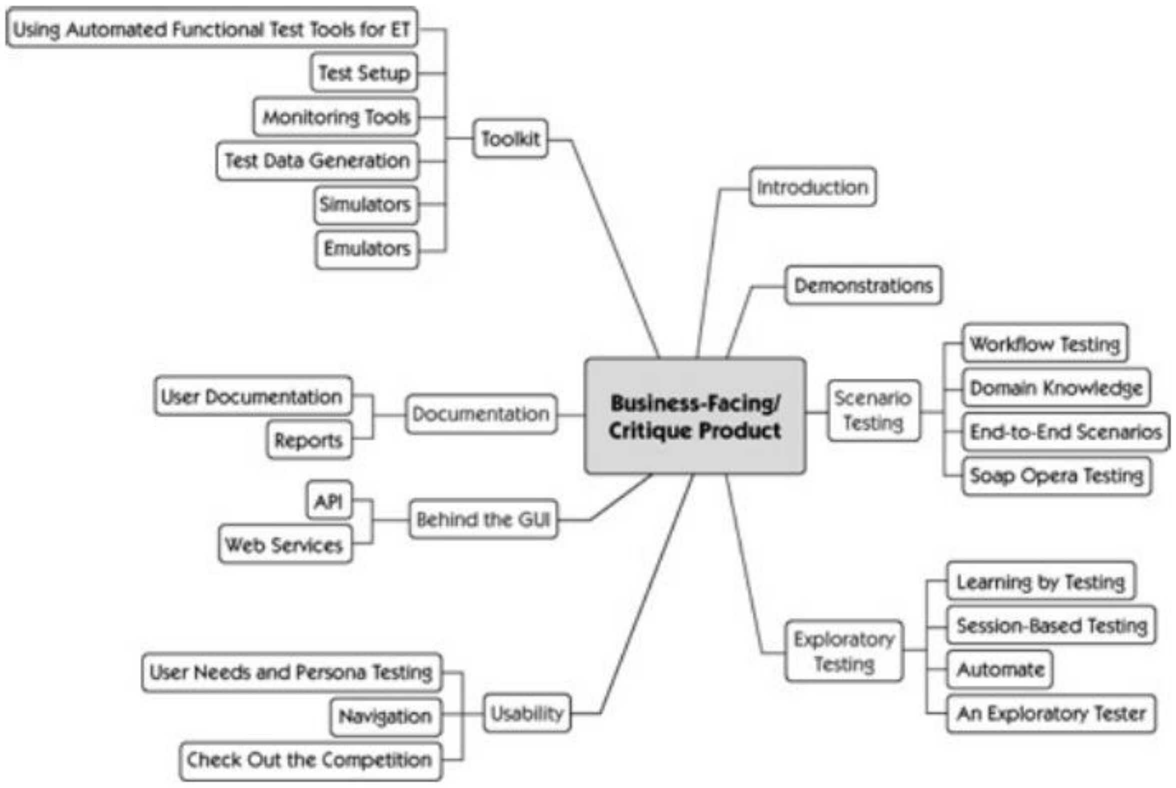
In this chapter, we've looked at tools you might want in your toolkit to help create business-facing tests that help drive development and guidelines to make sure the tools help rather than get in the way. The tools and guidelines included the following:

- Teams need the right tools to elicit requirements and examples, from the big picture down to details, including checklists, mind maps, spreadsheets, mock-ups, flow diagrams, and various software-based tools.
- Tools to express examples and automate tests, below and through the GUI, are also essential to agile test automation. Some of these tools include unit test tools, behavior-driven development tools, FitNesse, Ruby with Watir, Selenium, and Canoo WebTest.
- "Home brewed" test automation helps teams keep the total cost of ownership of their automated tests low.
- Driving development with business-facing tests is one way agile teams are motivated to design testable code.
- Test strategies for building your automation should include building your tests incrementally and making sure they always pass. Design patterns can be used to help you create effective tests.



- Keyword and data-driven testing is a common approach that works with the tools we've discussed in this chapter.
- Consider testability in your code design, and choose your test tools wisely, because they need to work with your code.
- We need some way to organize tests so that they can be used effectively and put into version control.

- Chapter 10. Business facing tests that critique the product



■

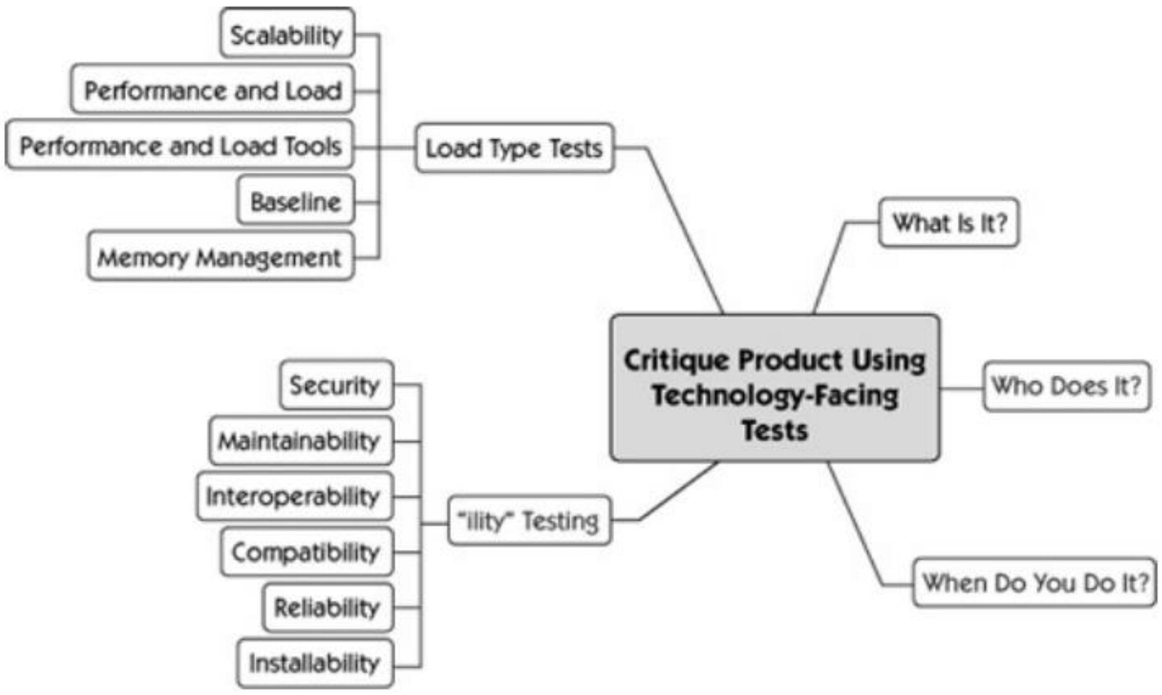
Summary

A large part of the testing effort is spent critiquing the product from a business perspective. This chapter gave you some ideas about the types of tests you can do to make your testing efforts more effective.

- ■ Demonstrate software to stakeholders in order to get early feedback that will help direct building the right stuff.
- ■ Use scenarios and workflows to test the whole system from end to end.
- ■ Use exploratory testing to supplement automation and to take advantage of human intellect and perceptions.
- ■ Without usability in mind when testing and coding, applications can become shelfware. Always be aware of how the system is being used.
- ■ Testing behind the GUI is the most effective way of getting at the application functionality. Do some research to see how you can approach your application.
- ■ Incorporate all kinds of tests to make a good regression suite.
- ■ Don't forget about testing documentation and reports.
- ■ Automation tools can perform tedious and repetitive tasks, such as data and test scenario setup, and free up more time for important manual exploratory testing.
- ■ Tools you're already using to automate functional tests might also be useful to leverage exploratory tests.
- ■ Monitoring, resource usage, and log analysis tools built into operating systems and IDEs help testers appraise the application's behavior.
- ■ Simulators and emulators enable exploratory testing even when you can't duplicate the exact production environment.
- ■ Even when tests are used to drive development, requirements for desired behavior or interaction with other systems can be missed or misunderstood. Quadrant 3 activities help teams keep adding value to the product.

