The Definitive Guide To

User Acceptance Testing (UAT)

BY ASH CONWAY

CEO & Founder of Bugwolf

Bugwolf

www.bugwolf.com

Introduction To This Guide

In London, up until 1992, human dispatchers did the job of sending ambulances in response to emergency calls.

That year, an automated dispatch system was introduced to make the process more efficient. However, this system was rolled out in a slipshod manner without sufficient testing.

The system started glitching a few days after it was introduced. The cause, traced to memory leaks, meant that ambulances could not be sent when they were needed. Eight days later, when the system finally stopped working, 46 people had lost their lives because of delayed response times.

Bear in mind that this was in 1992 when the Internet was an obscure academic and military project and almost all systems running the world were manually operated.

Today, when software has eaten and digested the world, poorly tested information systems have wiped out 20% of the US stock market in 1987, scuppered the travel plans of half a million people in the UK in 1999, and grounded the trillion dollar F-35 program because of faulty targeting systems.

These problems crop up because software isn't rigorously tested in real-world scenarios.

This is why user acceptance testing (UAT) is so important.

Why Did We Write This Guide?

Many digital teams test their own products and pass it off as UAT. With no third party vetting, this process is akin to a pharma company releasing a new drug in the market without getting approvals from the FDA.

Rigorous UAT gives you a stable product which works as expected, and is much cheaper to operate and maintain in the long run.

This guide will walk you through the entire UAT process. It will tell you how to plan, execute, and report the actual tests, interpret the test results, build and manage a UAT team, and help you sidestep potential minefields so that you can improve customer experience, increase revenues and sales, and maintain uninterrupted business operations.

Who Is This Guide For?

We have written this guide for senior personnel in large enterprises, digital teams in government departments, and decision makers in late-stage startups. This guide draws upon our experience of working with leading brands like Australia Post, National Australia Bank, Treasury Wine Estates, and many up-and-coming startups.

How Can You Use This Guide

If you are new to UAT, we recommend that you start from the beginning as each chapter builds on the previous one. However we have gone really deep with this guide, and you can jump in anywhere at the middle and follow along if you are looking for information on a specific topic.

Interesting Data Points About Software Fails & Software Testing

The number of incidents of software fails continues to increase. According to a study conducted by iSentia, the Asia-Pacific region's leading media intelligence company, on behalf of Bugwolf:

- 786.8 million people were affected by issues with malfunctioning software between July 1, 2016 and December 31, 2016.
- These failures equated to a $4,520,141,144 (AUD) burden on the global economy.
- The total time cost of these disasters in terms of downtime, lost productivity, and lack of service equates to **46 years, 9 months, 2 weeks, 6 days, 6 hours, and 11 minutes.**

The proliferation of technologies like mobile, cloud and IoT has lead to a paradigm shift in terms of QA budgets and priorities. According to Capgemini's World Quality Report **2016:**

- **60%** of the QA and testing budgets are allocated to new technologies like cloud, mobile, BI and IoT in 2016, up from **53%** in 2015.
- **48%** of companies have no testing processes for mobile and multi channel applications, **46%** don't have an in-house testing environment, and **45%** don't have the right tools for testing.
- **34%** of companies are focusing on testing functionality, down from **54%** in 2014 because most developers leave that part to actual users.

CHAPTER

01

What is UAT And Why Is It So Important?

Defining Of UAT

You wouldn't trust the emission and fuel efficiency figures generated by laboratory tests of a car, right?

There is a world of difference between lab conditions and real-world conditions.

These figures will differ widely for the same make and model depending on variables like the driving style of the individual driver, the exact configuration of aftermarket consumer comfort systems, driving conditions (rush hour traffic vs. rural roads), degree of maintenance or weather conditions.

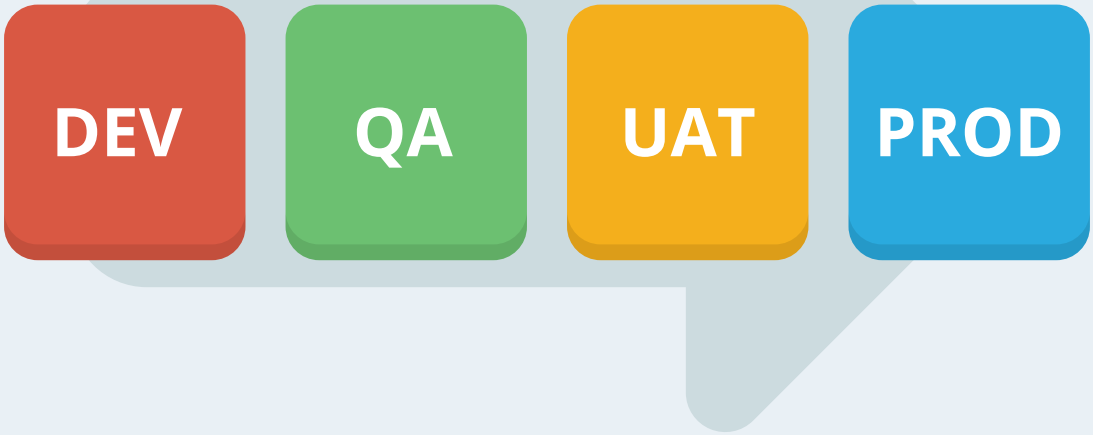
Complex software systems are the same.

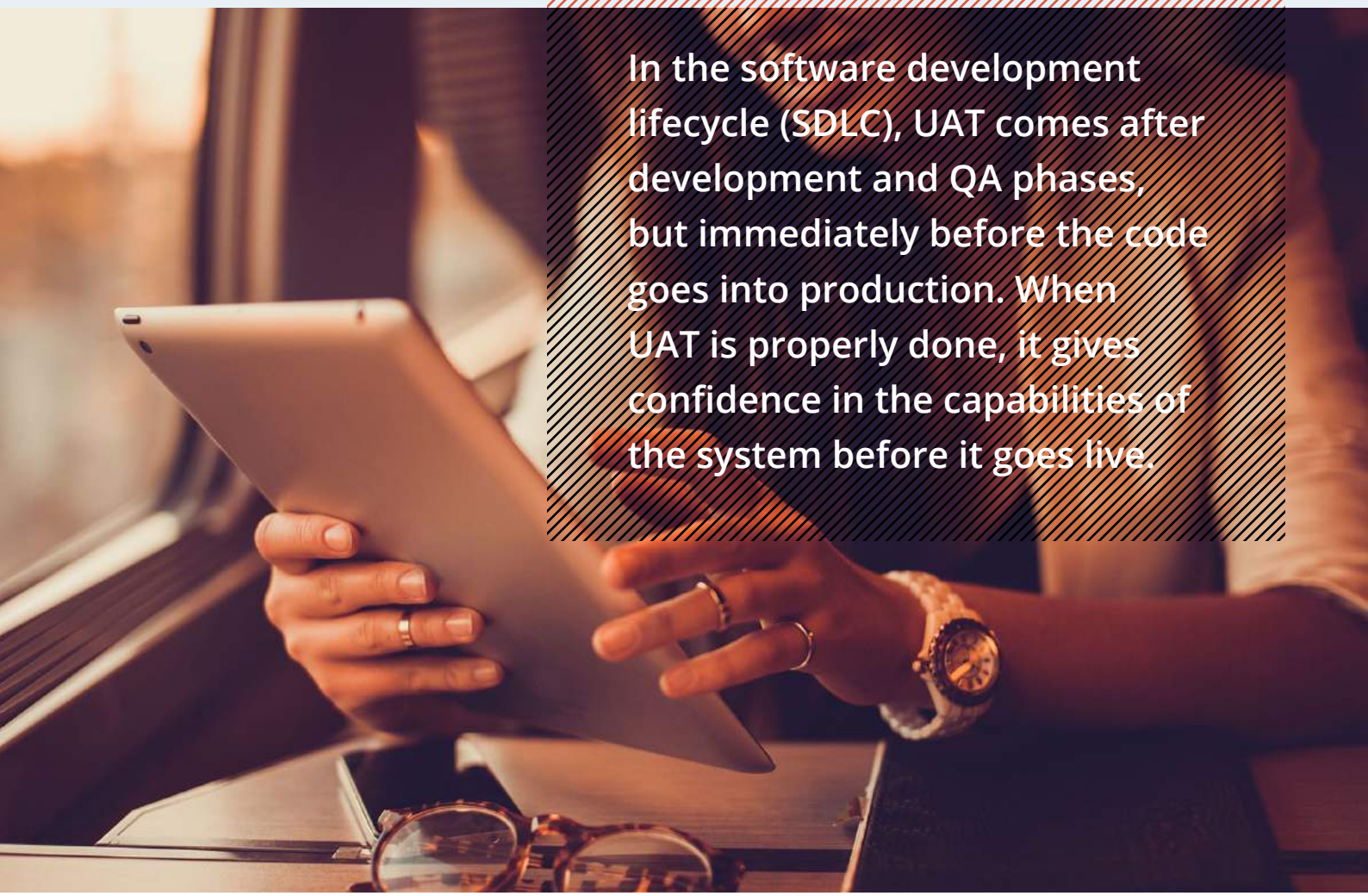
Regardless of how many functional tests they pass you will never know how the system behaves unless it's tested in real world conditions by actual users.

These tests are called User Acceptance Tests, and the process is called UAT.

Unlike functional testing, UAT takes both software performance and human behavior into account.

When is UAT performed?



