TestRail

The Fundamentals of Test Management

Test Strategy, Planning, and Execution

TestRail

Every software development project includes testing activities. These activities may be managed formally or informally, by a designated QA manager, or by various individuals on a development team. Managing these activities properly is critical if you want to have the best chance of verifying that your application works as intended.

Keep reading for essential test management topics and tactical ideas to help you apply the concepts presented.

Contents

Introduction

It is helpful to think of testing as less of a role and more of an activity that people do. Everyone tests, but some people specialize and make a career of it. In the same way, test management is an activity associated with testing. Whether you are the sole tester on a team or the QA lead manager overseeing all of the testing for a massive project, you have test management activities such as keeping good records of tests planned and/or executed as well as more strategic activities such as developing a strategic plan for test coverage.

Whether your development team follows an agile approach, waterfall approach, tests entirely manually, or is assisted by automation, this ebook describes the fundamentals of test management activities and provides insights into how you can apply them to your unique situation. This ebook will also help you to identify thinking processes that will help adapt these secrets for your own context.

The Fundamentals of Test Management

What is Test Management?

If you are the QA manager, it is very likely that people will assume you are the expert on all things testing-related. Other team members will have their own views on testing. Some may even have more experience than you. Expectations of testing are often unrealistic. There may be people who doubt your competence, value to the team, or even your motivations. It can be tough.

In your career, you will have to adapt to new and changing circumstances. You will encounter business and senior project stakeholders. You will be joining teams of various sizes with different people in them who have widely varying backgrounds. In your role, you may be running a large team, providing oversight to the testing in a project, advising an Agile team, or figuring out how testing fits into a continuous delivery process.
Before we start using terms like **testing**, **stakeholder**, and **stakeholder goals**, let's be clear about what we're talking about.

Definition of "Test"

The word test is used as a noun and as a verb. It's about testing as an activity and the outcome of that activity. It's about the people or organizations who commission that activity and those who use the results. It's very much about the people who call themselves testers and the complex systems on which we work.

To talk about context-neutral testing we'll need a definition of the word "test" that is context-neutral. The definition from the American Heritage Dictionary conveys the multiple ways that we can talk about testing well:

Test: (noun) a procedure for critical evaluation; a means of determining the presence, quality, or truth of something; a trial.

There is not just one definition; rather, there are three variations. All three taken together give us the foundations we need. Let's take a closer look at each one.

A procedure for critical evaluation

Critical evaluation involves a skillful judgment as to the truth or merit of something. A test is a procedure, usually with a series of steps that include some form of preparation, execution, results gathering, analysis, and interpretation. This isn't a definitive description of a test procedure. There could be more steps and one could break these main steps down further.

The procedure doesn't necessarily require prepared documentation, but some tests are documented. Automated tests are always scripted in some way. The important thing is that there is a clear thought process at the heart of a test. This thought process is driven by the need to evaluate the system under test with respect to its adequacy, consistency, behavior, accuracy, reliability, or any other significant aspect or property. Although documentation is not always necessary, it's critical to have access to information about the testing progress and test results to make important project decisions. Questions such as "What percent of a test run have we successfully completed?" or "Which tests have been executed in the past 24 hours?" can only be answered efficiently by using comprehensive test case management software that makes these details readily available.

A means of determining the presence, quality, or truth of something

The term quality is loaded with emotional connotations, but we are rescued by the same dictionary. Quality can be, "an essential or distinctive characteristic, property, or attribute". Now we can see that a test can reveal these properties.

Can a test determine the truth of something? Well, this makes good sense too. Typically, we need to test an assertion such as, "this system meets some requirement" or "this system behaves in such a way" or "this system is acceptable" and so on. There's a certain amount of subjective judgment involved but we can see that a test or tests could provide evidence for someone to exercise that judgment and make a decision.

A Trial

The notion of a trial implies that the process of testing a system will help us to evaluate that system with respect to its qualities. The purpose of such an evaluation is normally to make a decision.

The decision might be to accept or reject the system, but it might also be to expose its shortcomings so they can be remedied in some way. A test might also influence an individual or organization to change direction – to rethink a design; to relax or change a requirement; to scrap a component and start again; to buy rather than build or build rather than buy.

A natural way of looking at a system under test is that it is on trial, and will be judged in some way.

Definition of "Testing"

From our definition of the noun test, we can derive a verb easily enough.

Test: (verb) to critically evaluate; to determine the presence, quality, or truth of something; to conduct a trial.

So far, so good. But not that good. Unfortunately, the testing profession is dogged with terminological problems. There's no way we can present a definite glossary here. To avoid misunderstandings in your projects, I suggest you ask what the goal of any defined test type is rather than rely on an assumed definition to tell you.

The Fundamentals of Test Management

Defining Your Test Strategy

1. Identify Stakeholders

What is a stakeholder? In our case, stakeholders are those people who have an interest in the outcome of tests, and the information that testers provide.

If we are responsible for defining a test strategy, we need to identify and engage the people or organizations that will use and benefit from the evidence testing provides.

If we don't do this, how can we answer the following questions? What should and should not be tested? Who will use the evidence we generate from testing? What information do they need? How will we squeeze all this into the time available?

If there is no stakeholder, no one will take any notice of the outcome of the testing, so there is little point in doing any testing. We won't have a mandate or any authority for testing. We won't get the resources or time we need. We can't report test execution passes or failures or ask questions with any expectation of anyone answering them.

Who are the stakeholders and what do they want? There are four types of stakeholders based on their needs:

Sponsors: these stakeholders need evidence that the system can support their business goals and that the risk of failure is acceptable.

Users: these stakeholders need evidence that the system will 'work' for them.

Project Management: these stakeholders need to know the status of deliverables (i.e. application features): are they available, acceptable, and reasonably free of defects? If there is a problem, the manager may need to re-plan or at least manage expectations.

Designers and developers: these stakeholders need to know where their products fail – so they can fix defects or adjust their designs.

Stakeholders must articulate what evidence they need produced. This will start with the business goals and the product risks of most concern. QA managers and testers need to know what information is required from the test process.

Sometimes, this will take the form of documentation, but in smaller projects, this information is often managed through shared wikis, Kanban, or Scrum boards or ChatOps facilities. Team communications may focus on these shared resources, visible to all. Exactly how the evidence is provided needs to be agreed on with stakeholders in your project.

Even with stakeholders and contributors aligned around the requirements of a project, limitations around visibility prevent stakeholders of every level from clearly understanding and communicating around both testing coverage and status.

One key strategy is to track all of your testing efforts across all users in a tool that updates in real-time with clear visuals.