■

○ Chapter 11. Critiquing the product using technology facing tests

▪ 

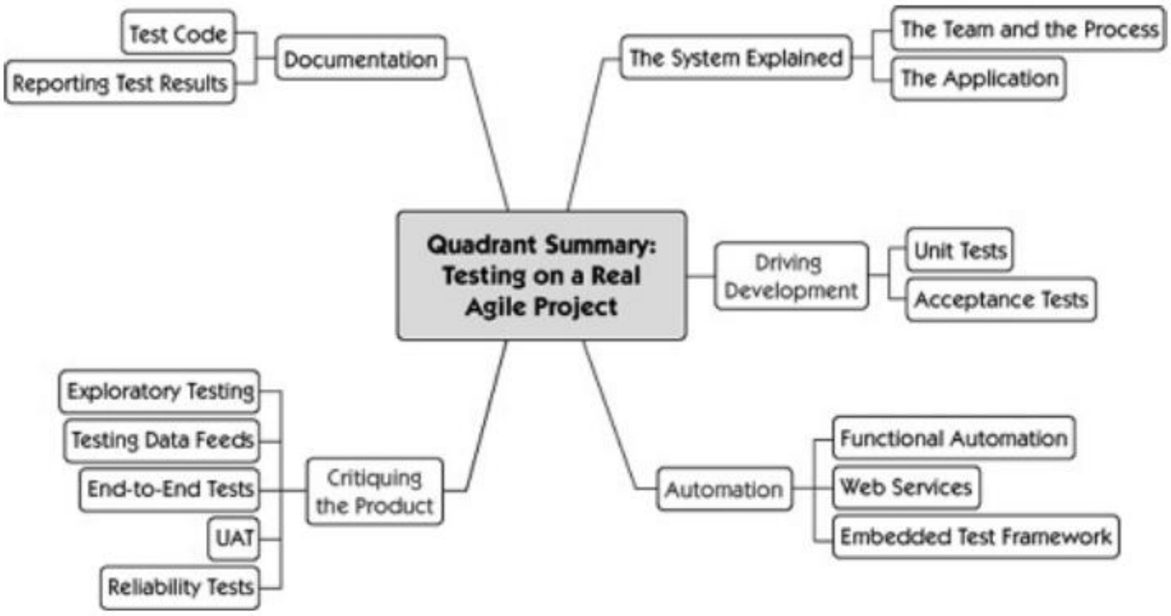
▪ **Summary**

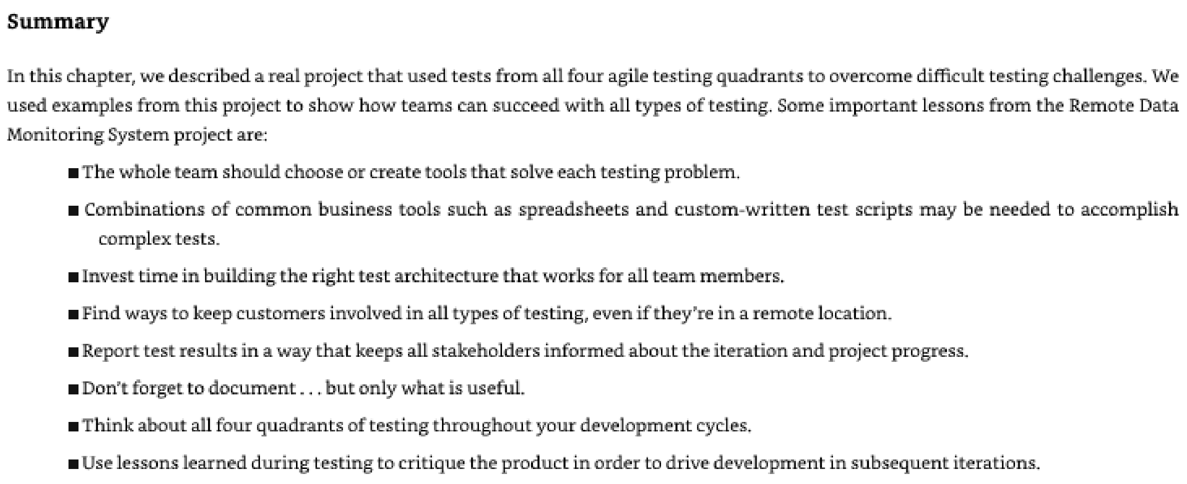
In this chapter, we've explored the fourth agile testing quadrant, the technology-facing tests that critique the product.

- The developer team should evaluate whether it has, or can acquire, the expertise to do these tests, or if it needs to plan to bring in external resources.
- An incremental approach to these tests, completing tasks in each iteration, ensures time to address any issues that arise and avoid production problems.
- The team should consider various types of "ility" testing, including security, maintainability, interoperability, compatibility, reliability, and installability testing, and should execute these tests at appropriate times.
- Performance, scalability, stress, and load testing should be done from the beginning of the project.
- Research the memory management issues that might impact your product, and plan tests to verify the application is free of memory issues.

▪

○ Chapter 12. Summary of testing quadrants

▪ 

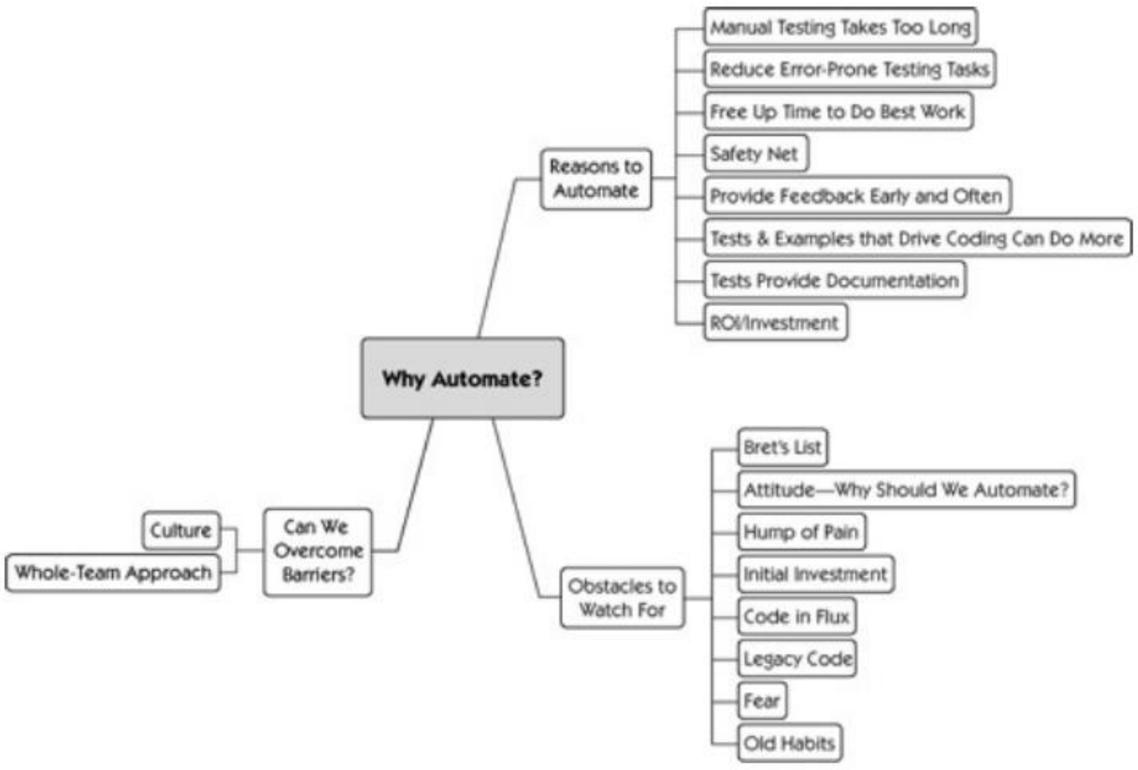


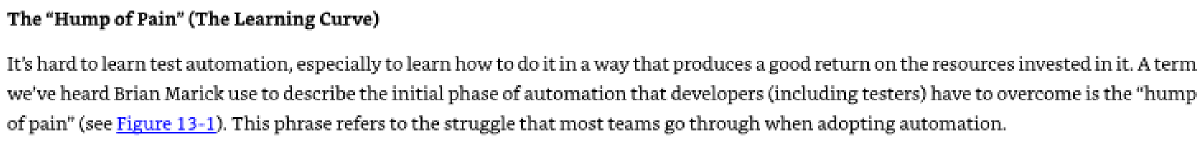
Summary

In this chapter, we described a real project that used tests from all four agile testing quadrants to overcome difficult testing challenges. We used examples from this project to show how teams can succeed with all types of testing. Some important lessons from the Remote Data Monitoring System project are:

- The whole team should choose or create tools that solve each testing problem.
- Combinations of common business tools such as spreadsheets and custom-written test scripts may be needed to accomplish complex tests.
- Invest time in building the right test architecture that works for all team members.
- Find ways to keep customers involved in all types of testing, even if they're in a remote location.
- Report test results in a way that keeps all stakeholders informed about the iteration and project progress.
- Don't forget to document . . . but only what is useful.
- Think about all four quadrants of testing throughout your development cycles.
- Use lessons learned during testing to critique the product in order to drive development in subsequent iterations.

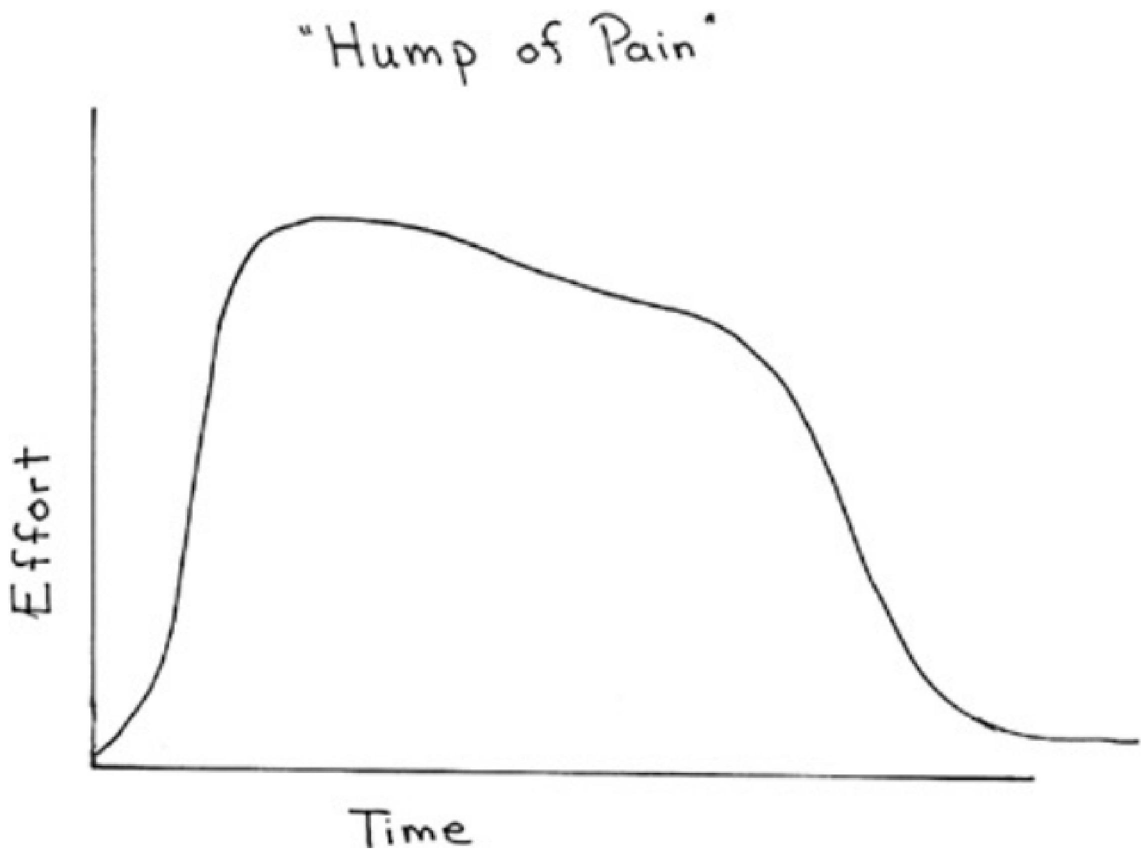
- Part IV. Automation
  - Chapter 13. Why we want to automate tests and what holds us back