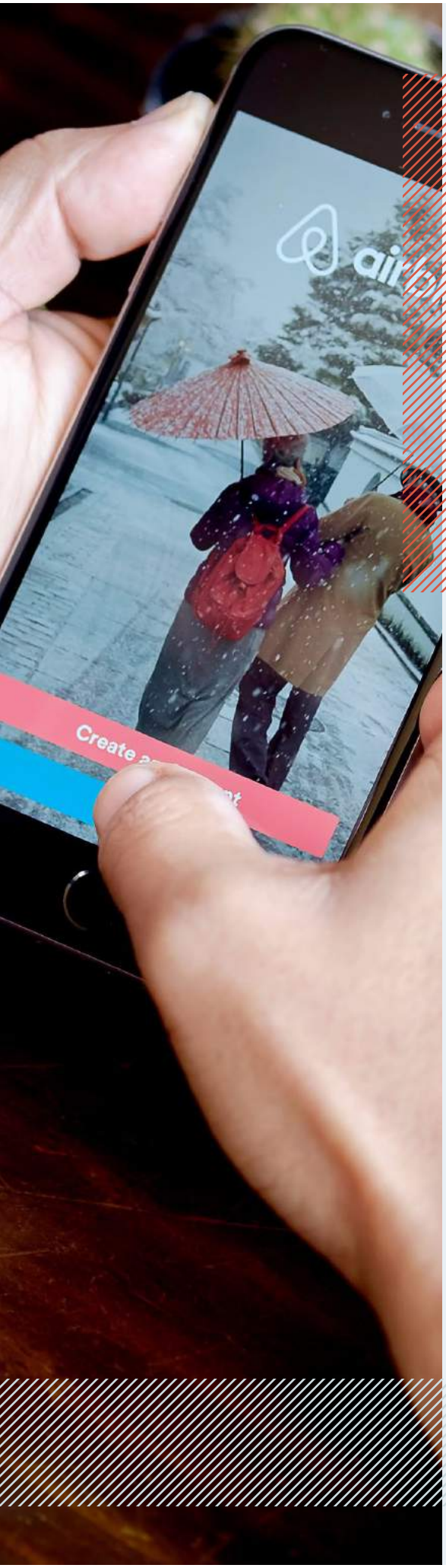


In the software development lifecycle (SDLC), UAT comes after development and QA phases, but immediately before the code goes into production. When UAT is properly done, it gives confidence in the capabilities of the system before it goes live.

Pros Of UAT	CONS OF UAT
Uncovers operational stresses that get overlooked during the functional tests of the application.	Requires some upfront investment in terms of setting up test environments and training users for the tests
Saves the cost of remediating bugs with potential to wreak havoc on the system by a factor of 15x-100x	Needs well trained users and specific expertise in design, planning, execution, and reporting for user acceptance tests
Shields the organisation from potential PR nightmares and legal liabilities arising from unpredictable software	Unless well planned, could divert valuable time of testers who would be otherwise engaged in revenue generating activities
Helps users assess the capabilities of the system in terms of compatibilities with existing business processes	Might result in some short term delays in system rollout for fixing critical bugs uncovered during the process
Lets organisations determine any operational hurdles in advance of rollout	Requires testers to learn new technologies or skills which might have no relevance to their day-to-day activities
Catches bugs early into the SDLC making them relatively inexpensive to fix.	
Drastically cuts down on customer complaints and support issues down the line, and improves retention and NPS scores	
Stakeholders can better understand the needs of the target market based on feedback from UAT	
Eliminates communication gaps between the project owners and vendors and gives all stakeholders clarity about the business requirements or feature changes	
Creates evangelists and champions out of users as they see the benefits of the new system first hand	
Reduces the burden on developers, who might not be subject matter experts, to get features and workflows aligned with real-world environments	

The pros of UAT outweigh the cons. That said, you will have to take an informed decision on running UATs depending on whether your organisation can support the process at a specific point of time.

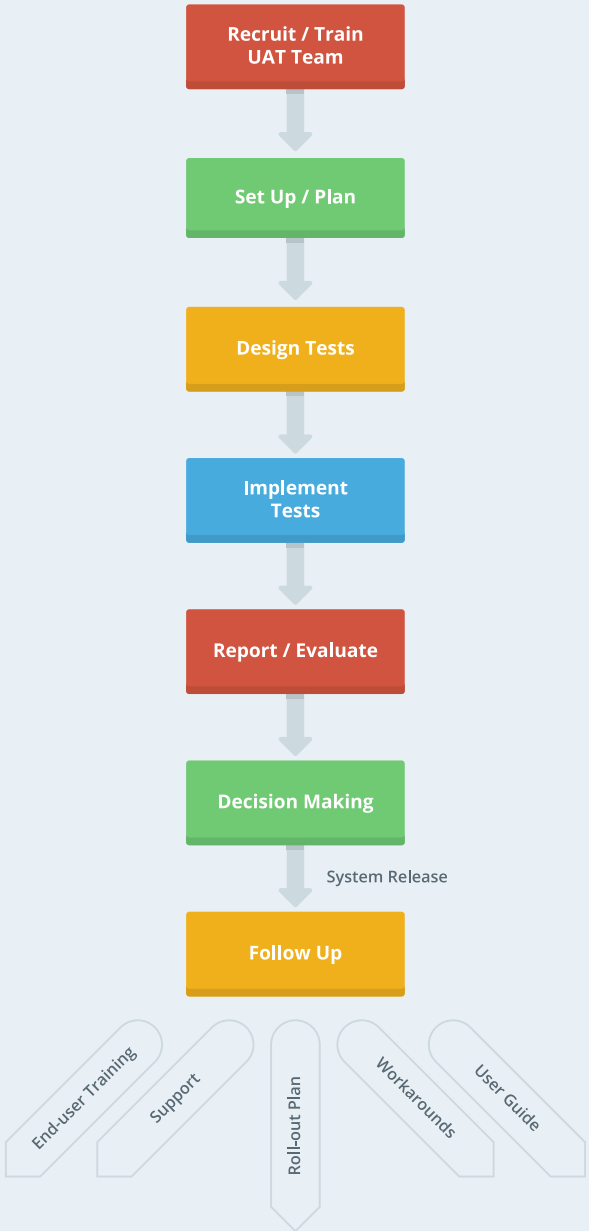
Like most things valuable UAT requires upfront investment. But the gains from a well run UAT program can deliver exponential returns on that investment, especially when you consider the crippling reputation and business costs associated with unreliable systems.



The UAT Workflow

At its most basic, a UAT process has three components: plan, design and implement.

The diagram below gives a high level overview of the entire testing process.



In brief, here's what the process is about:

Recruit And Train UAT Team

Unlike functional tests, UAT should be conducted by end users. Depending on their exact job profile they might not be technically savvy or have any kind of familiarity with testing processes and software. Unless you vet and train these users properly there's always the danger of usability tests going off the rail.

Set Up Plan

In this phase, the general plan of attack is determined. In this stage you should identify the purposes and business goals of the project, and gather business requirements.

Design Test Cases

In this phase, test cases are designed to closely mimic real world situations. These test cases will be designed whilst keeping the business requirements in mind.

Implement Tests

At this point, you will use the system and the test environment to execute all the test cases identified in the previous step. During this stage, the users will communicate with stakeholders and the development team about the status of the system and any high level corrections to be made.

Report/Evaluate

After the tests are completed the team will gather the results to determine whether they meet the acceptance criteria. This step is vital for determining next steps.

Decision Making

Based on the evaluation of the usability test results, a high level decision will have to be made about how to address the shortcomings of the system. This may take the form of redesign of features, better documentation, or more comprehensive end user training.

Follow Up

In this stage, the UAT owners (typically the managers) co-ordinate with other stakeholders like the sponsors and the developers so that the accepted changes are implemented in the system.

We will take a detailed look into each of these phases in the later chapters.

People Involved In UAT

Because of the manual nature, UAT involves multiple stakeholders both within and outside the organisation.

Here's a rundown of the most important characters:

1. The Sponsor (i.e. the person or group who commissions the system or defines the business goals).

Depending on the size of the company, the sponsor will be either the owner who signs the cheques, or an executive who is accountable for outcomes of the project.

The sponsor will focus on identifying potential risks and barriers to success so that these can be eliminated and a positive ROI realised in terms of revenue and profit. The sponsor will also set the success criteria and and define test scenarios at a high level.

2. The Manager/Managers, who will be responsible for delivering business results from the system in the real world. The business managers will go into more details, examining the system for compatibility with existing business processes.

Based on the high level test scenarios outlined by the sponsor the business managers will design tests which mimic the interactions of actual users over a realistic time period.

The UAT test results will also serve as a benchmark for performance of the new system.

Apart from business managers, other individuals in management roles might also be involved, like quality managers (responsible for meeting quality standards of the new system) and test managers (responsible for the planning and execution of the actual tests).

3. The End Users who will actually operate the system. Depending on the size of the organisation or the context of use, there will be different types of end users.

For instance, consider the inventory management system at Amazon. That system will have multiple users inside Amazon, and users at each of these suppliers who will need access to the system to manage the shipments and fulfill the orders.