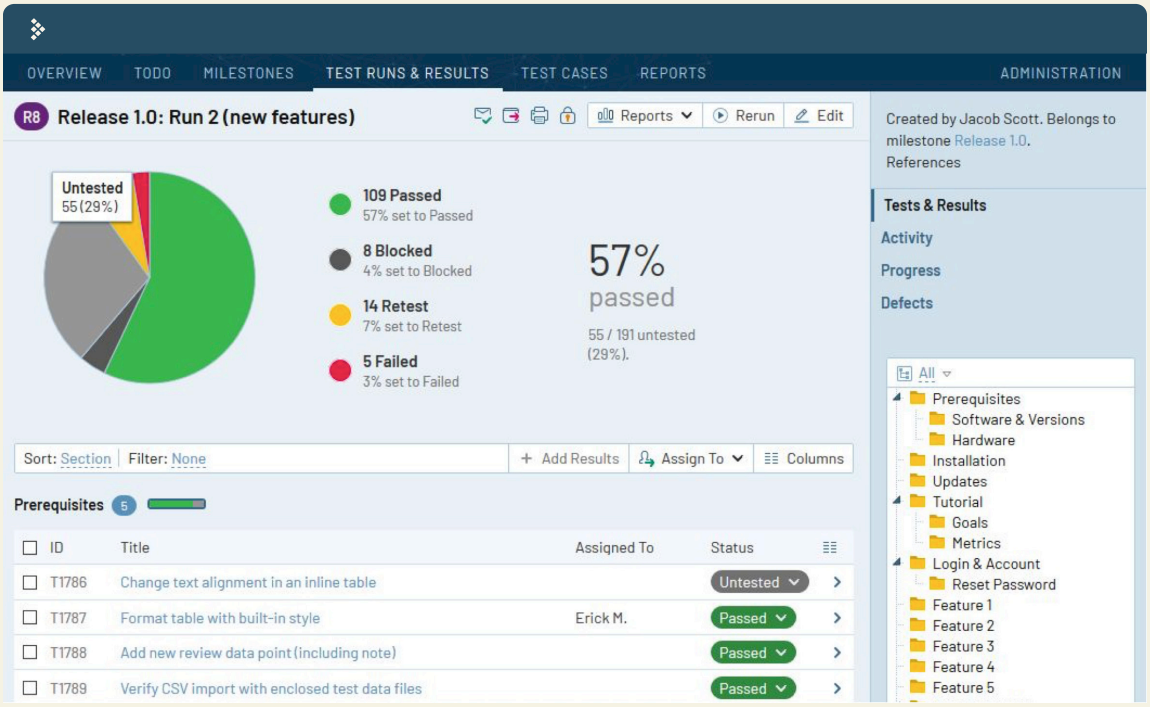


Image: TestRail test runs and results

2. Define Your Test Strategy

As you encounter new project situations, what you do to achieve your testing goal will vary. The test approach you use and almost every activity in that approach varies with the context. What does not change is the thought process you go through to define that approach.

Thinking: When you explore your situation, of course, the answers to your questions will vary. But the objective – meeting your testing goal – is generic. We'll explore the questions you should ask in order to formulate your test strategy.

Logistics: You might decide that testing will take place in stages, with each stage having different objectives, techniques, and responsibilities for test activities. You must also decide whether and how you automate some or all of the testing. Perhaps there are no stages, just a rapid series of activities (possibly automated) that must be completed within a project sprint. These are the practical, logistical choices you must make. When the dust settles and testing stages are defined, a Test Case Management solution like TestRail will streamline test execution, reporting, future strategy, management, and overall quality.

Strategic Test Planning

Your strategy is not simply a document. Your strategy is a result of exploration, thinking, and collaboration. A strategy seeks to define the process you will use to achieve your testing goals. It could be a brief set of guidelines that your team follows. It could be a document of 20 to 2,000 pages. The goal isn't a document, it's the thinking.

> **"Planning is everything. The plan is nothing."**
> – Dwight D. Eisenhower, of the D-Day preparations.

Before you can plan a test, you usually need many questions answered and decisions made. Some can be answered now, others will have to wait. A strategy, therefore:

- Presents some decisions that can be made ahead of time.
- Defines the process/method/information that will allow decisions to be made.
- Sets out the principles or process to follow for uncertain situations.

So, a strategy attempts to answer as many questions as possible ahead of time. Why bother doing this? Surely, we can address problems in testing as we encounter them? By raising these questions early, and getting people to think about the consequences, huge difficulties might be avoided, or at least mitigated, before they threaten the success of your project.

Developing A Test Strategy Framework

Below are the most important starting points for your information gathering. You could ask more questions and/or structure the questions differently. In the table below, there is no mention of the planning process. This could be defined in the strategy or not.

Stakeholder Objectives

- **Stakeholders:** See section 1 for questions regarding stakeholders
- **Goal and risk management:** How are risks identified? Who assesses/approves them?
- **Decisions:** What decisions must stakeholders make, and how will they be made?
- **Confidence:** How will the test results give stakeholders confidence in the testing?
- **Assessment:** How will the quality/thoroughness of the testing be assessed?
- **Scope:** How will scope be defined?

Design approach

- **Sources of knowledge:** What/where/who are the knowledge sources to scope and specify tests?
- **Sources of uncertainty:** What causes uncertainty in our sources of knowledge?
- **Models:** How will test models be derived? How will they relate to stakeholders?
- **Prioritization approach:** Under time pressure, how will priorities be assigned to tests?

Delivery approach

- **Test sequencing:** How will the sequence of tests be determined?
- **Retesting:** What is the policy for retesting? For regression testing?
- **Environment requirements:** Who provides test environments? How will they be delivered, controlled, and managed?
- **Information delivery approach:** How will the team deliver results to the stakeholders?
- **Incident management approach:** How will incidents be managed?
- **End-game approach:** How will the test process end? (How) will outstanding bugs be fixed and retested?

The Role of Shift-Left Testing in Your Test Strategy

Shift-Left can mean developers take more ownership and responsibility for their testing; it can also mean testers get involved earlier, challenge requirements, and feed examples through a Behavior-Driven Development (BDD) process to developers. It can mean users and BAs together with developers take full responsibility for testing, and it can mean no test team and no testers. We have seen all configurations and there is no "one true way."

Sources of knowledge that provide direction for the design and development of software should be challenged and/or tested.

In a staged project, this might involve formal reviews. In an Agile project, the tester (or developer or BA or user) can suggest scenarios or examples that challenge the author of a requirement or story to think through concrete examples and discussion before any code is written.

Shift-Left is mostly about bringing the thinking about testing earlier in the process.

Shift-Left implies that, whenever possible, you should provide feedback that will help the team to understand, challenge, or improve goals, requirements, design, or implementation. Users, BAs, developers, and the entire team should be ready to provide and accept feedback in this way. There might be resistance, but the overall aim is to run a better, more informed project.

As a QA manager, you need to get testers involved as early as possible by engaging in the discussion. Be ready to collaborate on ideas, requirements, and every stage where the outcome of that stage has a bearing on the value of the final deliverable of the project. Simply put, a tester challenges sources of knowledge, whether these sources are stakeholders, users, developers, business stories, documents, or "received wisdom."

Whether you have a Shift-Left approach or not, your test strategy should encourage and align with these principles.