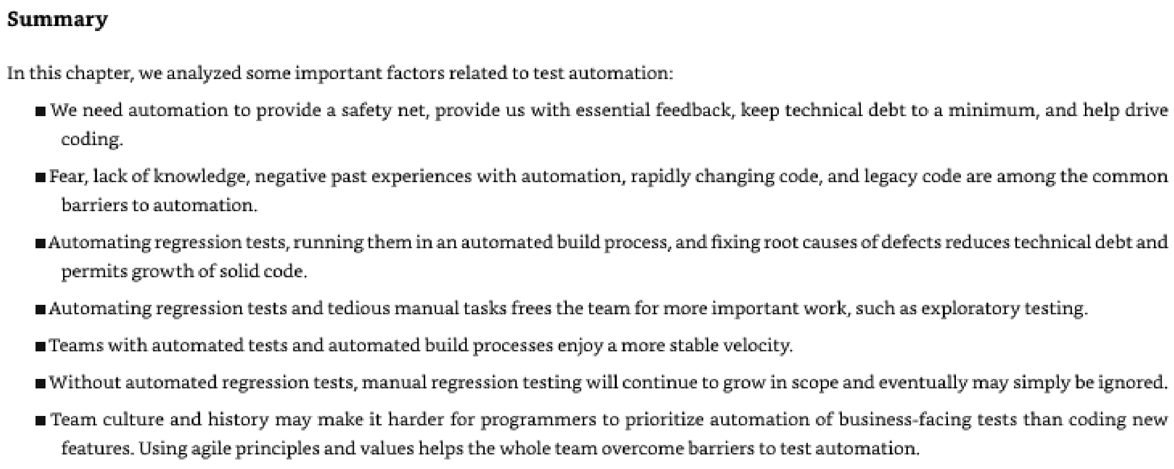

Why Automate?
Reasons to Automate
Manual Testing Takes Too Long
Reduce Error-Prone Testing Tasks
Free Up Time to Do Best Work
Safety Net
Provide Feedback Early and Often
Tests & Examples that Drive Coding Can Do More
Tests Provide Documentation
ROI/Investment
Obstacles to Watch For
Bret's List
Attitude—Why Should We Automate?
Hump of Pain
Initial Investment
Code in Flux
Legacy Code
Fear
Old Habits
Can We Overcome Barriers?
Culture
Whole-Team Approach



The "Hump of Pain" (The Learning Curve)

It's hard to learn test automation, especially to learn how to do it in a way that produces a good return on the resources invested in it. A term we've heard Brian Marick use to describe the initial phase of automation that developers (including testers) have to overcome is the "hump of pain" (see Figure 13-1). This phrase refers to the struggle that most teams go through when adopting automation.



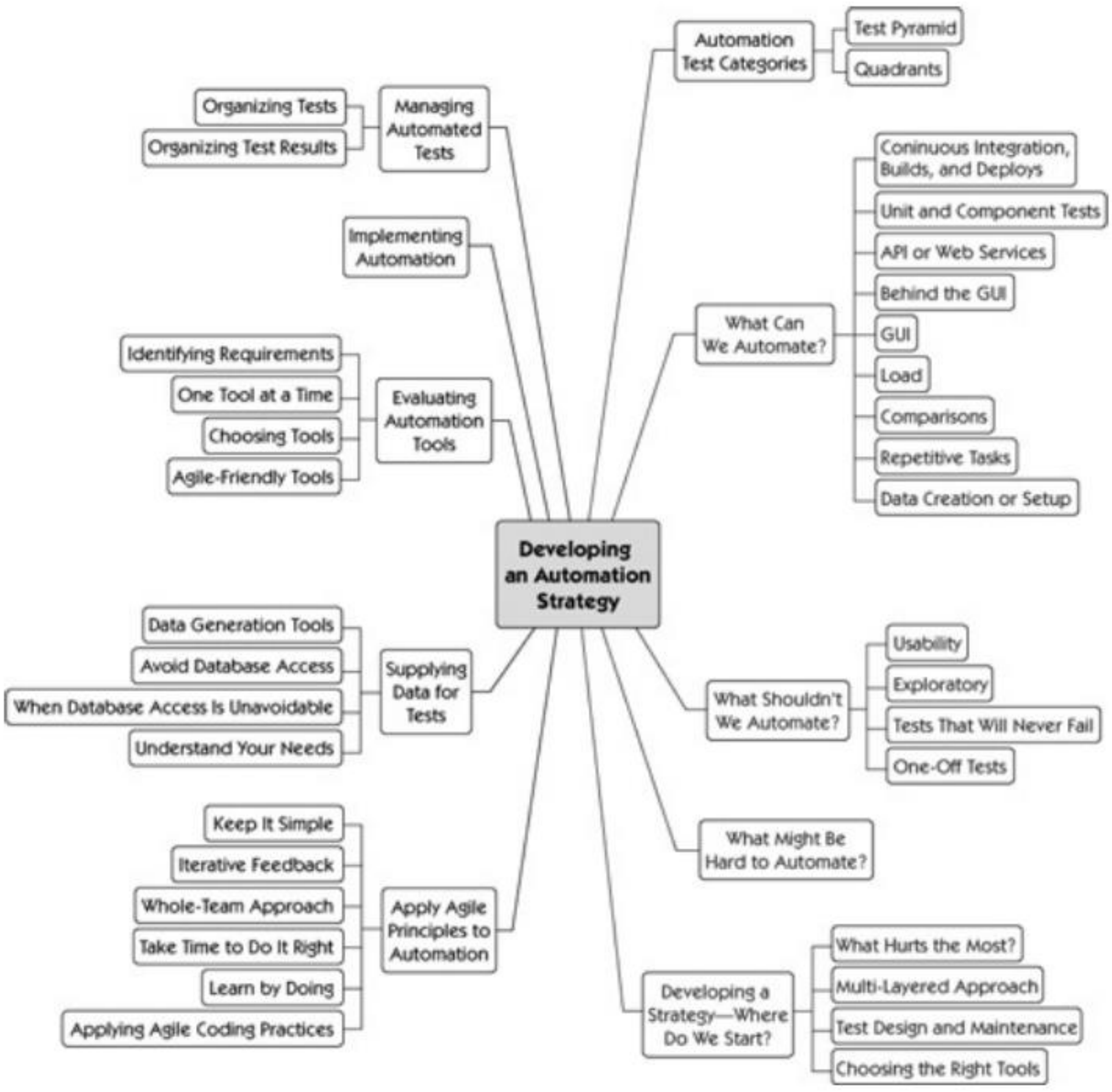
- 

Summary

In this chapter, we analyzed some important factors related to test automation:

- We need automation to provide a safety net, provide us with essential feedback, keep technical debt to a minimum, and help drive coding.
- Fear, lack of knowledge, negative past experiences with automation, rapidly changing code, and legacy code are among the common barriers to automation.
- Automating regression tests, running them in an automated build process, and fixing root causes of defects reduces technical debt and permits growth of solid code.
- Automating regression tests and tedious manual tasks frees the team for more important work, such as exploratory testing.
- Teams with automated tests and automated build processes enjoy a more stable velocity.
- Without automated regression tests, manual regression testing will continue to grow in scope and eventually may simply be ignored.
- Team culture and history may make it harder for programmers to prioritize automation of business-facing tests than coding new features. Using agile principles and values helps the whole team overcome barriers to test automation.