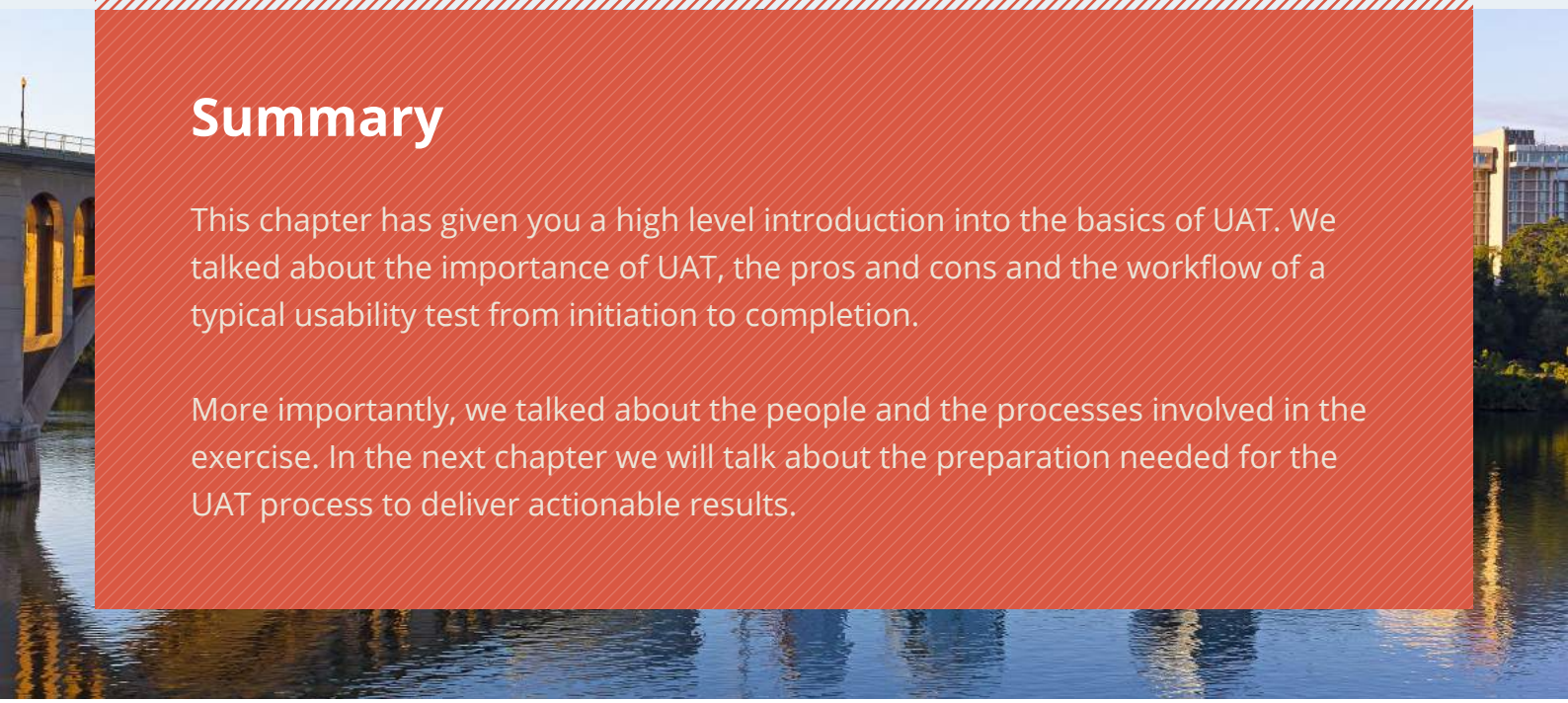
Because these two groups will have different expectations of the system, comprehensive UAT would happen only if the users responsible for testing are drawn from various possible user types. The end users are primarily responsible for designing the test cases and running the tests.

While designing these tests the end users should also focus on identifying boundary conditions using realistic data to test the resilience of the system.

4. The Developers responsible for supporting the entire testing process. They are required for familiarising the actual testers with the features of the system and evaluating data generated from the tests.

The outcomes of the entire UAT exercise will depend on how fast can the developers fix the issues uncovered through these tests. Without the active involvement of developers, the entire exercise is meaningless.

They are also responsible for setting up the test environment and keeping the system stable and usable.



Summary

This chapter has given you a high level introduction into the basics of UAT. We talked about the importance of UAT, the pros and cons and the workflow of a typical usability test from initiation to completion.

More importantly, we talked about the people and the processes involved in the exercise. In the next chapter we will talk about the preparation needed for the UAT process to deliver actionable results.

CHAPTER

02

Getting Started With UAT: Business Requirements

The User Expectations Of Software

It's not enough to make software that's secure, functional, and reliable: that's just the basic requirement.

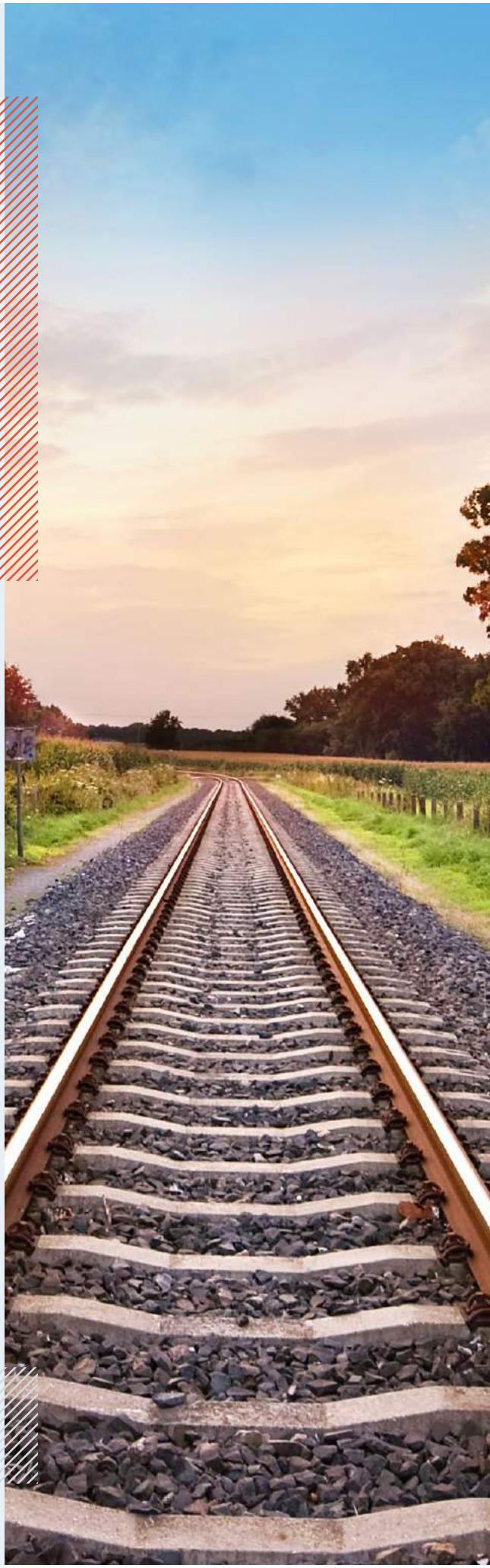
Users of enterprise software, for instance, have long complained about the poor user experience, the inflexibility, and the lack of usability of existing tools. A survey by Forrester found that:

- 75% of users couldn't easily access information from existing enterprise systems.
- 69% of enterprise employees want an engaging mobile-first experience but only 55% enterprises have implemented three or less mobile apps.
- Because of poor design 62% employees delay tasks which need them to log into multiple systems, affecting overall efficiency and outcomes.

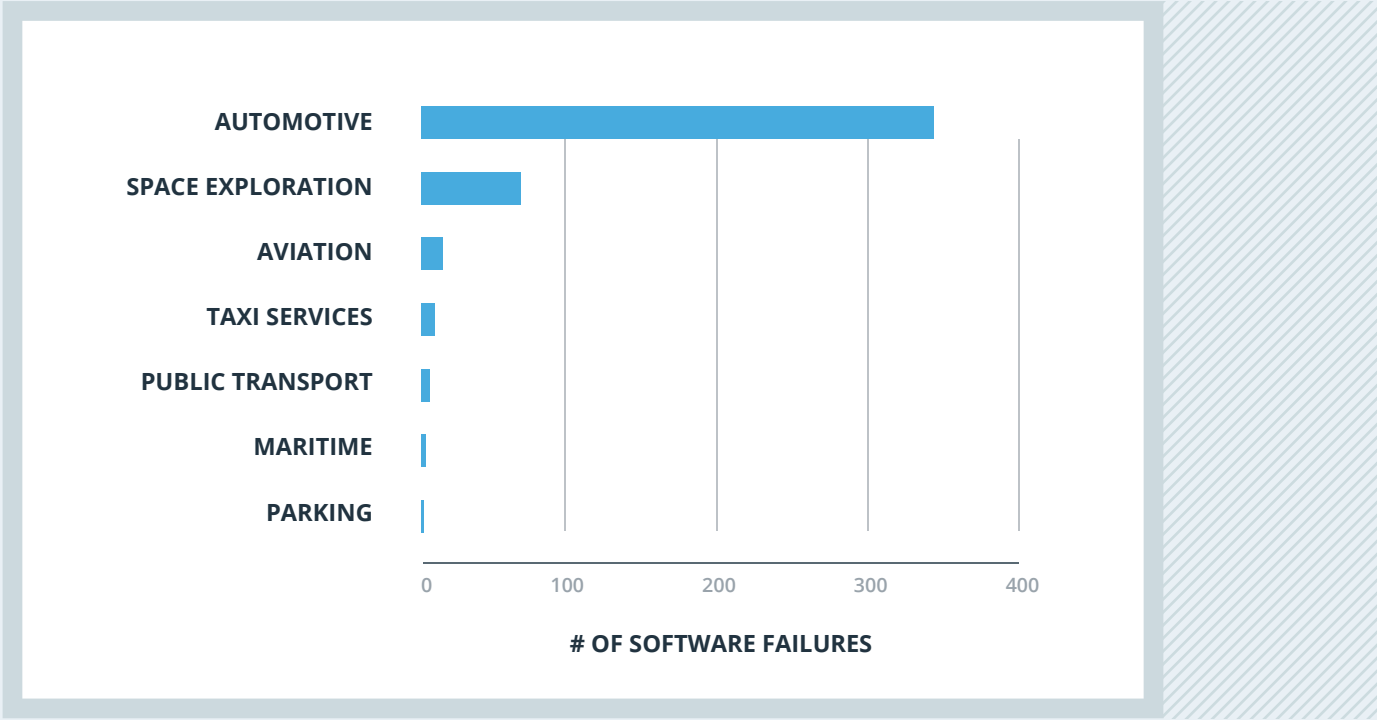
The usability issues with existing enterprise tools have contributed to the shadow IT phenomenon where enterprise users are increasingly using user friendly third party tools like Dropbox or Slack instead of sticking to officially-approved software, sometimes with serious security and data governance repercussions.