Creating a Test Plan

There are a number of ways that you can document your test plans. The simplest methods are usually the best. Emphasize creating lean, dynamic documentation that captures the key pillars of your test planning instead of writing lengthy plans.
One method for generating a lean test plan is mind-mapping. Here is how TestRail Product Manager Simon Knight writes about approaching mind maps for testing planning in his article, Test Planning Simplified.

Mind mapping your test plan

(1) Start with a central node. What needs to be delivered? What is the outcome your stakeholders are looking to accomplish? This should form the locus of your testing efforts.

(2) From this central point, create branches for the other key components of your test plan:

- **Testing scope —** What will you address with your testing (in scope)? What will you not address (out of scope)?
- **Timescale —** When will the testing start and finish?
- **Testing resources —** Who will do the testing? What will they need? Where will they do it?
- **Testing approaches —** How will the testing be carried out?
- **Risks and assumptions —** What obstacles can you foresee? How will those be addressed?

(3) From those branches, drill down further into the various items and activities. For the scope, you can take the requirements, features, or stories as being the next level down (sub-branches of the Testing Scope branch); and once you have those, you can drill further down into specific test cases, scenarios, exploratory sessions, or whatever is needed depending on your preferred testing style.

(4) Do the same thing with all the other nodes and branches until you have enough detail for a test approach that stands up to some level of scrutiny from your stakeholders and team.

Simon Knight, "Test Planning Simplified." https://dzone.com/articles/test-planning-simplified

Mind Maps

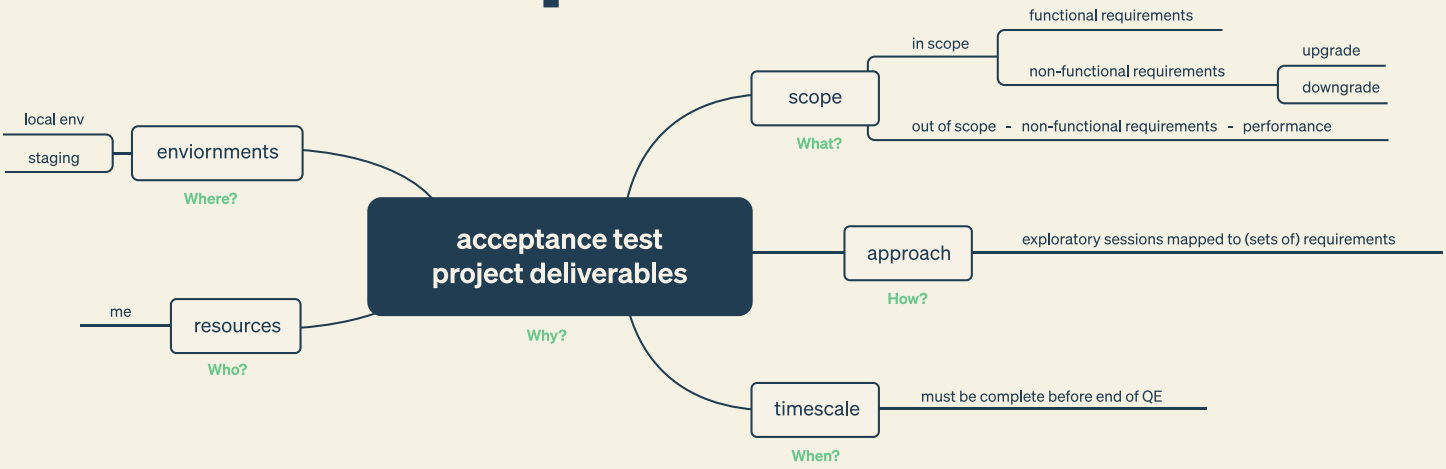
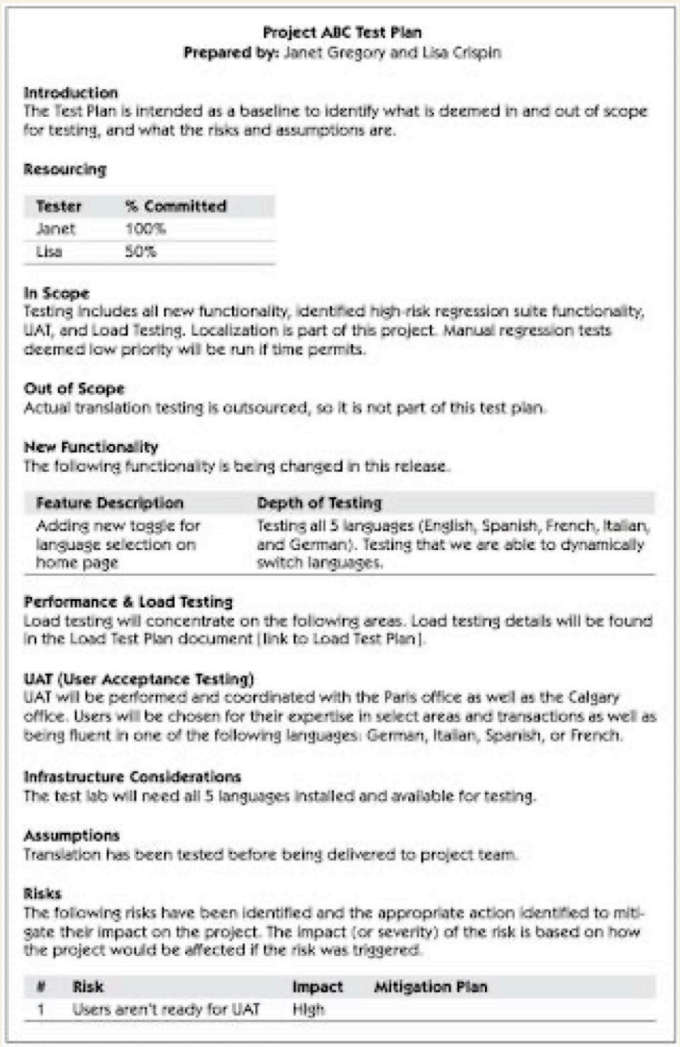


Image: Example of a test planning mind map. Image Credit: Simon Knight, "Test Planning in 2021, Part 1: Test Planning Simplified," 2021.



Project ABC Test Plan
Prepared by: Janet Gregory and Lisa Crispin

Introduction
The Test Plan is intended as a baseline to identify what is deemed in and out of scope for testing, and what the risks and assumptions are.

Resourcing

Tester	% Committed
Janet	100%
Lisa	50%

In Scope
Testing includes all new functionality, identified high-risk regression suite functionality, UAT, and Load Testing. Localization is part of this project. Manual regression tests deemed low priority will be run if time permits.

Out of Scope
Actual translation testing is outsourced, so it is not part of this test plan.

New Functionality
The following functionality is being changed in this release.

Feature Description	Depth of Testing
Adding new toggle for language selection on home page	Testing all 5 languages (English, Spanish, French, Italian, and German). Testing that we are able to dynamically switch languages.