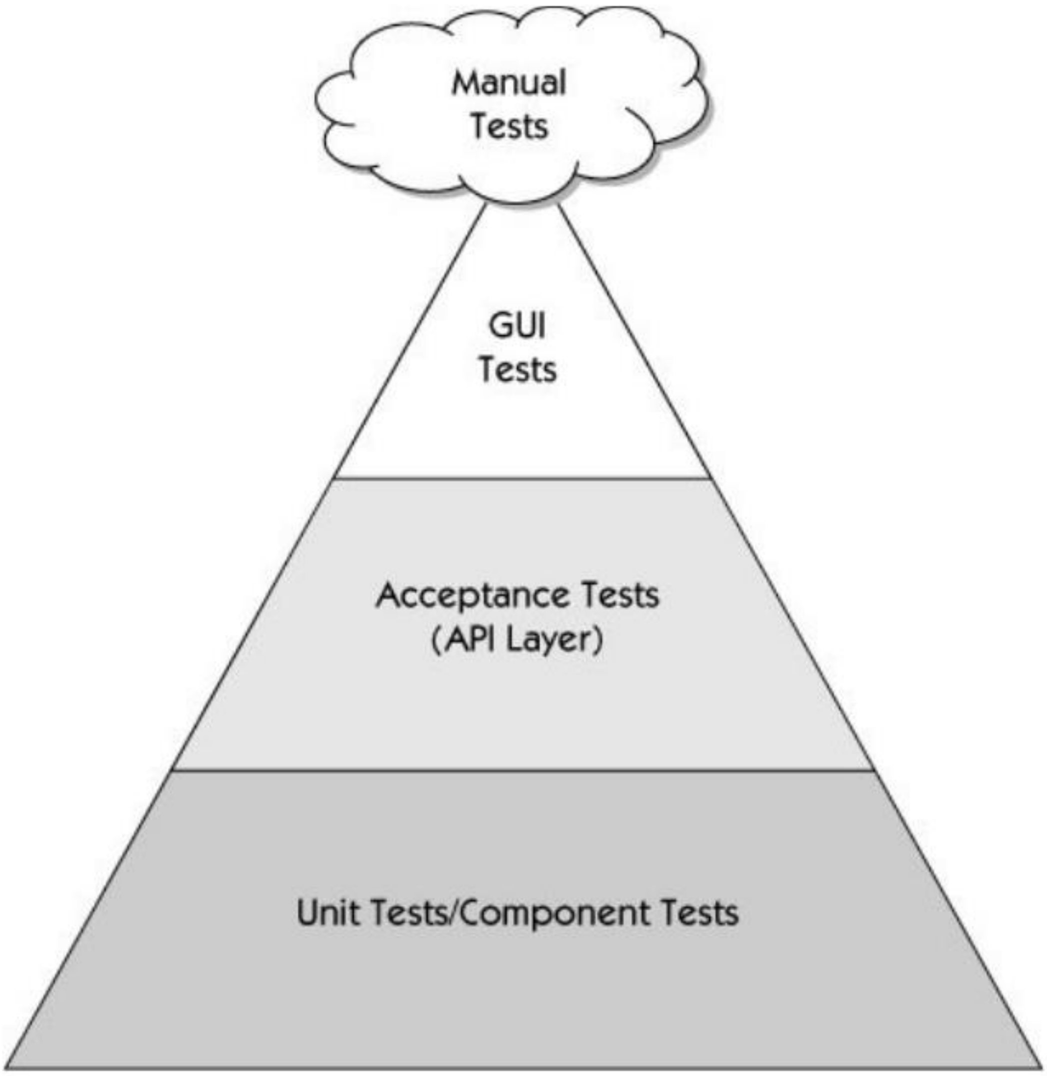
- Chapter 14. An agile test automation strategy



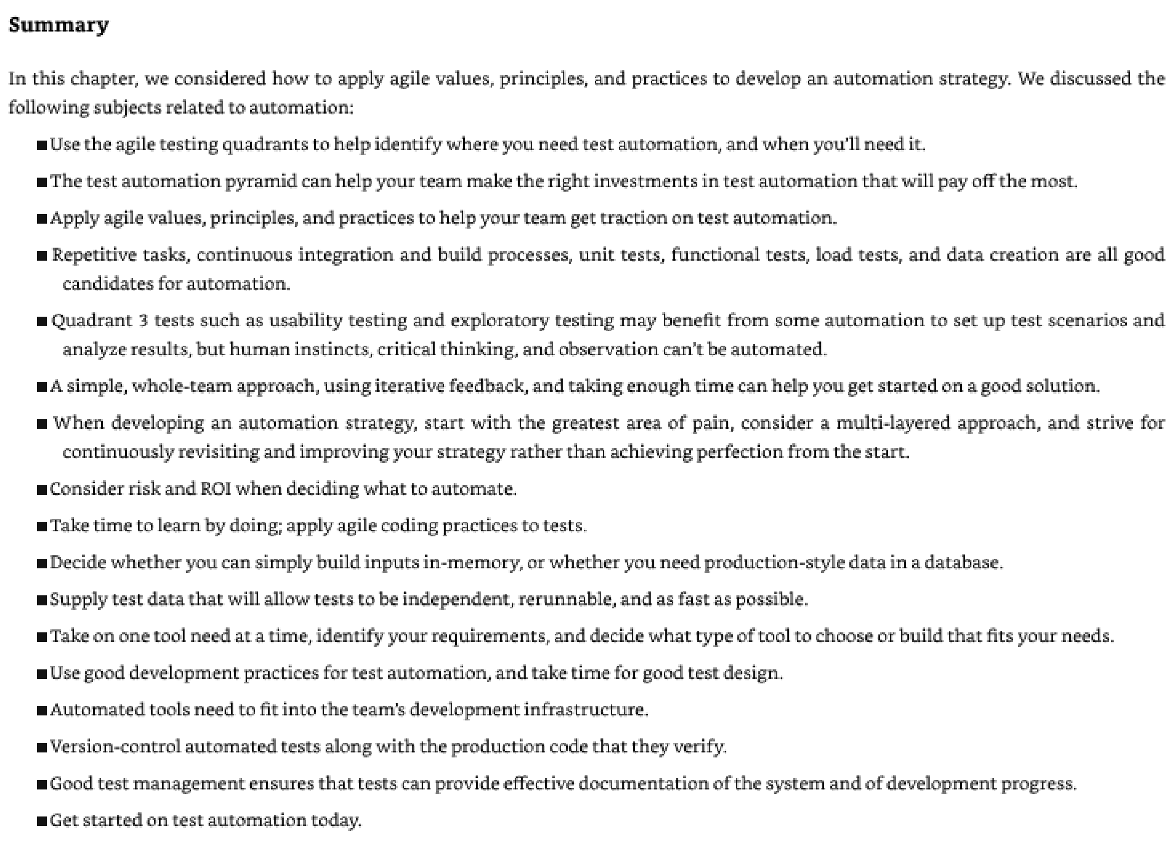
Developing an Automation Strategy
Automation Test Categories
Test Pyramid
Quadrants
What Can We Automate?
Coninuous Integration, Builds, and Deploys
Unit and Component Tests
API or Web Services
Behind the GUI
GUI
Load
Comparisons
Repetitive Tasks
Data Creation or Setup
What Shouldn't We Automate?
Usability
Exploratory
Tests That Will Never Fail
One-Off Tests
What Might Be Hard to Automate?
Developing a Strategy—Where Do We Start?
What Hurts the Most?
Multi-Layered Approach
Test Design and Maintenance
Choosing the Right Tools
Managing Automated Tests
Organizing Tests
Organizing Test Results
Implementing Automation
Evaluating Automation Tools
Identifying Requirements
One Tool at a Time
Choosing Tools
Agile-Friendly Tools
Supplying Data for Tests
Data Generation Tools
Avoid Database Access
When Database Access Is Unavoidable
Understand Your Needs
Apply Agile Principles to Automation
Keep It Simple
Iterative Feedback
Whole-Team Approach
Take Time to Do It Right
Learn by Doing
Applying Agile Coding Practices