However use-centered design isn't always a nice to have about the product. Sometimes, it's the product in itself: a confusingly designed Internet banking application will make customers jump ship even if the bank offers attractive interest rates or better perks compared to the competition.

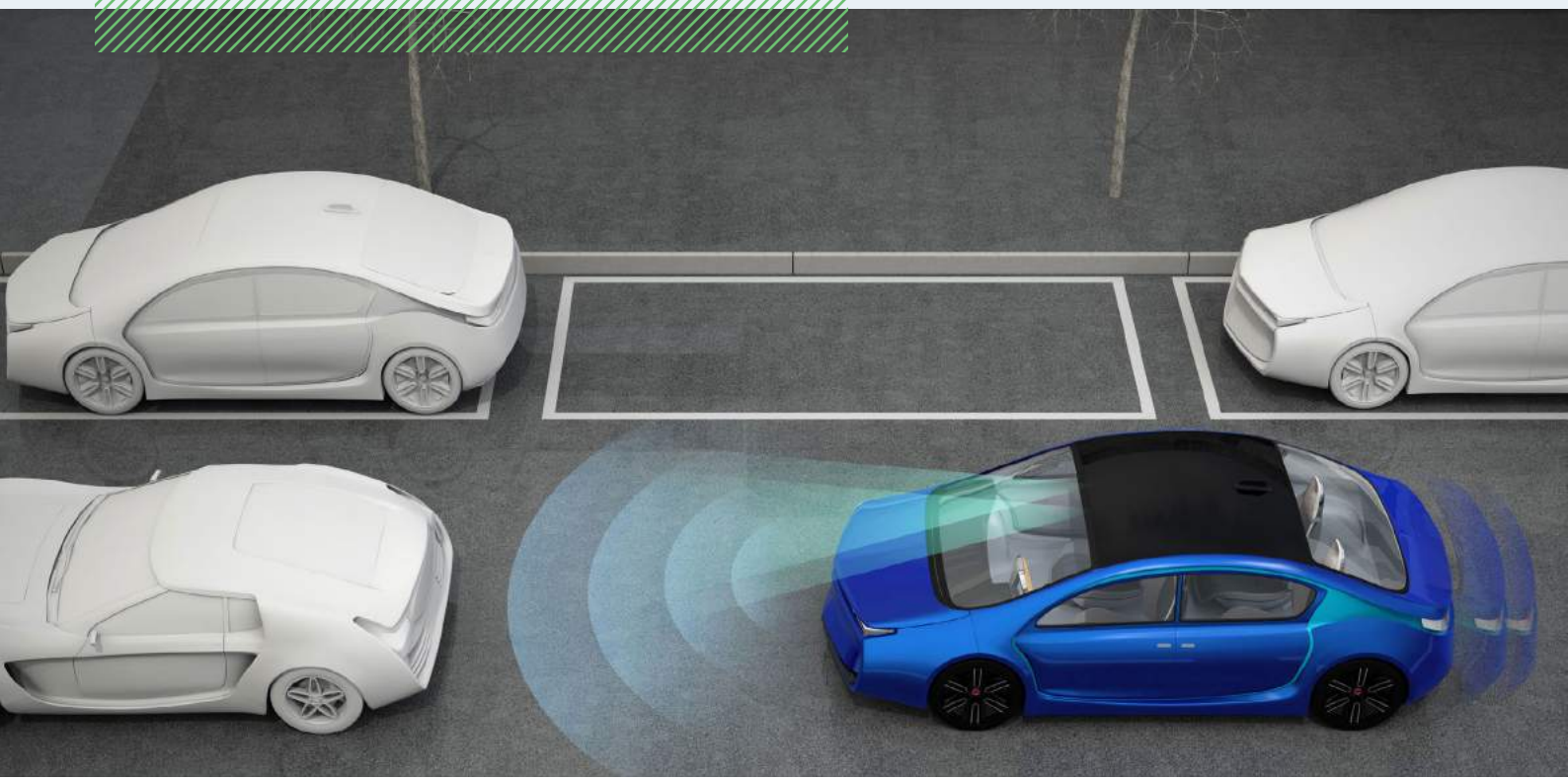
Poorly designed software has real-world implications beyond a user spending twice as much time trying to understand how a system works.

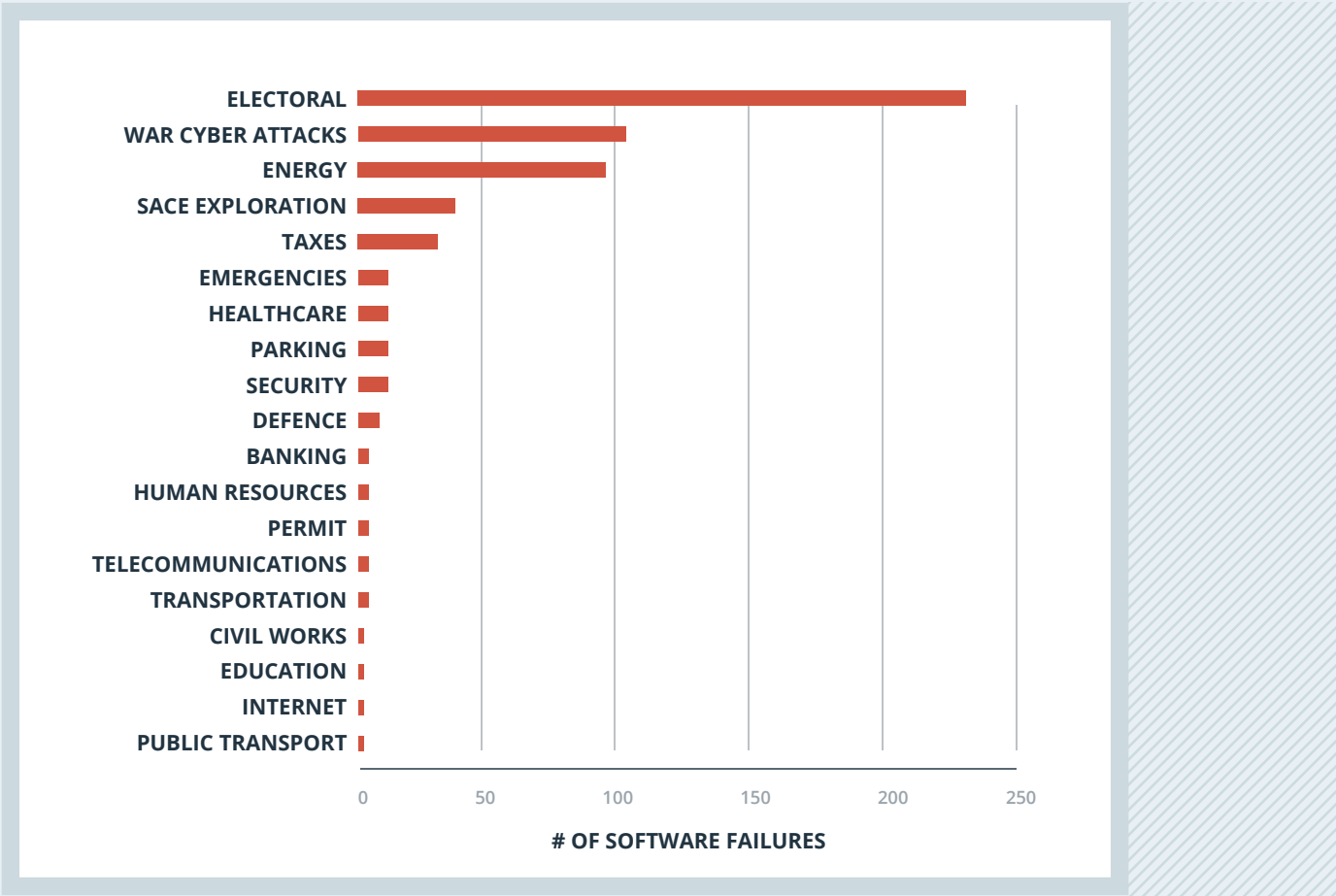


The table below shows the number of incidents associated with transportation software. These failures resulted in $455,451,946 (AUD) worth of damage to the economy, business and customers.



The government sector has seen the maximum number of fails (for example: the Australian Census blunder or the US Healthcare.gov debacle) this year, with far reaching impact for millions of people who depend on public sector services.





On the whole, the cost of software failure has risen from 2015 to 2016 in terms of people affected, assets impacted, and companies afflicted.

The seeds of software failure are sown early in a project, when business requirements are not managed properly (CIO magazine found the numbers of failed projects to be as high as 71%) or when the end user doesn't have a say in the design and execution.

Prioritising Business Requirements

So if you want to set up your project for success you will have to focus on getting your requirements right. In the context of UAT, the sponsor is in charge of setting the business requirements which will then be made into test cases.

The usability tests will have both functional as well as non-functional (stress, reliability, performance, speed, etc.) requirements to be tested.

One way to prioritise business requirements and user stories is to use the MoSCoW method, which Wikipedia defines as:

> *"a prioritisation technique used in management, business analysis, project management, and **software development** to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement"*

The MoSCoW acronym breaks down as:

Mo: Must have this test done.
S: Should run this test, if possible.
Co: Could run this test if other issues are fixed.
W: Would run this test if possible in the future.

This arrangement makes it easy for sponsors to eliminate any kind of confusion while drawing up business requirements.

This prioritisation ensures that the most important tests are conducted first, and more importantly, tests which don't really matter in the larger scheme of things are deferred for a later date.

Given how expensive and time consuming the UAT process can become, this process guarantees the highest impact and keeps UAT cycles short and manageable.

UAT Acceptance Criteria

The UAT acceptance criteria (UAC) is a series of simple statements that distill the business requirements and give stakeholders an idea of the time and costs involved in the entire project.

When you get your UAC right you will be laser focused on your testing processes and not embark on a wild goose chase.