Performance & Load Testing
Load testing will concentrate on the following areas. Load testing details will be found in the Load Test Plan document [link to Load Test Plan].

UAT (User Acceptance Testing)
UAT will be performed and coordinated with the Paris office as well as the Calgary office. Users will be chosen for their expertise in select areas and transactions as well as being fluent in one of the following languages: German, Italian, Spanish, or French.

Infrastructure Considerations
The test lab will need all 5 languages installed and available for testing.

Assumptions
Translation has been tested before being delivered to project team.

Risks
The following risks have been identified and the appropriate action identified to mitigate their impact on the project. The impact (or severity) of the risk is based on how the project would be affected if the risk was triggered.

#	Risk	Impact	Mitigation Plan
1	Users aren't ready for UAT	High	

Image Credit: Lisa Crispin & Janet Gregory, "Agile Testing," 2008.

Another helpful format for developing your test plan is what Lisa Crispin and Janet Gregory call a One-Page Test Plan in their book Agile Testing.

Crispin and Gregory call out 11 key components that you should evaluate when you are creating a One-Page Test Plan.

- **Resourcing** for testing
- Identification of what is **In** and **Out of Scope**
- Description of **new functionality** to be tested, including the **depth** to which that feature should be tested
- Any **performance or load testing** that needs to be performed
- How and when **user acceptance testing (UAT)** will be performed
- **Infrastructure** considerations
- **Assumptions**
- **Risks & mitigation plans**

Many lean test plans also include other key areas such as defining the **timeline** for your test sprint, identifying how you will handle **test automation** during the sprint, and how you will generate any **test data** that will be needed for testing.

Risk-Based Testing

One important output for any kind of test planning is identifying risks in your application. As a team responsible for quality at your organization, you need to make risks visible to management and propose reliable test methods to address these risks.

Three Types of Software Risks:

Project risk: These risks relate to the project in its own context. Projects usually have external dependencies such as the availability of skills, dependency on suppliers, constraints such as a fixed-price contract, or fixed deadlines. External dependencies are project management responsibilities.

Process risk: These risks relate primarily to the internals of the project and the project's planning, monitoring, and control. Typical risks here are under-estimation of project complexity, effort, or the required skills. The internal management of a project such as good planning, progress monitoring, and control are all project management responsibilities.

Product risk: These risks relate to the definition of the product, the stability (or lack) of requirements, the complexity of the product, and the fault-proneness of the technology which could lead to a failure to meet requirements. A product risk represents a mode or pattern of failure that would be unacceptable in a production environment.

To assess a product risk we need to understand what the consequence of that mode of failure is. If a failure happens, we say the risk materializes. How serious is the risk? We need to understand how exposed we are to that risk.

- The **probability** of a risk materializing. This value is typically expressed as a percentage between (but not including) zero and 100 percent.
- The **consequence** or **severity** of a risk materializing. This is the potential cost of the damage if this mode of failure happens.

The exposure of a risk—how serious a risk it is—is calculated as the product of the probability and the consequence/severity rank.

Risk-Based Test Planning:

Once you have identified a product risk of concern, formulate a set of tests to assess the likelihood of that risk materializing. The test approach will be to stimulate the failure mode in as many ways as possible. In so doing, you will see one of two outcomes:

Failures, which detect bugs to be fixed, thus reducing the risk from that mode of failure.

Passes, which increases confidence that a mode of failure is unlikely to occur.

Our tests, collectively, focus on ways in which a selected mode of failure can occur. If we use a risk assessment to steer our test activity, the testers' aim becomes explicitly to design tests to detect faults so they can be fixed, and in so doing, reduce the risk of a faulty product.

Fault detection reduces the residual risk of failures in production, where costs increase very steeply. When a test finds a fault and it is corrected, the number of faults is reduced and consequently, the overall likelihood of failure is reduced.

If we focus on critical features and find faults in these, undetected faults in these critical features are less likely. The faults left in the non-critical features of the system are of lower consequence.

Ultimately, risk-based testing assessments should guide how you **prioritize** certain types of testing around certain areas of your application. If you knew ahead of time what bugs would occur, how might that alter your test planning choices at the beginning? This is what risk-based testing does: it helps you to make better choices in test planning at the level of detail of your risk assessment.

The Fundamentals of Test Management

Designing Your Tests

Test design is the process by which we select – from the infinite number possible – the tests that we believe will be most valuable to us and our stakeholders. Our test model helps us to select tests systematically.

A test model might be a checklist or set of criteria. It could be a diagram derived from a design document or an analysis of a narrative text. In the case of system and acceptance testers, typical models are requirements documents, use cases, flowcharts, or swim-lane diagrams. More technical models such as state models, collaboration diagrams, sequence charts and so on also provide a sound basis for test design.

Using Models to Test

We use test models to:

Simplify the context of the test. Irrelevant or minor details are ignored in the model.

Focus attention on one perspective of the system's behavior. These might be critical or risky features, technical aspects or user operations of interest, or aspects of the construction or architecture of the system.

Generate a set of tests that are unique and diverse within the context of the model.

Enable the testing to be estimated, planned, monitored, and evaluated for its completeness, or "coverage."

Coverage measurement can help to make testing more manageable. If we don't have a notion of coverage, we may not be able to answer questions like:

- What has been tested?
- What has not been tested?
- Have we finished yet?
- How many tests remain?

Test models and coverage measures can be used to define quantitative or qualitative targets for test design and execution. To varying degrees, we can use such targets to plan and estimate. We can also measure progress and infer the thoroughness or completeness of the testing we have planned or executed. But we need to be very careful with any quantitative coverage measures or percentages we use.

Once you have developed your test design and test models, you should start to define which types of tests you are going to run and in what order. Start by outlining the key areas you need to test, and the priority of those tests.

Start drafting the tests that will cover your areas of highest risk, and test outwards from there

For example, here is the order in which you might prioritize the test sprint for a new feature being developed:

1. Functional smoke tests around a key feature
2. Smoke tests around key risk areas
3. Exploratory tests around new functionality
4. Regression testing (use similar priority-based approach)

You should use a tool to draft your tests that allows you to assign each test a priority level while you're drafting them, like Google Sheets or TestRail. Then, you can sort and filter your tests to ensure you are testing the highest areas of risk first when it actually comes to executing your testing.