▪

Figure 14-2 Test automation pyramid



■

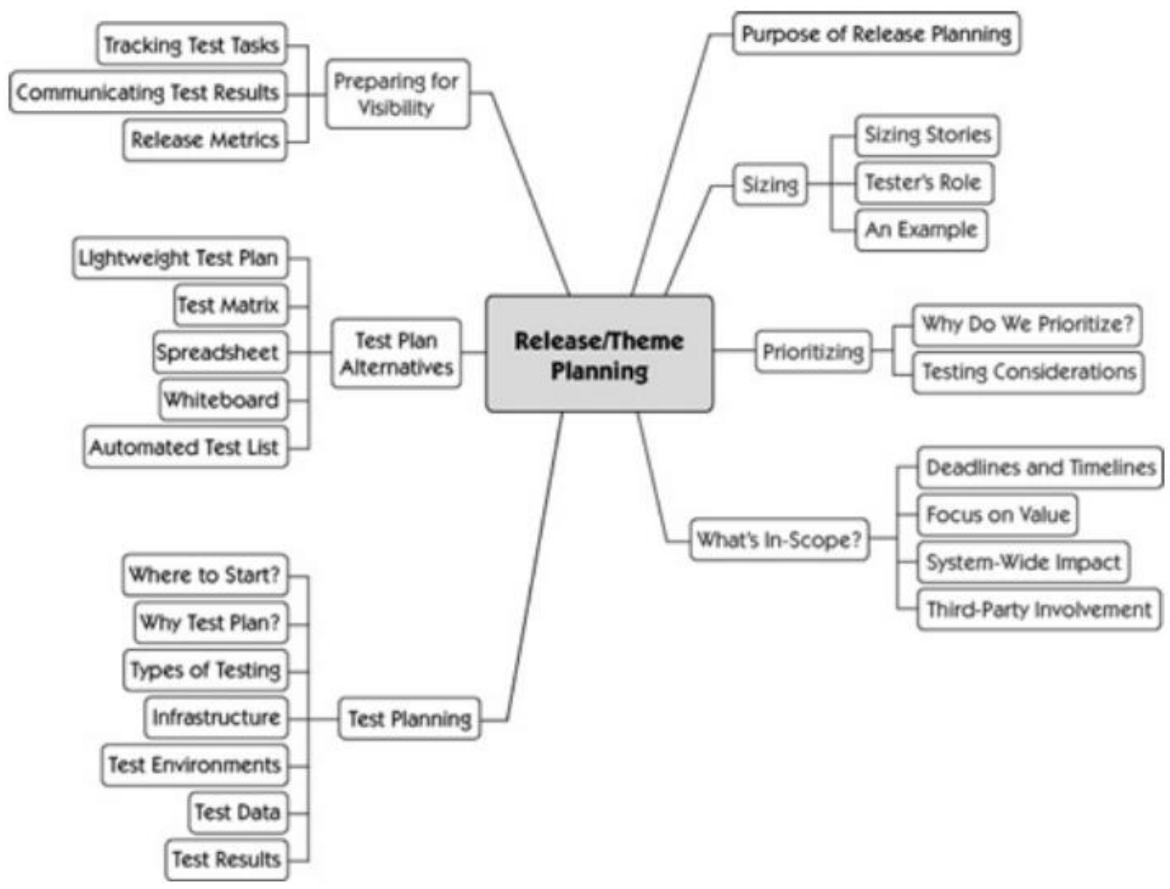


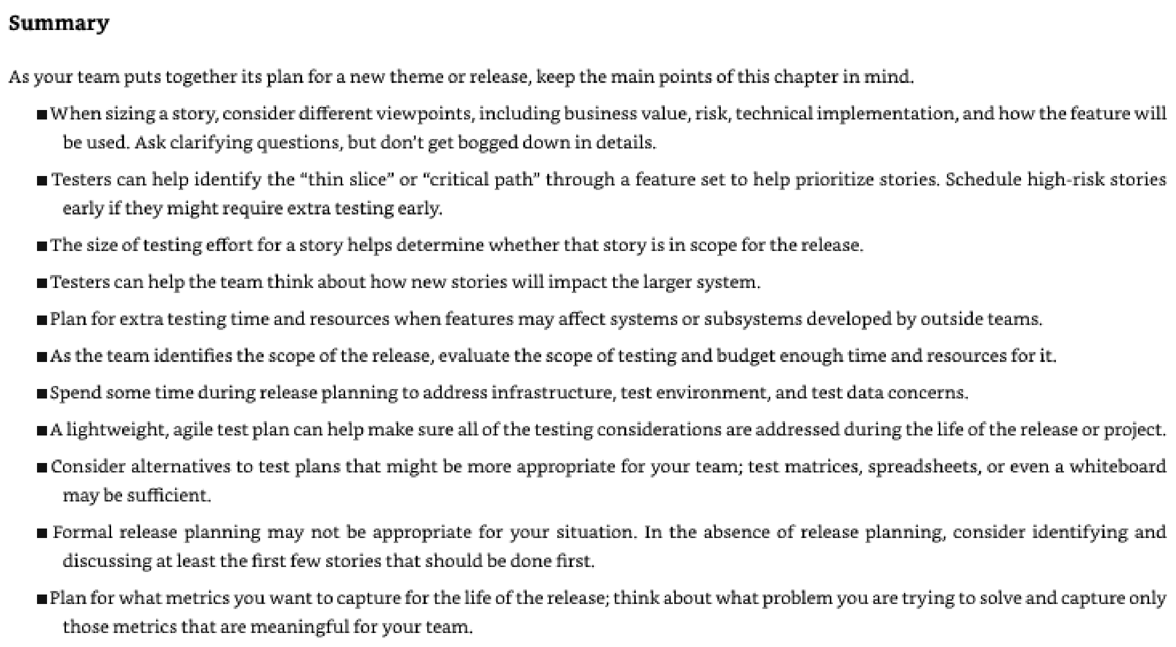
Summary

In this chapter, we considered how to apply agile values, principles, and practices to develop an automation strategy. We discussed the following subjects related to automation:

- Use the agile testing quadrants to help identify where you need test automation, and when you'll need it.
- The test automation pyramid can help your team make the right investments in test automation that will pay off the most.
- Apply agile values, principles, and practices to help your team get traction on test automation.
- Repetitive tasks, continuous integration and build processes, unit tests, functional tests, load tests, and data creation are all good candidates for automation.
- Quadrant 3 tests such as usability testing and exploratory testing may benefit from some automation to set up test scenarios and analyze results, but human instincts, critical thinking, and observation can't be automated.
- A simple, whole-team approach, using iterative feedback, and taking enough time can help you get started on a good solution.
- When developing an automation strategy, start with the greatest area of pain, consider a multi-layered approach, and strive for continuously revisiting and improving your strategy rather than achieving perfection from the start.
- Consider risk and ROI when deciding what to automate.
- Take time to learn by doing; apply agile coding practices to tests.
- Decide whether you can simply build inputs in-memory, or whether you need production-style data in a database.
- Supply test data that will allow tests to be independent, rerunnable, and as fast as possible.
- Take on one tool need at a time, identify your requirements, and decide what type of tool to choose or build that fits your needs.
- Use good development practices for test automation, and take time for good test design.
- Automated tools need to fit into the team's development infrastructure.
- Version-control automated tests along with the production code that they verify.
- Good test management ensures that tests can provide effective documentation of the system and of development progress.
- Get started on test automation today.

- Part V. An iteration in the life of a tester
  - Chapter 15. Tester activities in release or theme planning



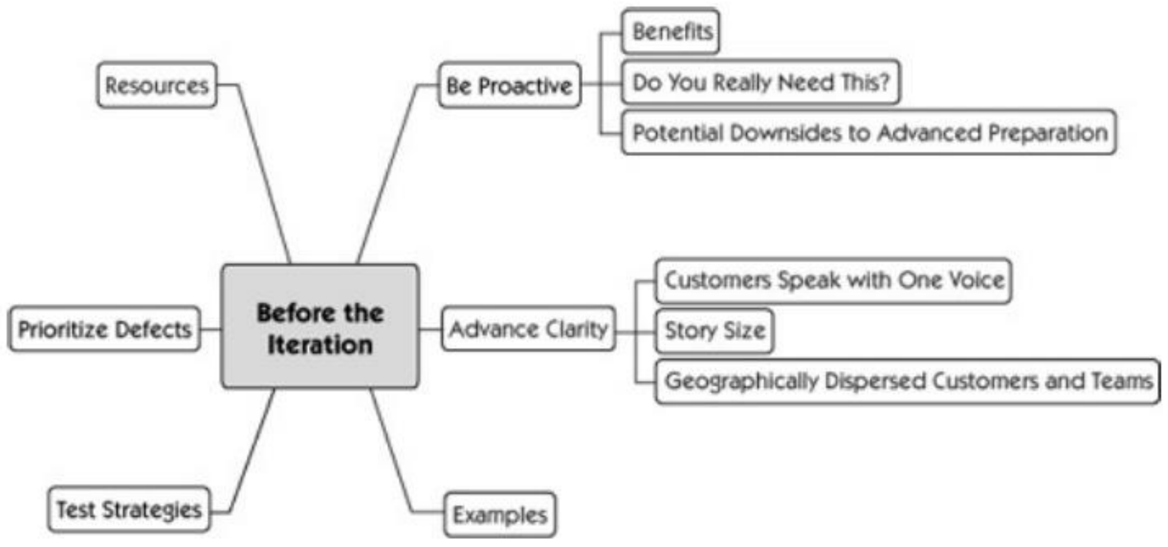
- 

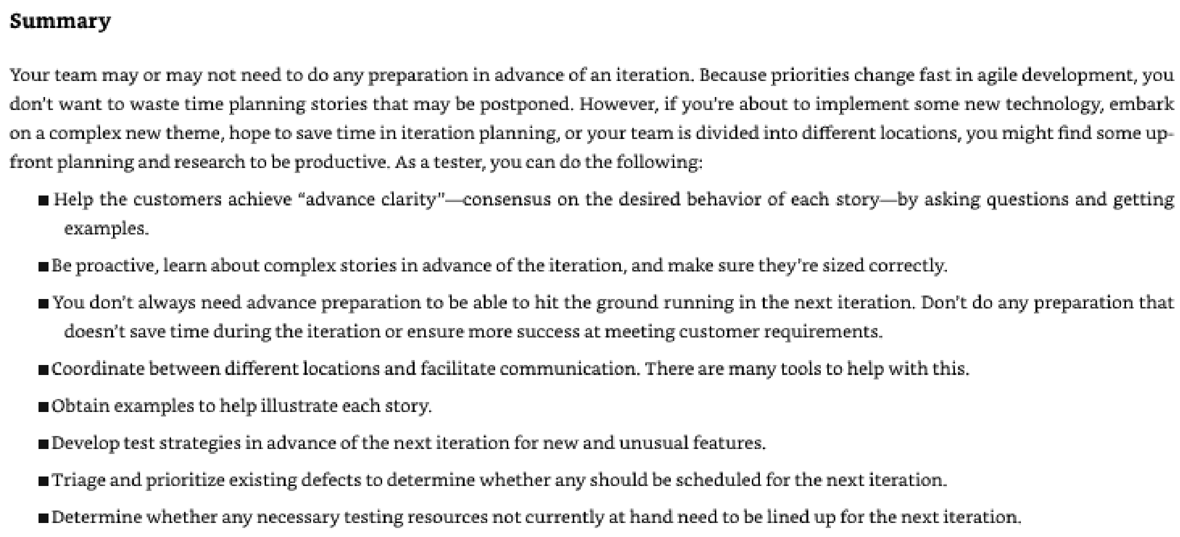
  Summary

  As your team puts together its plan for a new theme or release, keep the main points of this chapter in mind.

  - When sizing a story, consider different viewpoints, including business value, risk, technical implementation, and how the feature will be used. Ask clarifying questions, but don't get bogged down in details.
  - Testers can help identify the "thin slice" or "critical path" through a feature set to help prioritize stories. Schedule high-risk stories early if they might require extra testing early.
  - The size of testing effort for a story helps determine whether that story is in scope for the release.
  - Testers can help the team think about how new stories will impact the larger system.
  - Plan for extra testing time and resources when features may affect systems or subsystems developed by outside teams.
  - As the team identifies the scope of the release, evaluate the scope of testing and budget enough time and resources for it.
  - Spend some time during release planning to address infrastructure, test environment, and test data concerns.
  - A lightweight, agile test plan can help make sure all of the testing considerations are addressed during the life of the release or project.
  - Consider alternatives to test plans that might be more appropriate for your team; test matrices, spreadsheets, or even a whiteboard may be sufficient.
  - Formal release planning may not be appropriate for your situation. In the absence of release planning, consider identifying and discussing at least the first few stories that should be done first.
  - Plan for what metrics you want to capture for the life of the release; think about what problem you are trying to solve and capture only those metrics that are meaningful for your team.