Here's an example of user acceptance criteria as applied to an Internet banking scenario.

Acceptance Criteria 1

Given (inputs/preconditions)	Given that the customer has one savings account
When (actions/triggers)	When they have logged in successfully
Then (outputs/consequences)	Then, the account details are listed in the following order (Account number, Name of Branch, Available Balance)

User Acceptance Test 1 (Positive)

Verify that preconditions and actions occur Verify that output is generated	**Steps:** 1. Verify that the customer has one savings account 2. Verify that the customer has logged in successfully 3.Verify that the account details are listed in the following order (Account number, Name of Branch, Available Balance)

10% Acceptance Criteria Fulfilled

User Acceptance Test 2 (Negative)

Verify that invalid inputs occur Verify that trigger doesn't occur Verify that outputs are not generated, and an error message is displayed	**Steps** 1.Verify that the customer doesn't have a savings account 2. Verify that the customer cannot log in 3. Verify that an error message is displayed

20% Acceptance Criteria Fulfilled

User Acceptance Test 3 (Negative)

Verify that preconditions occur Verify that actions or trigger do not occur, and error message is displayed	**Steps** 1.Verify that the customer has a savings account 2. Verify that the customer cannot sign in successfully 3. Verify that error message is displayed

30% Acceptance Criteria Fulfilled

User Acceptance Test 4

Select attribute: performance	Verify that response time is less than 3 seconds

40% Acceptance Criteria Fulfilled

User Acceptance Test 5

Select attribute : security	Verify that login process is secure

50% Acceptance Criteria Fulfilled

User Acceptance Test 6

Select attribute: availability	Verify that the portal is online 24/7

60% Acceptance Criteria Fulfilled

User Acceptance Test 7

Verify that the system is working for all users	Run tests 1-6 for each user

100% Acceptance Criteria Fulfilled

If you were to use a decision tree, this is what it would look like:

Acceptance Criteria

Given	When	Then
a. Input **b.** Preconditions	**a.** Triggers **b.** Actions	**a.** Output **b.** Consequences

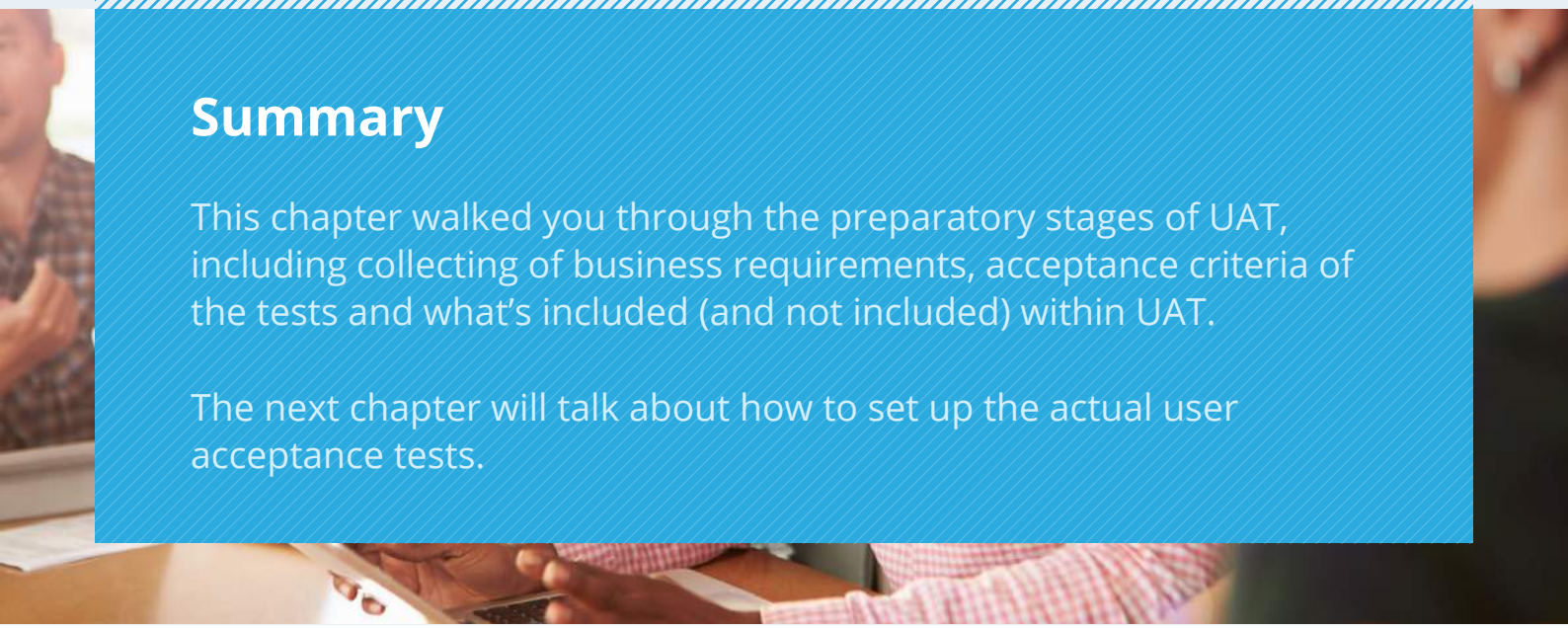
Scope Of UAT

Because UAT deals with user experience, it should ideally cover:

- **Operational Requirements:** Are the requirements around data capture, data processing, data distribution, and data archiving met?
- **Functional Requirements:** Are all business functions met as per expectations?
- **Interface Requirements:** Is data passing through the system as per business requirements?

UAT isn't about testing whether the radio buttons on a particular form function properly. These tests fall in the entry criteria of UAT, which also include:

- Completion of unit testing, integration testing and systems testing
- Absence of dealbreakers, high or medium level defects during integration testing
- Fixing of all major errors, except for cosmetic errors
- Defect free completion of regression testing
- Complete Requirements Traceability Matrix (RTM)
- Communication from systems testing team certifying that the system is ready for UAT.



Summary

This chapter walked you through the preparatory stages of UAT, including collecting of business requirements, acceptance criteria of the tests and what's included (and not included) within UAT.

The next chapter will talk about how to set up the actual user acceptance tests.

CHAPTER

03

The Nuts And Bolts Of UAT: Setting Up Your Tests

If you have read through the previous chapters, you will have an idea of the preparatory steps needed before you jump into the actual process of testing.

This chapter takes it forward, and will illustrate how to set up the actual tests.

The Different Types Of Tests For UAT

The following types of tests are included in UAT:

- **Contract Acceptance Tests:** These tests determine whether contractual obligations are met. They are conducted on systems acquired from vendors and third parties and are based on the requirements outlined in the original contract.

- **Regulation Acceptance Tests:** These tests are used to determine whether the system complies with regulations. They are especially important for software designed to be used in regulated environments like medical and financial industries, or by government departments.

- **Factory/site Acceptance Tests:** Many systems require on site installations after building. For these systems factory acceptance tests are needed before the installation meets its own contractual obligations. The importance of such tests are even more pronounced if the system is to be installed overseas.

- **Alpha/beta Testing:** Sometimes the exact requirements are difficult to define or are open ended. In such cases, developers often run alpha tests at their end and customers run beta tests on specific activities at the discretion of the users. Beta tests (also known as field tests) and the results of beta tests are fed back to the developers for fixes and improvements.

Each of these tests will fall under different processes inside the UAT workflow.

Fundamental UAT processes

If you want to complete UAT as efficiently as possible you will need to implement two key processes first: the FTP (Fundamental Test Process), which lays out the right sequence of activities done during testing, and the Test Development Process, which ensures that you design the right tests to get a clear idea about whether the system meets business requirements and acceptance criteria.

The Five Steps Of FTP Are:

1. Test planning, monitoring and control.
2. Test analysis and design.
3. Test implementation and execution.
4. Evaluation of exit criteria and reporting.
5. Test closure activities.

1. Test condition

A test condition highlights certain aspects of the business requirements (a function, transaction, feature etc.) in a form which enables you to create specific tests. A test condition can be either true or false.

For example, if you want to test the secure login functionality of a system you can create multiple test conditions, which all need to be true.

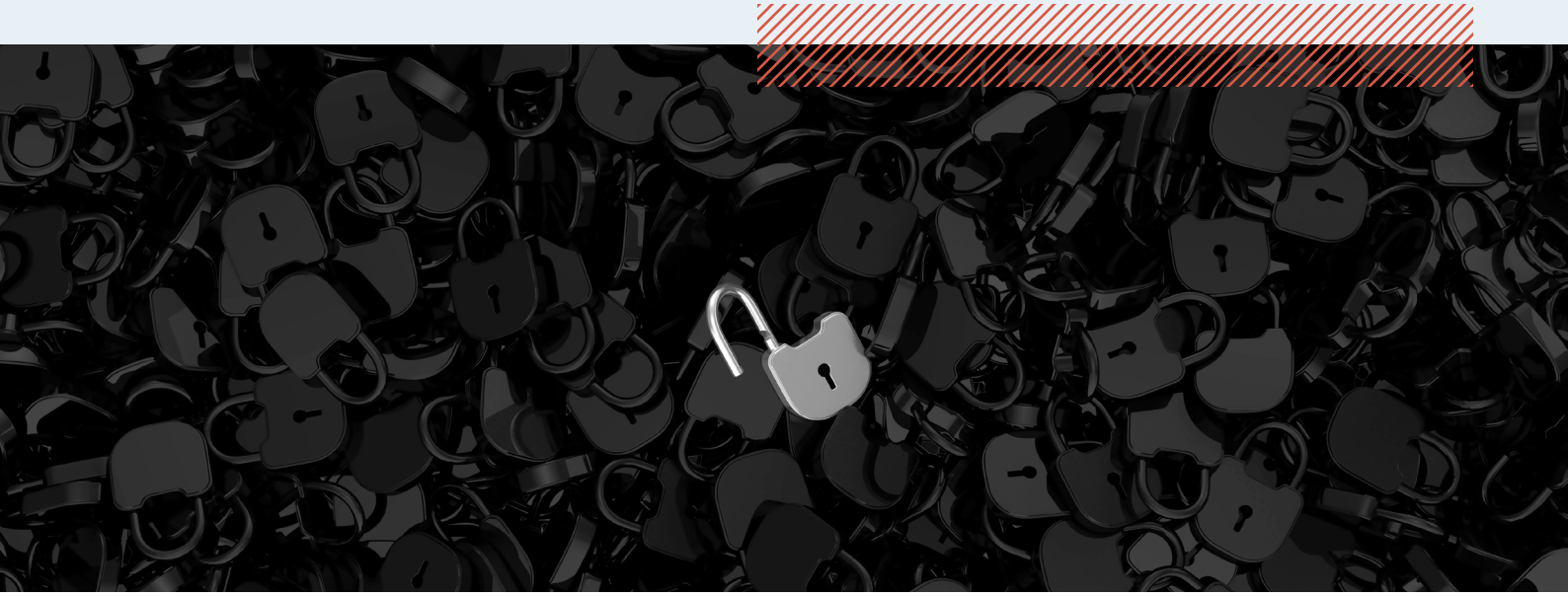
Test Conditions Table				
Valid Username	True	True	False	False
Correct Password	True	False	True	False
User Logged-in	True	False	False	False