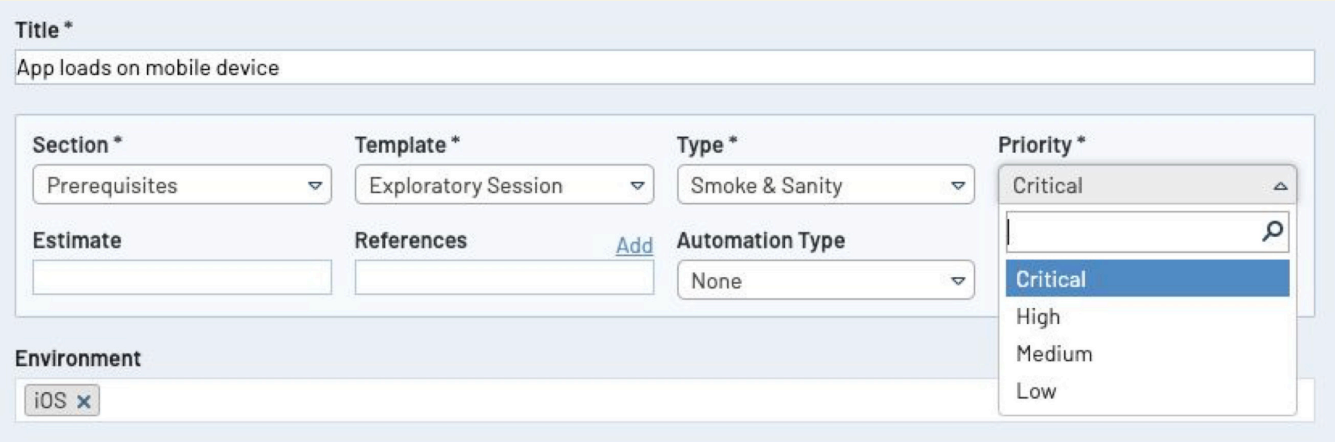


Image: Assigning priority levels in TestRail.

Once you have outlined your tests for a given sprint, you can then go back and add additional details as needed. For example, you may create different groups or sections of tests based on functional area or type of testing.

If you're using a test management platform like TestRail to write your test cases, you can also add custom data like development branch name, component, environment details, etc. You can also write out detailed preconditions, test steps, and expected results to document exactly what is in scope for testing and help identify potential defects more methodically.

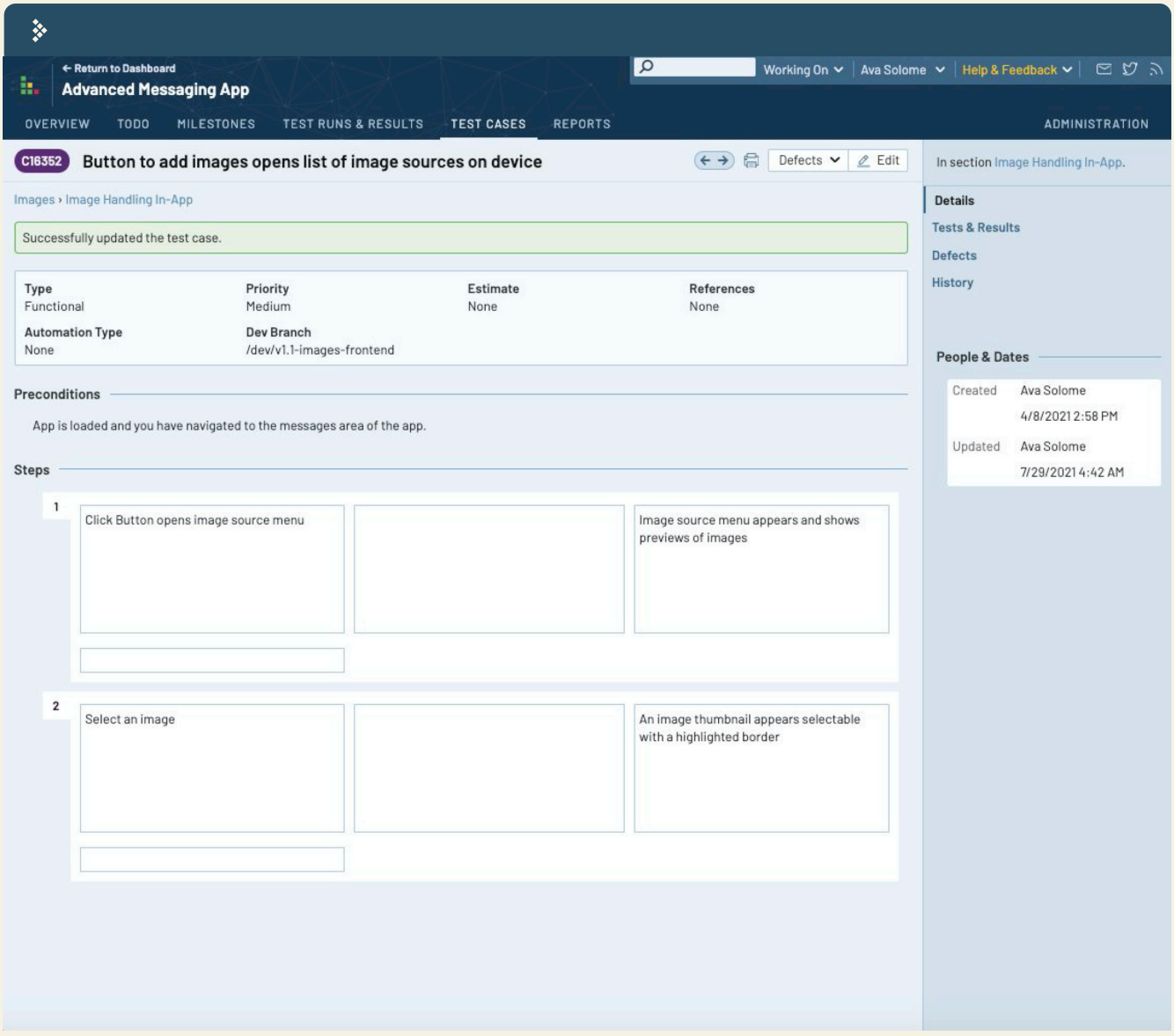


Image: For some tests, it is important to specify individual test steps and expected results, shown in the screenshot of a test case in TestRail above.

The Fundamentals of Test Management

Managing & Executing Your Testing

There are four keys to the successful execution of your plan:

1. **People –** is your team ready?

2. **Environments –** do you have the technologies, data, devices, interfaces to implement meaningful tests?

3. **Knowledge –** have you prepared your tests with an appropriate level of detail, or is your team ready and able to explore and test the system in a dynamic way?

4. **System Under Test –** is the software or system you are to test available?

The redistribution of testing triggered by the shift-left approach makes it clear that testers are not solely responsible for testing; testers don't own testing anymore.

Developers are adopting better test practices and visibility into their work. Good component-level or unit testing has specific goals that are distinct from system testing, so the scope (or amount) of system-level testing could be reduced. The correct distribution of testing goals and testing to meet those goals is the primary purpose of the test strategy.

Agile Test Interventions

The shift-left approach is fundamental to a test strategy for agile projects. In an agile context, test strategy can be viewed as a series of test interventions. There are critical moments in all projects where opportunities to gather and give feedback present themselves. The tester focuses on these critical moments and is ready to contribute at those times.

In your own projects, identify the critical moments where intervention is possible, and the choices that you and your team can make. For example, should the tester write unit tests for developers? Should you provide examples to get them started or coach them to improve their testing ability? Only you and your team can decide this.