○ Chapter 16. Hit the ground running

- 

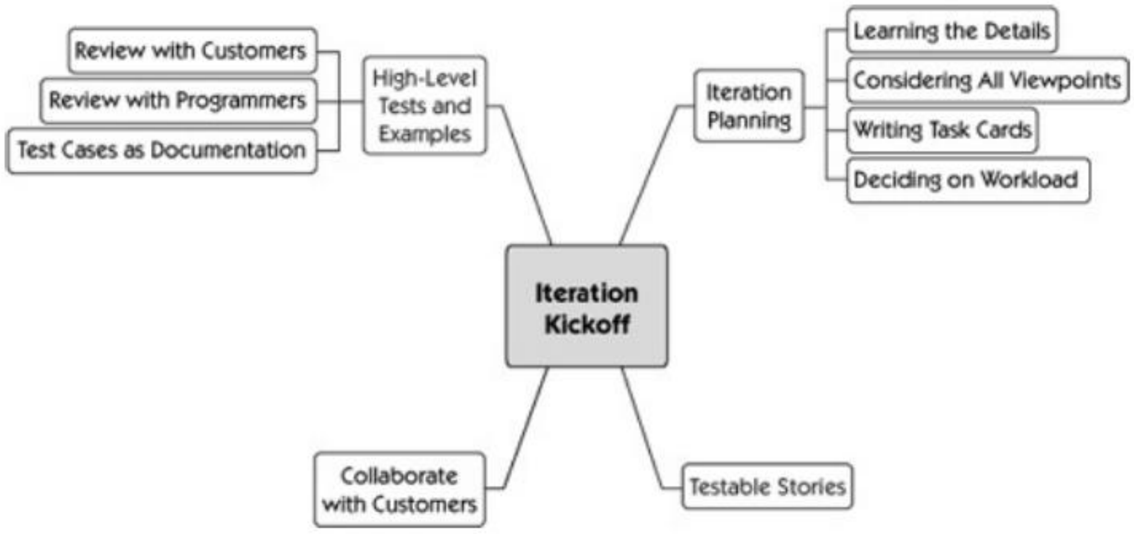
- 

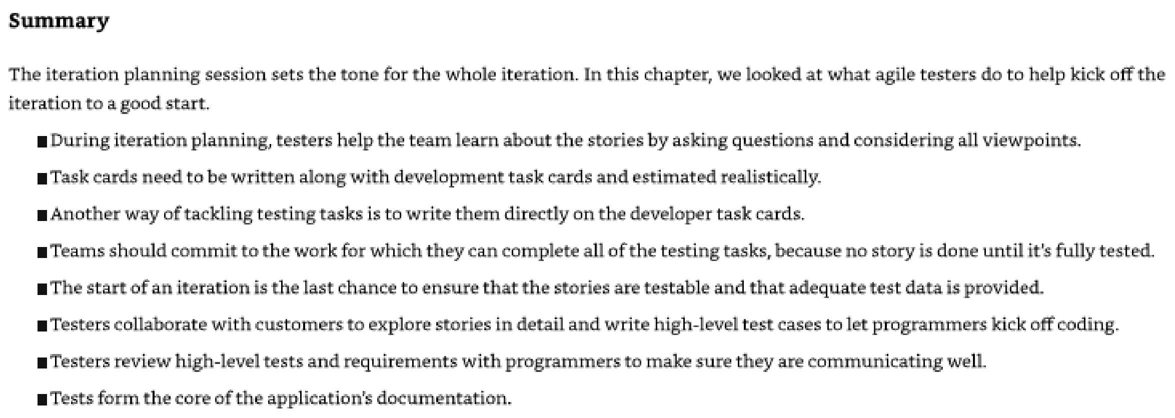
  Summary

  Your team may or may not need to do any preparation in advance of an iteration. Because priorities change fast in agile development, you don't want to waste time planning stories that may be postponed. However, if you're about to implement some new technology, embark on a complex new theme, hope to save time in iteration planning, or your team is divided into different locations, you might find some up-front planning and research to be productive. As a tester, you can do the following:

  - Help the customers achieve "advance clarity"—consensus on the desired behavior of each story—by asking questions and getting examples.
  - Be proactive, learn about complex stories in advance of the iteration, and make sure they're sized correctly.
  - You don't always need advance preparation to be able to hit the ground running in the next iteration. Don't do any preparation that doesn't save time during the iteration or ensure more success at meeting customer requirements.
  - Coordinate between different locations and facilitate communication. There are many tools to help with this.
  - Obtain examples to help illustrate each story.
  - Develop test strategies in advance of the next iteration for new and unusual features.
  - Triage and prioritize existing defects to determine whether any should be scheduled for the next iteration.
  - Determine whether any necessary testing resources not currently at hand need to be lined up for the next iteration.

- Chapter 17. Iteration kick-off

  - 

  - 

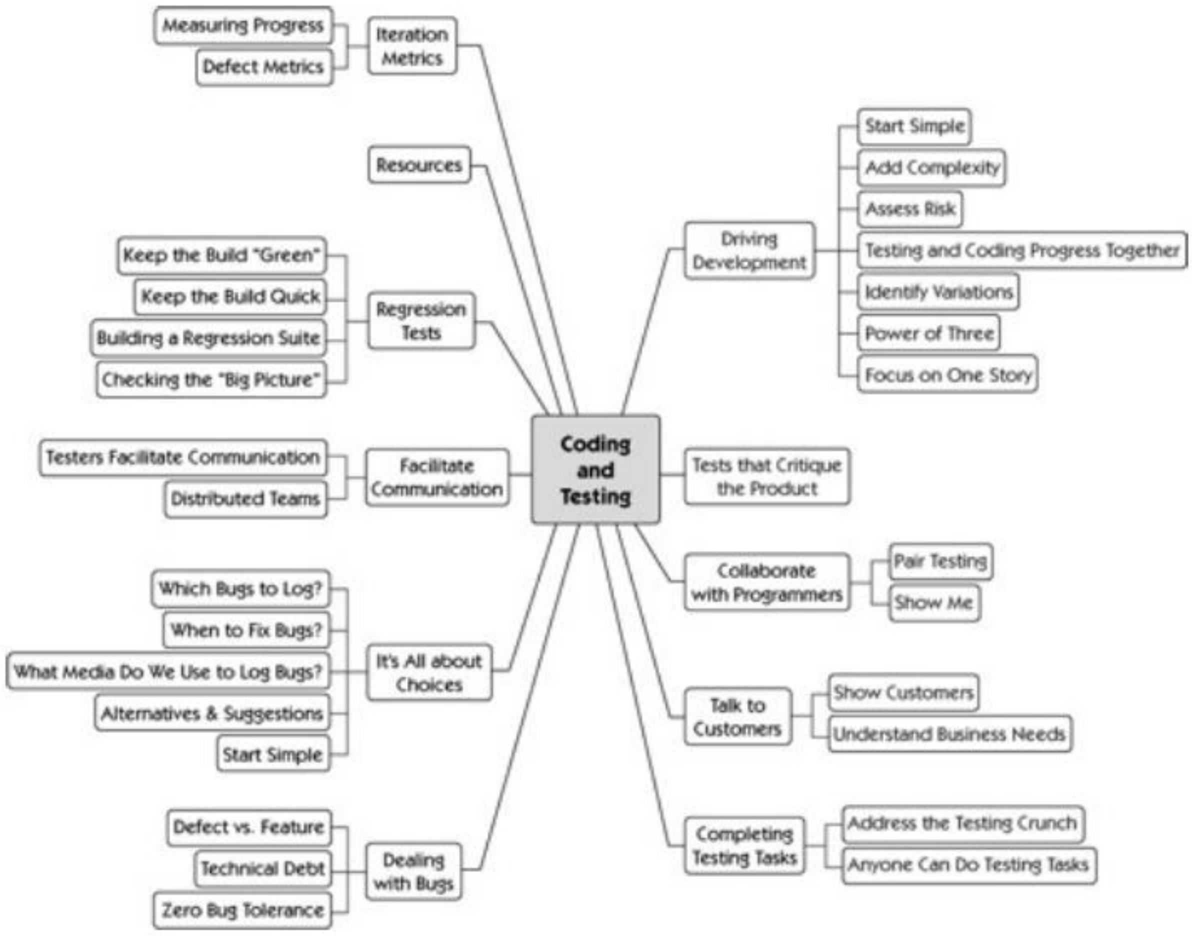
    Summary

    The iteration planning session sets the tone for the whole iteration. In this chapter, we looked at what agile testers do to help kick off the iteration to a good start.

    - During iteration planning, testers help the team learn about the stories by asking questions and considering all viewpoints.
    - Task cards need to be written along with development task cards and estimated realistically.
    - Another way of tackling testing tasks is to write them directly on the developer task cards.
    - Teams should commit to the work for which they can complete all of the testing tasks, because no story is done until it's fully tested.
    - The start of an iteration is the last chance to ensure that the stories are testable and that adequate test data is provided.
    - Testers collaborate with customers to explore stories in detail and write high-level test cases to let programmers kick off coding.
    - Testers review high-level tests and requirements with programmers to make sure they are communicating well.
    - Tests form the core of the application's documentation.