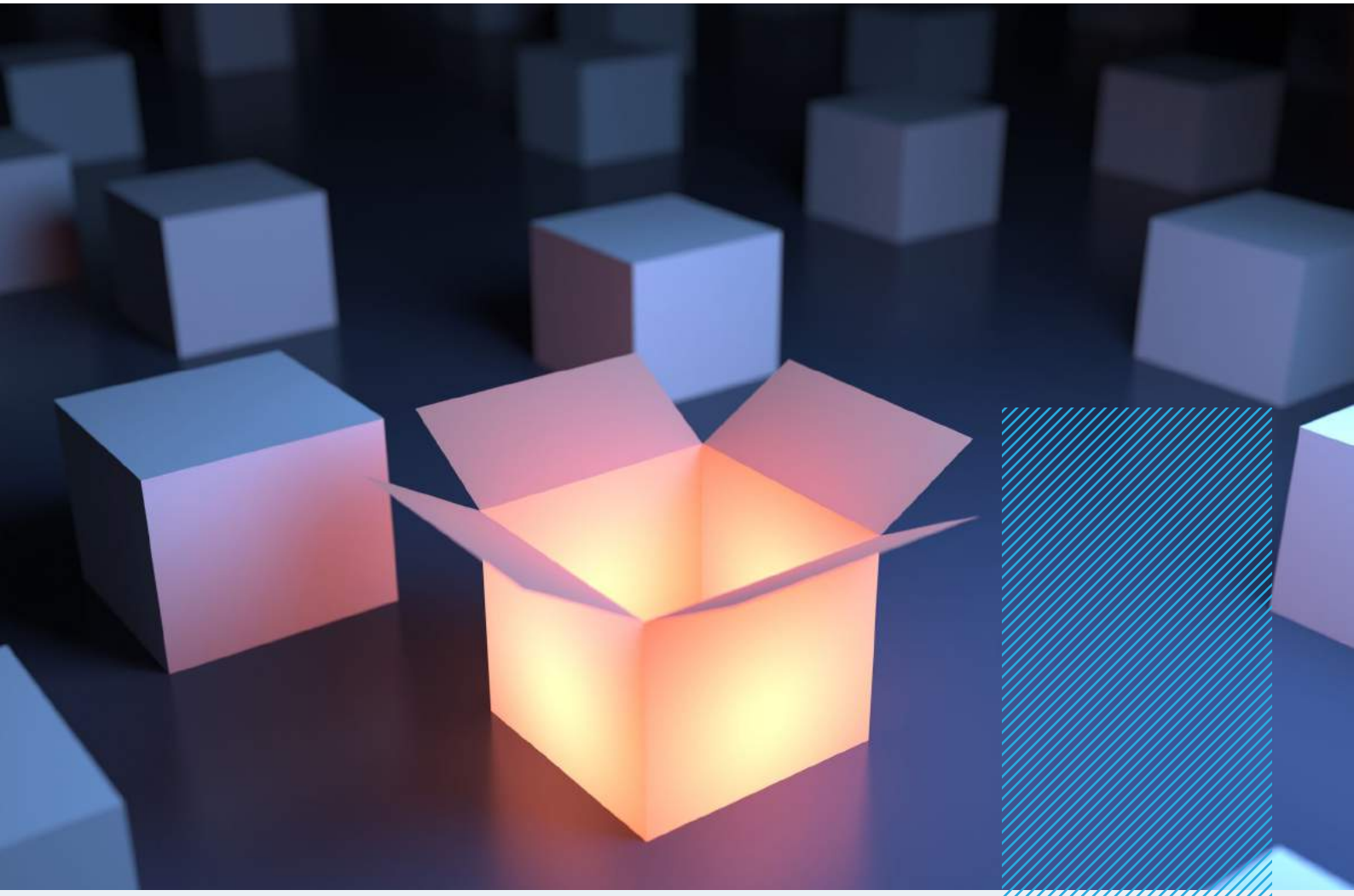
2. Test case

A test case is a set of inputs, preconditions or expected results developed for a test condition. Some of the test cases for our secure sign in process could be:

Test Case 1	Precondition	User not logged in
	Inputs	Valid username
		Valid password
	Outputs	None
	Post Conditions	User logged in
Test Case 2	**Precondition**	**User not logged in**
	Inputs	Valid username
		Non valid password
	Outputs	Error message
	Post Conditions	User not logged in

N.B. The preconditions and postconditions are needed for sequencing the tests so that they make sense.





3. Test Procedure Specification (A.K.A Test Script)

To execute the test cases which are generic templates with real data you will need test scripts.

Every test case will have multiple test scripts. Here's the test script for the Test case #1 from the above example:

Login 1: Normal User Login	
Purpose	User is able to login with an acceptable user ID and password
Preconditions	User not logged into the system. Test account set up successfully.
Test Data	User ID: tester@acme.com Password: UAT1
Process Steps	1. Click system icon 2. Enter user ID 3. Enter password 4. Click login
Results	User logged in to the system on the home page.
Post Test	User tester@acme.com is logged in.
Notes	

Different Approaches For UAT

Unlike other types of testing, which are based on testing outcomes against a specification, UAT is based on three elements which revolve around the end user:

1. Business requirements
2. Business processes
3. User expectations.

Because none of these three elements can be adequately documented as far as the user is concerned you need a specific approach to take into account the idiosyncrasies of UAT.

You may go with:

1. Requirements-Based Test Cases

These test cases cover the business requirements. They can either be written right after the RS document is prepared, or at the end of the project. However an error in the requirements will also cause an error in the test cases.

2. Business Process-Based Test Cases

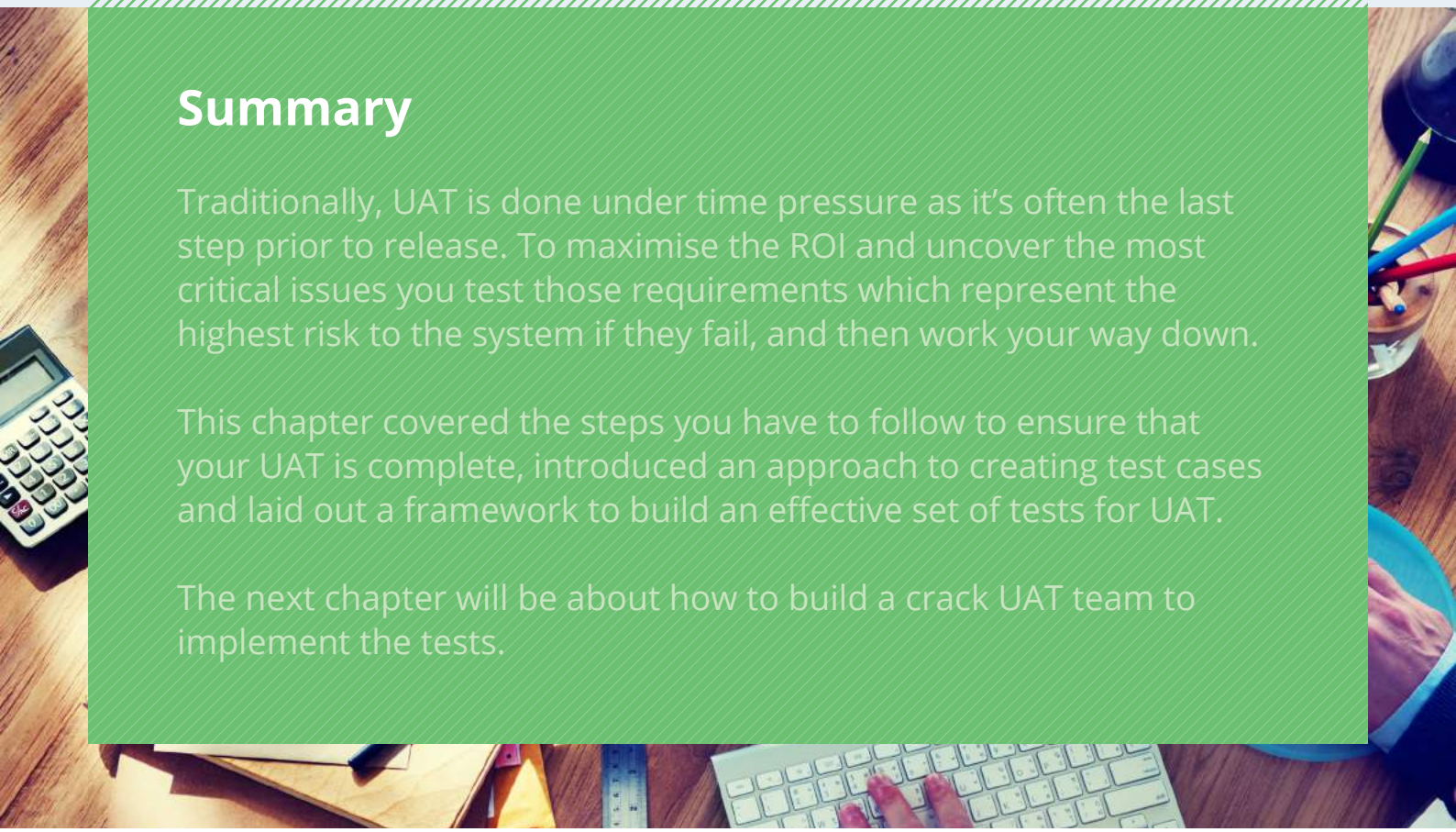
These test cases are written to make sure that the system will support the business processes. For business process-based testing, the tests must be sequenced so that they reflect how those processes work in real world environments.