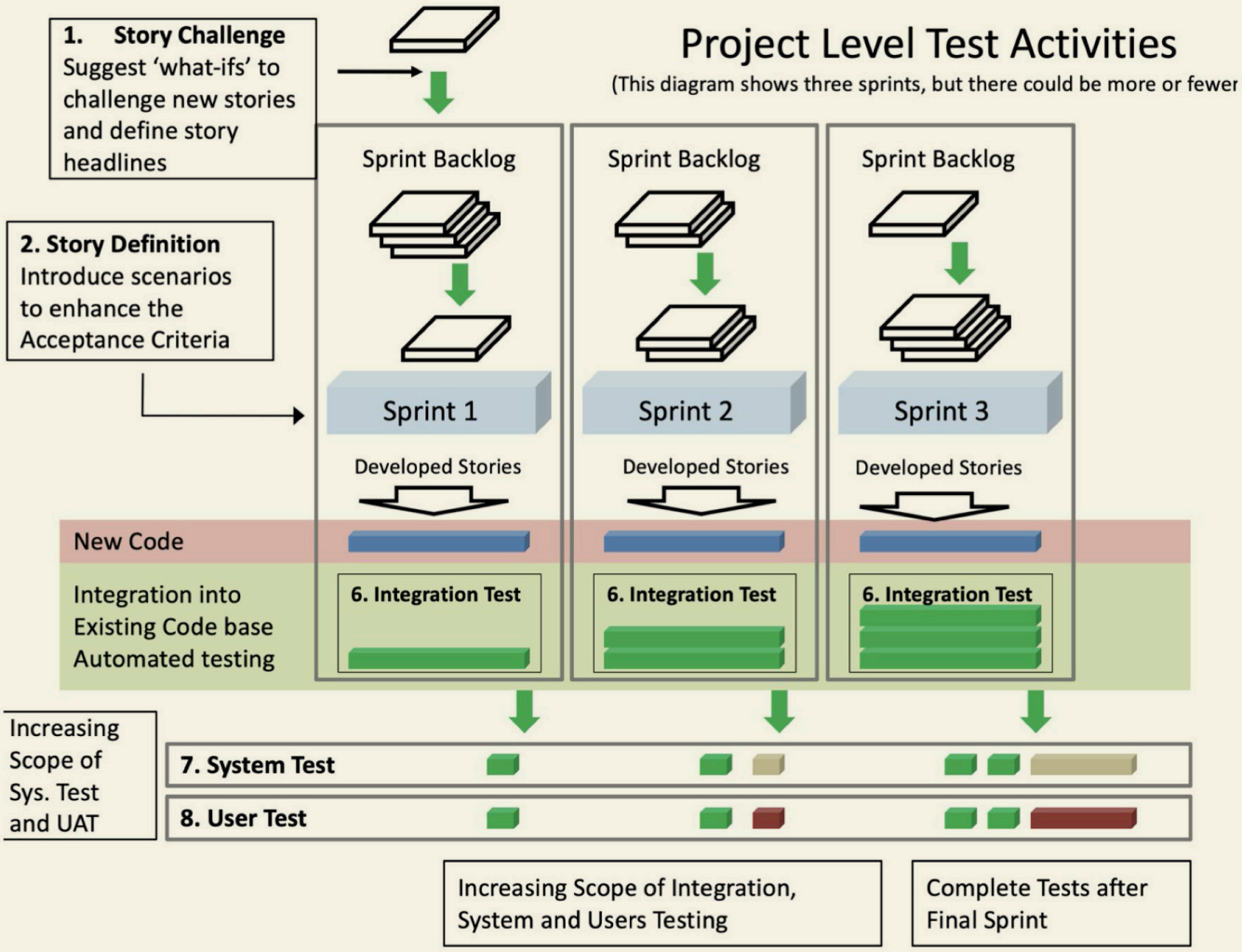
In the table below, you can see the typical intervention types. You might have more or less interventions active at different points in your own unique process.

Step	Level	Activity	When?
1	Project	Story Challenge	As stories are added to the Product Backlog
2	Project	Story Definition	As stories are added to a Sprint Backlog
3	Sprint	Daily Stand-Up	Once per day during the Sprint
4	Sprint	Story Refinement	Occurs throughout the Sprint as new information emerges
5	Sprint	Developer Testing	Occurs throughout the Sprint as the developer codes the stories
6	Sprint	Integration (and incremental System) Testing	During and at the end of each sprint, including the final sprint
7	Project	System Testing	At the end of each sprint, including the final sprint
8	Project	User Acceptance Testing	At the end of each sprint, including the final sprint
9	Project	Non-functional Testing and Pre-Production Testing	As needed.

Project Level Interventions

As shown in the table above, interventions occur at either the project-level or at the level of a sprint. The diagram on the next page shows a project-level view and the five key project-level interventions.

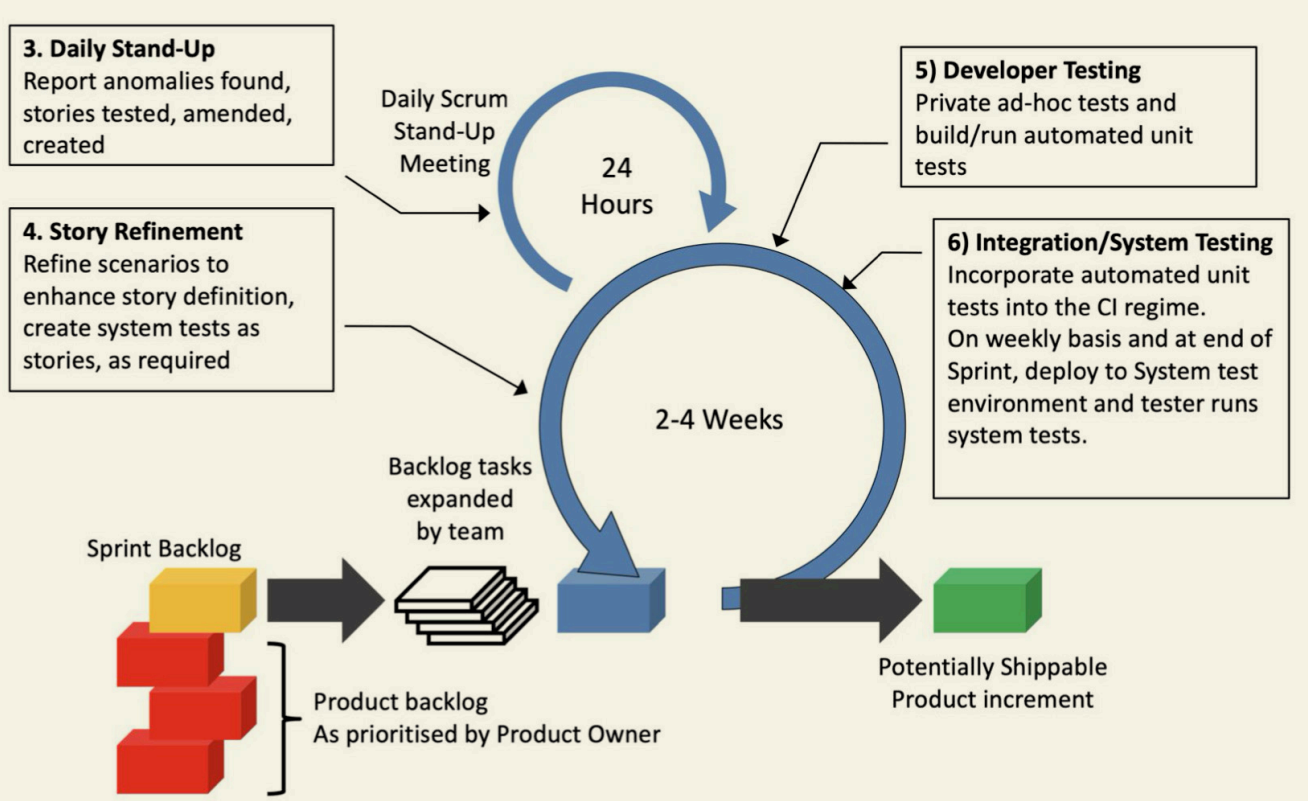
- Story challenge (1) is where the tester validates a user story.
- Story definition (2) is where a tester validates proposed acceptance criteria for a story.
- Integration tests (6) check that new features link correctly with other features and the system as a whole.
- System tests (7) and user acceptance tests (8) are conducted as appropriate.