  -

- Chapter 18. Coding and testing

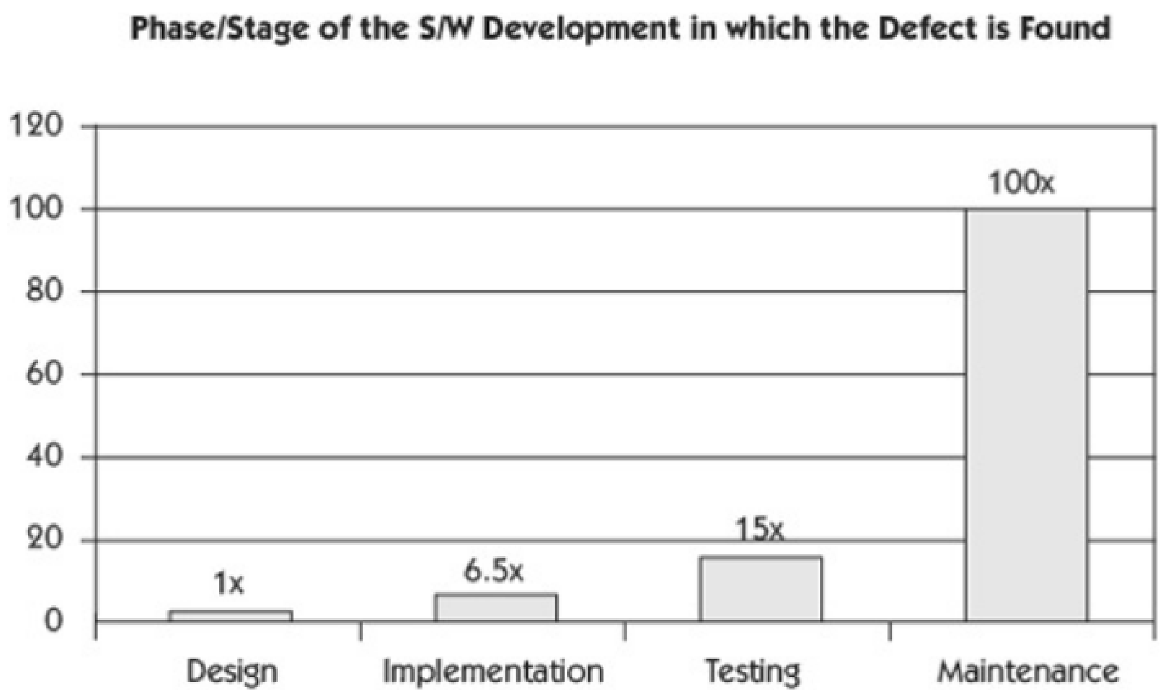

Coding and Testing
Iteration Metrics
Measuring Progress
Defect Metrics
Resources
Regression Tests
Keep the Build "Green"
Keep the Build Quick
Building a Regression Suite
Checking the "Big Picture"
Facilitate Communication
Testers Facilitate Communication
Distributed Teams
It's All about Choices
Which Bugs to Log?
When to Fix Bugs?
What Media Do We Use to Log Bugs?
Alternatives & Suggestions
Start Simple
Dealing with Bugs
Defect vs. Feature
Technical Debt
Zero Bug Tolerance
Driving Development
Start Simple
Add Complexity
Assess Risk
Testing and Coding Progress Together
Identify Variations
Power of Three
Focus on One Story
Tests that Critique the Product
Collaborate with Programmers
Pair Testing
Show Me
Talk to Customers
Show Customers
Understand Business Needs
Completing Testing Tasks
Address the Testing Crunch
Anyone Can Do Testing Tasks

▪

Fix Now

The more bugs you can fix immediately, the less technical debt your application generates and the less "defect" inventory you have. Defects are also cheaper to fix the sooner they are discovered. In an article in *iSixSigma Magazine*, Mukesh Soni [2008] quotes a report from IBM that the cost to fix an error found after product release was four to five times as much as one uncovered during design, and up to 100 times more than one identified in the maintenance phase (see Figure 18-5).

Figure 18-5 Relative costs to fix software defects



-

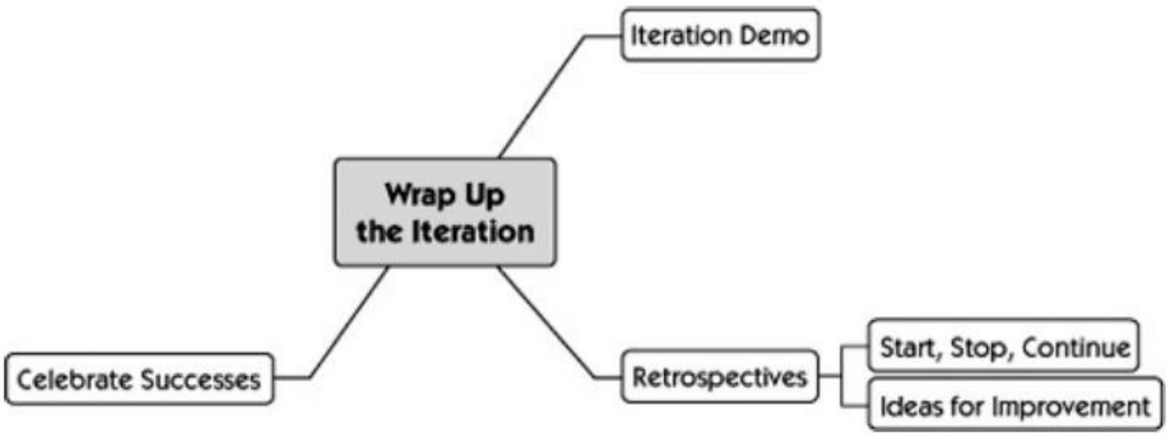
Summary

At this point in our example iteration, our agile tester works closely with programmers, customers, and other team members to produce stories in small testing-coding-reviewing-testing increments. Some points to keep in mind are:

- Coding and testing are part of one process during the iteration.
- Write detailed tests for a story as soon as coding begins.
- Drive development by starting with a simple test; when the simple tests pass, write more complex test cases to further guide coding.
- Use simple risk assessment techniques to help focus testing efforts.
- Use the "Power of Three" when requirements aren't clear or opinions vary.
- Focus on completing one story at a time.
- Collaborate closely with programmers so that testing and coding are integrated.
- Tests that critique the product are part of development.
- Keep customers in the loop throughout the iteration; let them review early and often.
- Everyone on the team can work on testing tasks.
- Testers can facilitate communication between the customer team and development team.
- Determine what the best "bug fixing" choice for your team is, but a good goal is to aim to have no bugs by release time.
- Add new automated tests to the regression suite and schedule it to run often enough to provide adequate feedback.
- Manual exploratory testing helps find missing requirements after all the application has been coded.
- Collaborate with other experts to get the resources and infrastructure needed to complete testing.
- Consider what metrics you need during the iteration; progress and defect metrics are two examples.

-

- Chapter 19. Wrap up the iteration



▪

Summary

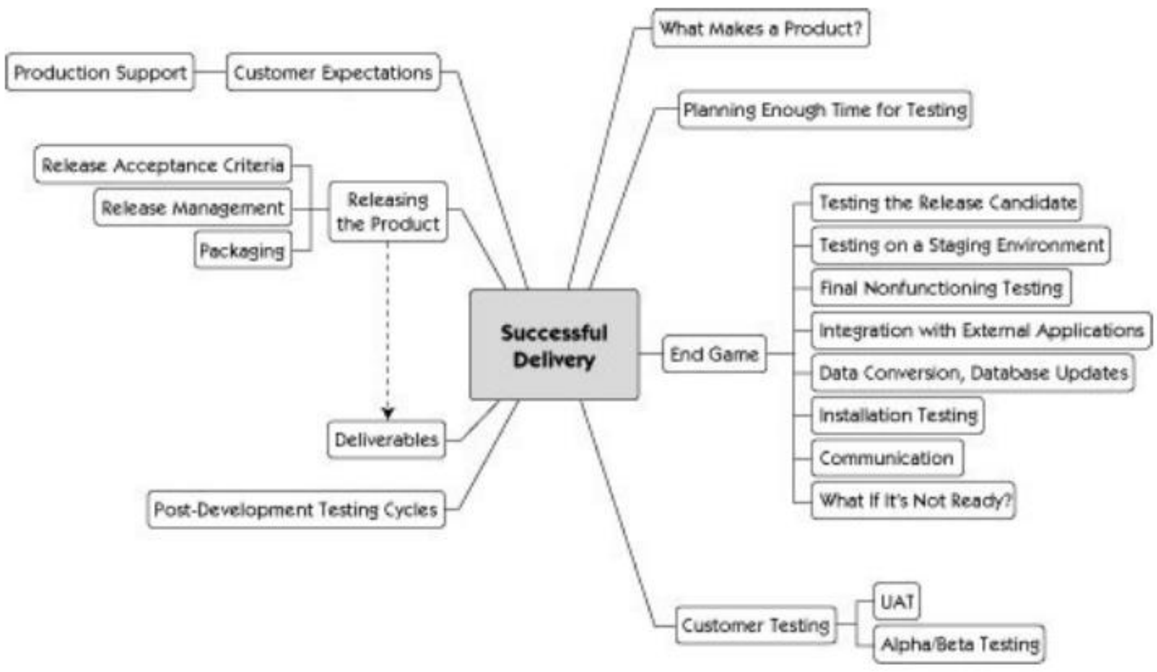
- ■ In this chapter, we looked at some activities to wrap up the iteration or release.
- ■ The iteration review is an excellent opportunity to get feedback and input from the customer team.
- ■ Retrospectives are a critical practice to help your team improve.

▪

- ■ Look at all areas where the team can improve, but focus on one or two at a time.
- ■ Find a way to keep improvement items in mind during the iteration.
- ■ Celebrate both big and small successes, and recognize the contributions from different roles and activities.
- ■ Take advantage of the opportunity after each iteration to identify testing-related obstacles, and think of ways to overcome them.