3. User Interface-Driven Test Cases

User interface-driven test cases are based on data entry, interactions via the screen and reporting. In each case these will be related through a scenario so that data is manipulated in a realistic way. They can be run inside business process-based test cases where the business process involves data entry, user interaction or reporting. Some test cases include checking for:

1. Tab order.
2. Required fields.
3. Data-type errors.
4. Save and delete confirmations.
5. Shortcut functionality.

Keep in mind that the system might meet every technical specification and still fail the UAT process if it doesn't comply with existing business processes or is hard to use.



Summary

Traditionally, UAT is done under time pressure as it's often the last step prior to release. To maximise the ROI and uncover the most critical issues you test those requirements which represent the highest risk to the system if they fail, and then work your way down.

This chapter covered the steps you have to follow to ensure that your UAT is complete, introduced an approach to creating test cases and laid out a framework to build an effective set of tests for UAT.

The next chapter will be about how to build a crack UAT team to implement the tests.

CHAPTER
04

Building Your UAT Team

We have already broadly covered the concept of stakeholders in UAT, each with a different role to play in UAT, in Chapter 1.

Most organisations fail at UAT because they don't have the right team, primarily because UAT is so different from regular types of testing.

This chapter will take a deep dive into the process of building your testing team and address its relationship with other stakeholders.

Stakeholders In UAT

Depending on the scope of the project and the size of the organisation the stakeholders might include:

- Program manager, who is responsible for delivering a number of related projects.
- Project manager, who is responsible for delivering the project.
- Project management office (PMO) or administrative staff, who are responsible for organising UAT.
- Test/quality manager, who is responsible for all testing including UAT.
- UAT team leader/manager, who is responsible for delivering UAT.
- UAT trainer, who is responsible for delivering UAT training.
- Business analysts, who are responsible for documenting requirements.

Important Roles In A UAT Team

The UAT team's job is to plan and execute testing and provide the stakeholders with enough data so that they can decide whether or not to accept the system in the current state.

The team usually consists of a team leader or manager, business analysts, and the UA testers. Larger teams can have additional specialist roles

Here's a rundown of the key roles in UAT:

Business Analysts (In-house or Outsourced)

Business analysts can talk the language of both IT and business.

Their job is centered around interpreting business requirements into functional specifications. They will also help in making sure that test cases and test scripts match the end-user experience.

Business analysts are also involved in test execution and reporting, and can help rate severity of incidents, discount any duplicate incidents, or explain and resolve issues during testing.

UAT Co-ordinator/Manager/Team Leader (In-house or Outsourced)

The UAT coordinator will create a plan for UAT and organise/ plan resources for testing.

They will ensure that the test environment replicates the real world system as closely as possible.

They will also manage and track test incidents and, along with the business analyst, recommend to what extent the system requires changes or if business processes can be adapted.

UA Testers (In-house/ Outsourced)

The UAT testers could be both end-users or subject-matter experts with knowledge of the current system or processes. Ideally, the UA testers should be consulted when business needs and requirements are defined at the start of the project.

They will determine how appropriate a test case is. They will also execute test scripts, note incidents and provide feedback on the UX.

The Essential Skillsets Required For Successful Team Members

The UAT team needs to be able to operate independently and autonomously, with some specialist support, so that the content and schedule of the tests aren't biased in favor of a particular stakeholder.

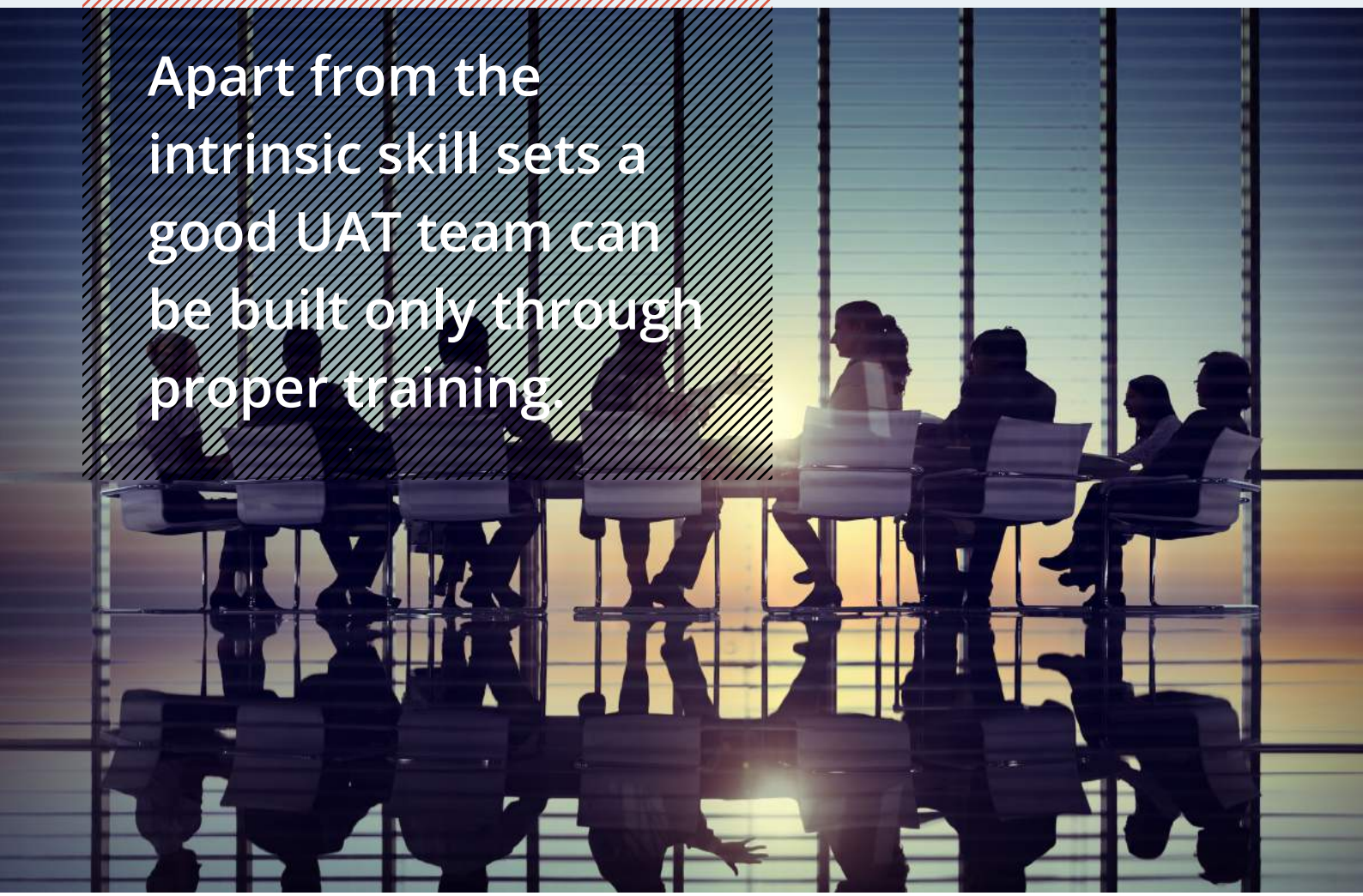
The following skills are mandatory for a well-oiled UAT team.

Business Analyst	Team Leader	Tester
In-depth knowledge of the business requirements	Thorough knowledge of UAT	IT literacy
Real-world working product knowledge and how that applies to the system, system architecture knowledge, including current fixes and workarounds	An understanding of formal testing methodology	Recognition as an expert ability to represent the wider user group
	Writing a UAT plan, assigning resources and deciding which issues to escalate	Detail-oriented and diligent in collecting proper documentation to support the test results
		Creative enough to improve existing processes.

Depending on the size and complexity of the UAT project, other desired skills are:

- Test automation experience.
- Familiarity with requirements management, version control, IM and test planning tools.
- Familiarity with general software tools like spreadsheets.

Apart from the intrinsic skill sets a good UAT team can be built only through proper training.



How To Train Your UAT Team

Training is essential for a team to run the UAT process efficiently so that the investment in setting up the UAT environment isn't wasted.

In many cases UAT training is when the team will experience the new system and meet the other stakeholders.