Project Level Test Activities
(This diagram shows three sprints, but there could be more or fewer
1. Story Challenge
Suggest 'what-ifs' to challenge new stories and define story headlines
2. Story Definition
Introduce scenarios to enhance the Acceptance Criteria
Sprint Backlog
Sprint Backlog
Sprint Backlog
Sprint 1
Sprint 2
Sprint 3
Developed Stories
Developed Stories
Developed Stories
New Code
Integration into Existing Code base Automated testing
6. Integration Test
6. Integration Test
6. Integration Test
Increasing Scope of Sys. Test and UAT
7. System Test
8. User Test
Increasing Scope of Integration, System and Users Testing
Complete Tests after Final Sprint

Sprint-Level Interventions

A project usually has multiple sprints. The diagram below shows the four sprint interventions that are repeated for each sprint:

1. The daily stand-up (3) is an opportunity to report progress, raise concerns, identify risks, or discuss questions raised and answers received during the sprint.
2. Story refinement (4) and contributions to developer testing (5) are day-to-day activities that occur as part of discussions with users, analysts, and developers.
3. The tester incorporates developer and new system tests into a growing collection of tests to be automated (6).



Identify the critical moments, propose your contribution, and negotiate with your team. You offer more test leadership and guidance rather than volunteering simply to take on responsibility for the testing work. It will be much easier to demonstrate your value to the team if you take this approach.

Effective testing requires curiosity, persistence, and a nose for the problem. Your goal is to stimulate failures along with the evidence required to trace those failures to defects so that they can be fixed. Although finding (and fixing) defects is good for the quality of the product, communicating defects often feels like you are giving bad news to someone. This could be a developer who has made a mistake somewhere and now has a defect to fix. But you could also be reporting to stakeholders that some critical functionality does not work correctly, the system is not ready, and/or delivery will be delayed.

As tests are run by the team, use a tool that allows you to record the status of tests and display the progress of testing in real time. All tests run would have the tester identified, the date/time, and test steps noted.

Passed tests might be assigned a simple pass status. Failed, blocked, or anomalous test results might have screenshots, test results assigned, and an incident report assigned. Many tools provide hooks to test execution tools that manage and run tests, log results, and can even create draft incident reports.

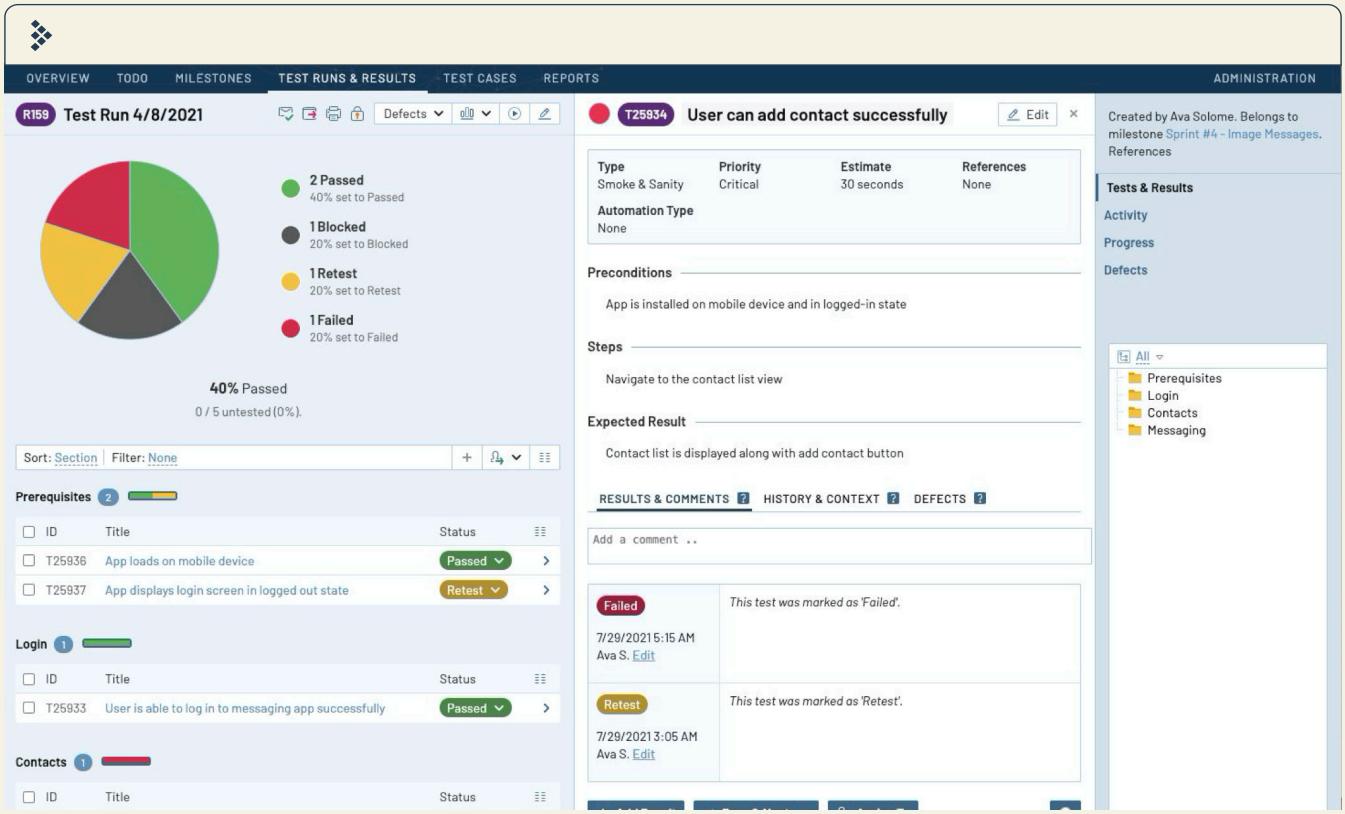


Image: TestRail test runs and results.