▪

○ Chapter 20. Successful delivery


What Makes a Product?
Production Support
Customer Expectations
Planning Enough Time for Testing
Release Acceptance Criteria
Release Management
Packaging
Releasing the Product
Testing the Release Candidate
Testing on a Staging Environment
Final Nonfunctioning Testing
Integration with External Applications
Successful Delivery
End Game
Data Conversion, Database Updates
Installation Testing
Deliverables
Communication
Post-Development Testing Cycles
What If It's Not Ready?
UAT
Customer Testing
Alpha/Beta Testing

▪

Figure 20-1 Release timeline with UAT

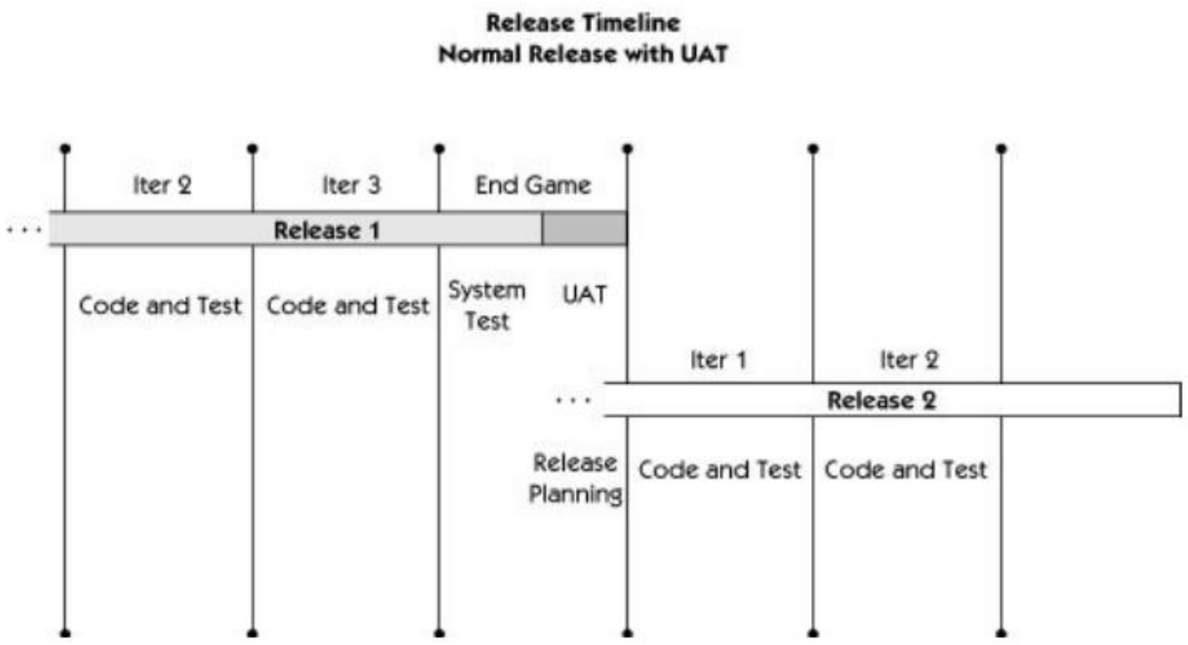
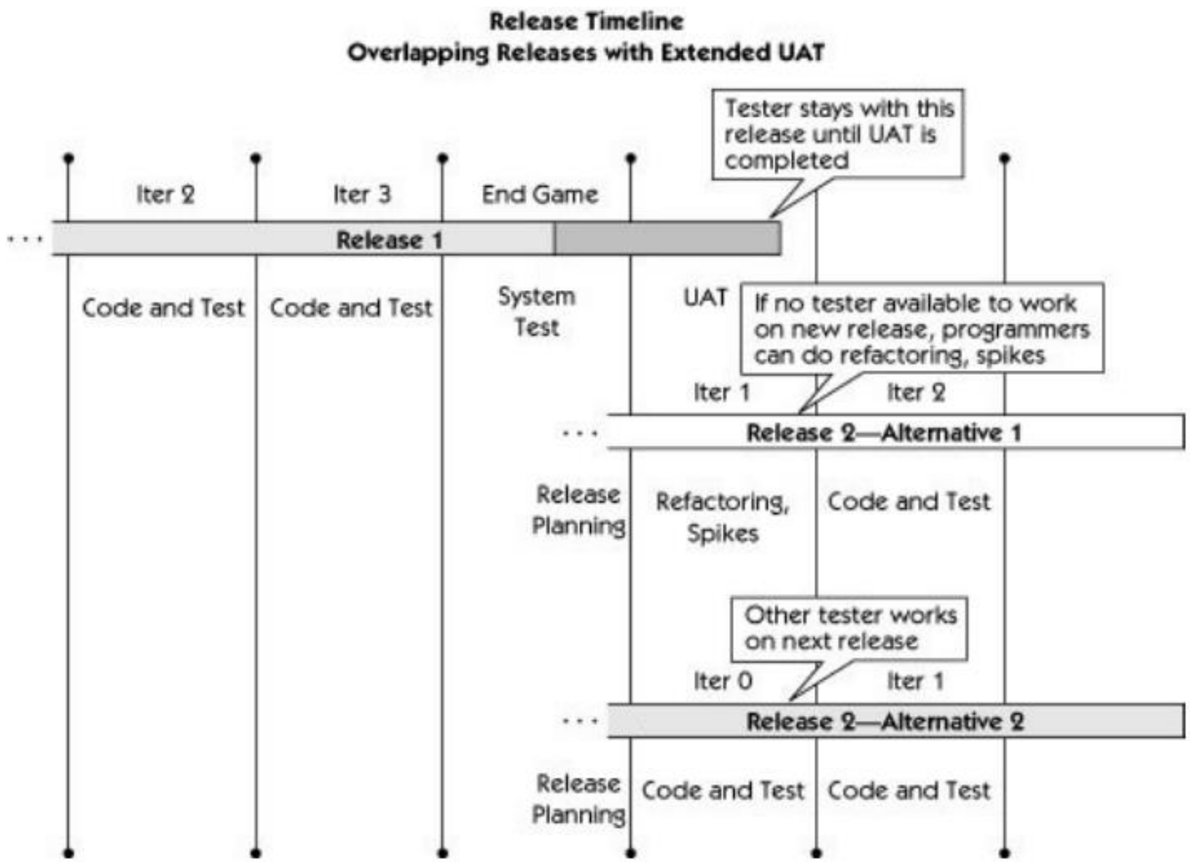


Figure 20-2 Release timeline—alternative approach with extended UAT

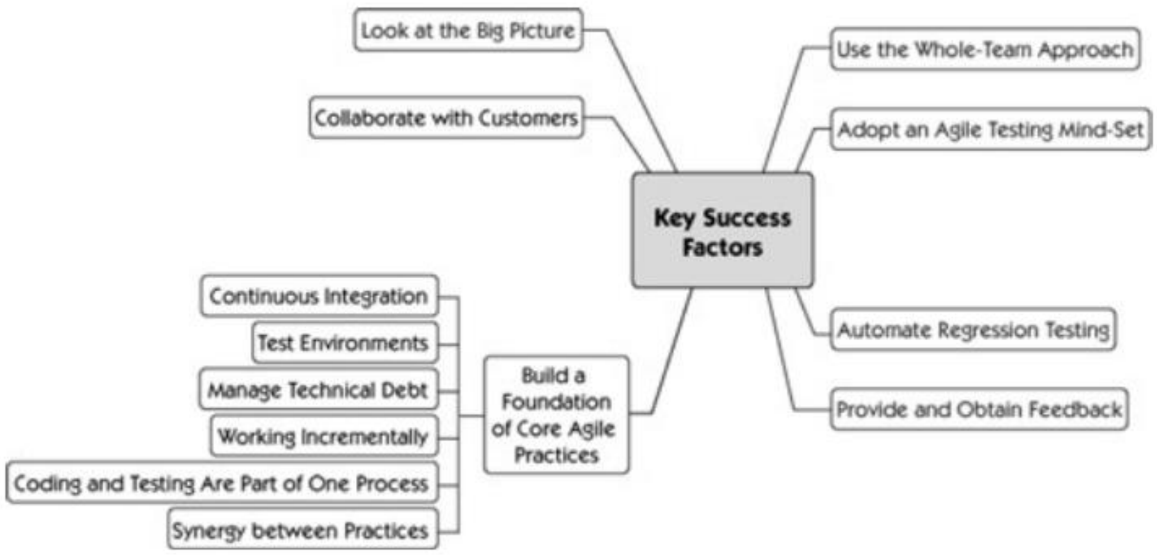


Summary

This chapter covered the following points:

- Successful delivery of a product includes more than just the application you are building. Plan the non-software deliverables such as documentation, legal notices, and training.
- The end game is an opportunity to put the spit and polish, the final finishing touches, on your product.
- Other groups may be responsible for environments, tools, and other components of the end game and release. Coordinate with them ahead of time.
- Be sure to test database update scripts, data conversions, and other parts of the installation.
- UAT is an opportunity for customers to test against their data and to build their confidence in the product.
- Budget time for extra cycles as needed, such as post-development cycles to coordinate testing with outside parties.
- Establish release acceptance criteria during release planning so that you can know when you're ready to release.
- Testers often are involved in managing releases and testing the packaging.
- When releasing the product, consider the whole package—what the customer needs and expects.
- Learn from each release, and adapt to improve your processes.

- Part VI Summary
  - Chapter 21 Key success factors


Look at the Big Picture
Collaborate with Customers
Use the Whole-Team Approach
Adopt an Agile Testing Mind-Set
Key Success Factors
Continuous Integration
Test Environments
Manage Technical Debt
Working Incrementally
Coding and Testing Are Part of One Process
Synergy between Practices
Build a Foundation of Core Agile Practices
Automate Regression Testing
Provide and Obtain Feedback

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.