While capturing test results and recording findings is important for the sake of measuring progress and producing a historical record of testing, the most important reason to document your test steps and results is to be able to quickly submit bug reports if you have identified a fault in the system under test.

By integrating a test management tool like TestRail with issue-tracking systems like Jira, you can streamline the process of alerting R&D to defects as soon as they are discovered. For example, Push Templates in TestRail allow you to define the information you would like to include in your defect report when you push it from TestRail to a new issue in Jira, like the summary of the test you were running when you discovered the defect, what code branch you were testing, what test steps you had taken, and any comments or attachments you added during the test process.

Then, when you push new defects to Jira, TestRail will automatically send the information you defined in your push template without you having to task-switch and spend time copying and pasting data between systems, reducing the chance for human error and helping you speed up the process of reporting new defects so that the engineering team can take action more quickly.

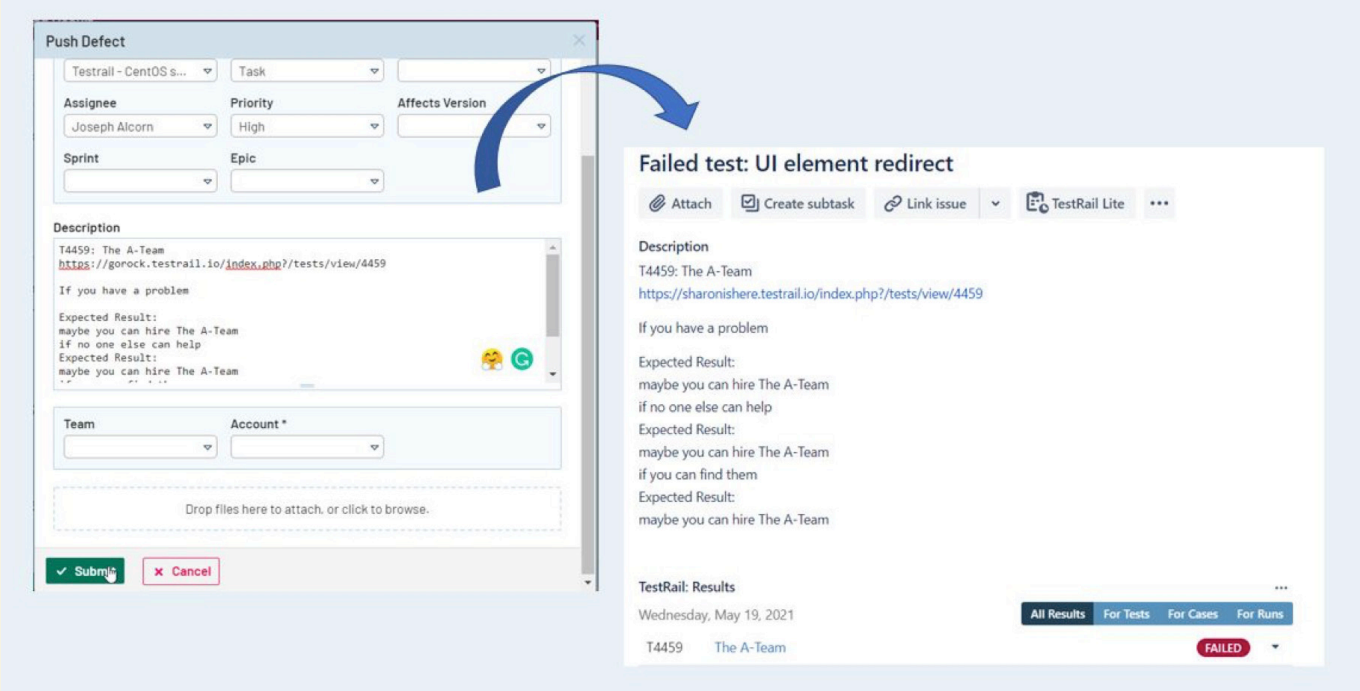


Image: Look for ways to save time, control for human error, and eliminate duplicative work. For example, with TestRail, you can log an issue in Jira right from TestRail that automatically contains all of your comments, steps to test, intermediate results, etc.

Conclusion

The fundamentals of test management start with the definitions of the word test: a procedure for critical evaluation, a means of determining the presence, quality, or truth of something, and a trial. These three varying definitions of test set the foundations needed to define your test strategy.

Your strategy is a result of exploration, thinking, and collaboration. A strategy seeks to define the process you will use to achieve your testing goals by presenting decisions that can be made ahead of time, defining the process that will allow decisions to be mad,e and setting out the process to follow for uncertain situations.

A test management strategy attempts to answer as many questions as possible ahead of time. Whether you have a Shift-Left approach or not, your test strategy should encourage and align with these fundamental principles and promote feedback that will help the team to understand, challenge, or improve goals, requirements, design, or implementation.

In the end, effective testing requires curiosity, persistence, and a nose for the problem. Your goal is to stimulate failures along with the evidence required to trace those failures to defects so that they can be fixed.

If you're involved in software development, hopefully this ebook has provided you with a range of essential test management topics (including process, modeling, risk, delivery, reporting, and tools) as well as tactical ideas to help you apply the concepts presented.

Acknowledgements

Paul Gerrard was the principal contributor to this guide. Paul is a consultant, teacher, author, webmaster, developer, tester, conference speaker, rowing coach, and publisher. He has conducted consulting assignments in all aspects of software testing and quality assurance, specializing in test assurance. He is Principal of Gerrard Consulting Limited and is the host of the UK Test Management Forum and the UK Business Analysis Forum.

Additional contributors include Jackie King, Hannah Son, Simon Knight, and Matt Caponigro.

About TestRail

TestRail helps quality assurance (QA), engineering, and development teams speed up testing, improve product quality, and ship releases faster. More than 10,000 organizations like NASA, Apple, Microsoft, Activision Blizzard, and Amazon trust TestRail to power their QA and test management processes.

TestRail is the flagship product of Gurock Software GmbH. Gurock was founded in 2004 and our globally distributed team focuses on building and supporting powerful tools with beautiful interfaces to help software teams around the world ship reliable software.

Gurock is part of the Idera, Inc. family of DevOps tools, which includes Xray, Ranorex, Kiuwan, Travis CI, Assembla, and, PreEmptive. Idera, Inc. is the parent company of global B2B software productivity brands whose solutions enable technical users to do more with less, faster.