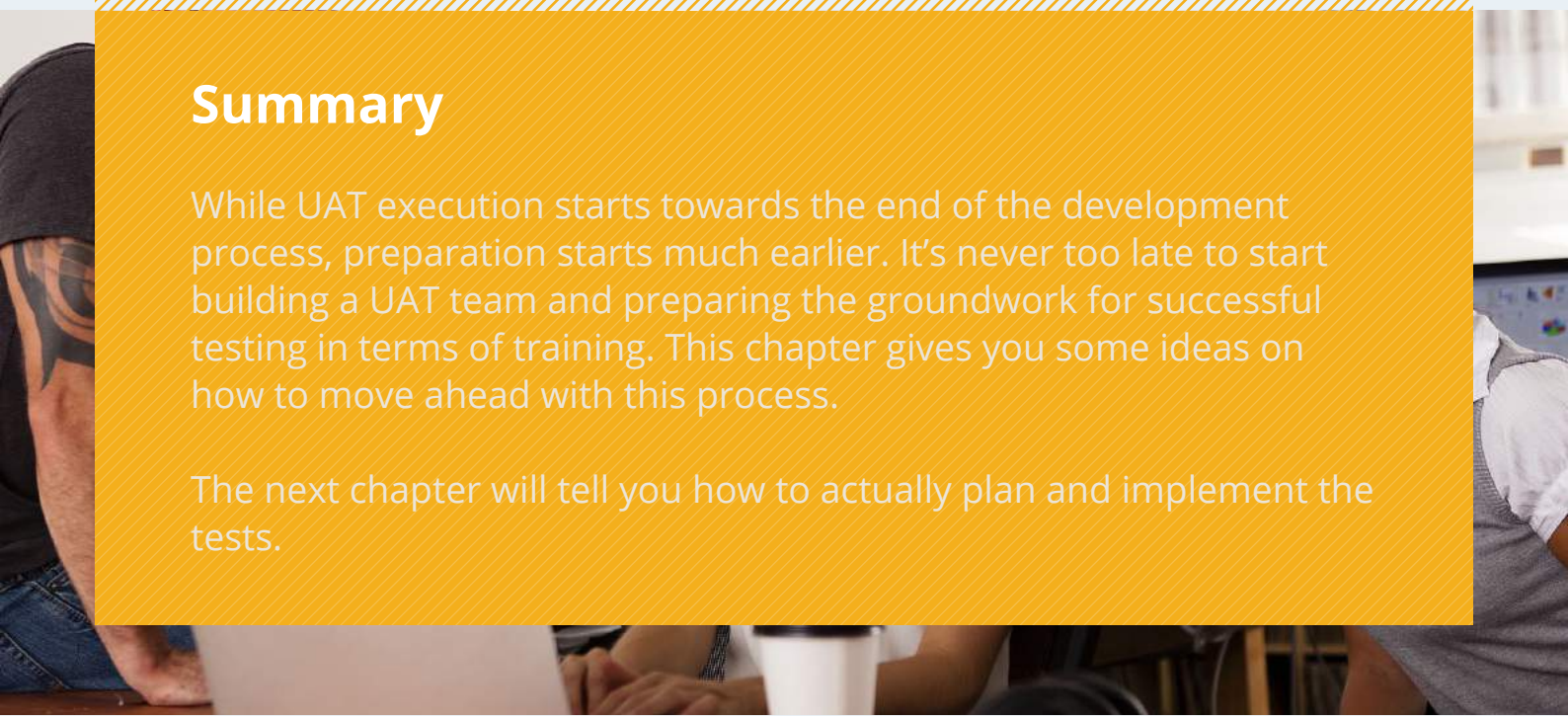
You should consider these questions when you are designing a UAT training program:

- Is the timing such that the participants can immediately implement the learnings?
- Does the training teach the skills necessary for a single user to be successful?
- Is there an opportunity to learn from other users or roles?
- Is the system stable enough to support training?

Depending on the different roles in the team the UAT training content should at least cover the following topics, apart from giving trainees a basic introduction to UAT:

- Information and timeline about the project. (Project manager)
- The basic functionalities of the system. (Project manager)
- Known issues and workarounds. (Project manager)
- Expected business benefits. (Manager)
- Details about previous tests. (Project team)

Testers should be thoroughly trained on the key steps involved in executing a test script, evaluating and logging of results, and reporting test incidents.



Summary

While UAT execution starts towards the end of the development process, preparation starts much earlier. It's never too late to start building a UAT team and preparing the groundwork for successful testing in terms of training. This chapter gives you some ideas on how to move ahead with this process.

The next chapter will tell you how to actually plan and implement the tests.

CHAPTER

05

How To Plan And Execute The Tests

Up until now we have built a number of deliverables from the development process including:

- A set of business requirements in the form of a requirement specification document.
- Functional testing done by the development team and perhaps also by some independent testers.
- A complete UAT ready system.
- A trained UAT team.

But these details aren't enough to begin the planning process. You also have to know when to stop testing and release the system into the wild, and this decision will be informed by the acceptance criteria.

Agreeing On Acceptance Criteria

The ideal acceptance criteria is a system which works correctly, has zero defects and is ready for release on the planned release date.

But that almost always never happens in the real world.

That is why we need to set realistic acceptance criteria well before the UAT process begins.

To determine a realistic acceptance criteria here are some questions to consider:

- What will happen if the system is not released on the planned date?
- How will business be affected?
- How much delay could be tolerated?
- How quickly would the costs and problems ramp up with a delayed release date?
- What's the acceptable mix between critical defects (which prevent the system from delivering its core capabilities), serious defects (defects that will affect performance significantly), and routine defects (defects that are routine in nature)?
- Which functionality is essential for the system to achieve its business benefits?

These questions will help all stakeholders think about the tradeoffs and the compromises to be made before the system is deemed release-worthy.

Your product roadmap will change based on the core acceptance criteria. If you're focused on the delivery date you might have to release the product by fixing critical bugs while pushing certain features to be completed at a later date.

Conversely if you decide that the acceptance criteria is zero defects, you will have to push the release date back.

Setting Up Entry Criteria And Test Management Controls

Along with acceptance criteria, it's also important to establish entry criteria so that the system doesn't change while the UAT process is underway. Not doing so will create nightmarish issues with change control and waste precious time and resources.

Entry criteria for UAT include:

- All testing up to system testing is completed without outstanding incidents
- No critical or serious defects
- No unresolved issues affecting testing
- Any medium issues remaining have an acceptable workaround
- A number of low severity issues (defined at a certain amount)

Another important step in test planning is test management control which includes:

- Management of defects already identified prior to UAT and the defects uncovered during the process
- Test logging which will tell you at any time the number of tests completed and tests still left to do
- Change control to ensure that all changes to the system or the tests are identified and followed up
- Incident management in the event of failures unearthed during tests (and subsequently defect management) where the incident is due to a defect in code or documentation.

Once this groundwork is completed you can now proceed to the actual job of creating the tests.

We start by identifying test conditions.

Identifying Test Conditions

We have already covered test conditions in Chapter 3.

Each test condition represents one component of a feature that can be assessed as either true or false, and the feature can be considered as correctly implemented if all the conditions are true.

Creating the test conditions is a crucial stage in the test design process, especially when it comes to complex system functionality.

In such scenarios it's helpful to create a test condition matrix which might look like this:

Spec/Design Ref	Ref I.D	Req Name	Condition 1	Condition 2	Condition 3

You can populate this table by cross referencing the business requirements and working with your team to come up with different conditions. This matrix can be extended both horizontally and vertically based on the complexity of the system, and makes it easy for all stakeholders to understand the test conditions and sign off on the process.

You also need to run risk analysis to prioritise the test conditions in the event that there are too many of them.

Creating A Test Set

You will now need to schedule all the tests to achieve the test strategy and assess the system against the acceptance criteria while maintaining control over the testing process.

The test schedule:

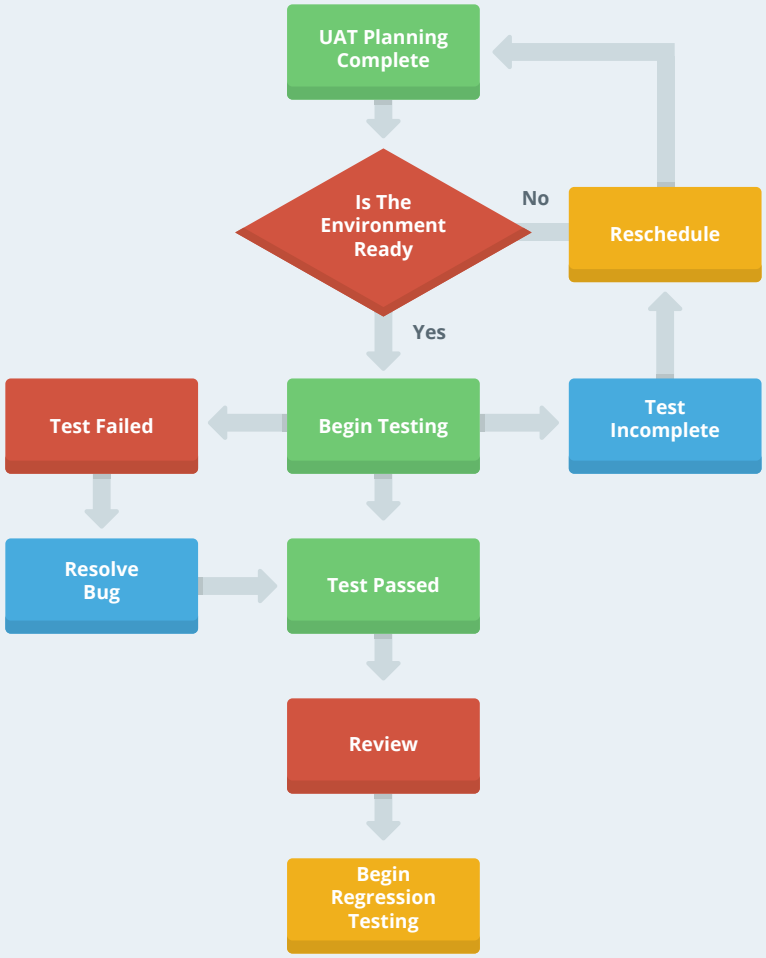
- Ensures that all the tests outlined in the test strategy are included.
- Sequences the tests for the most efficient use of time and resources and in accordance with business processes.
- Allocates testing resources to different activities.
- Enables you to log testing and keep track of progress.
- Lets you reschedule in case of problems.

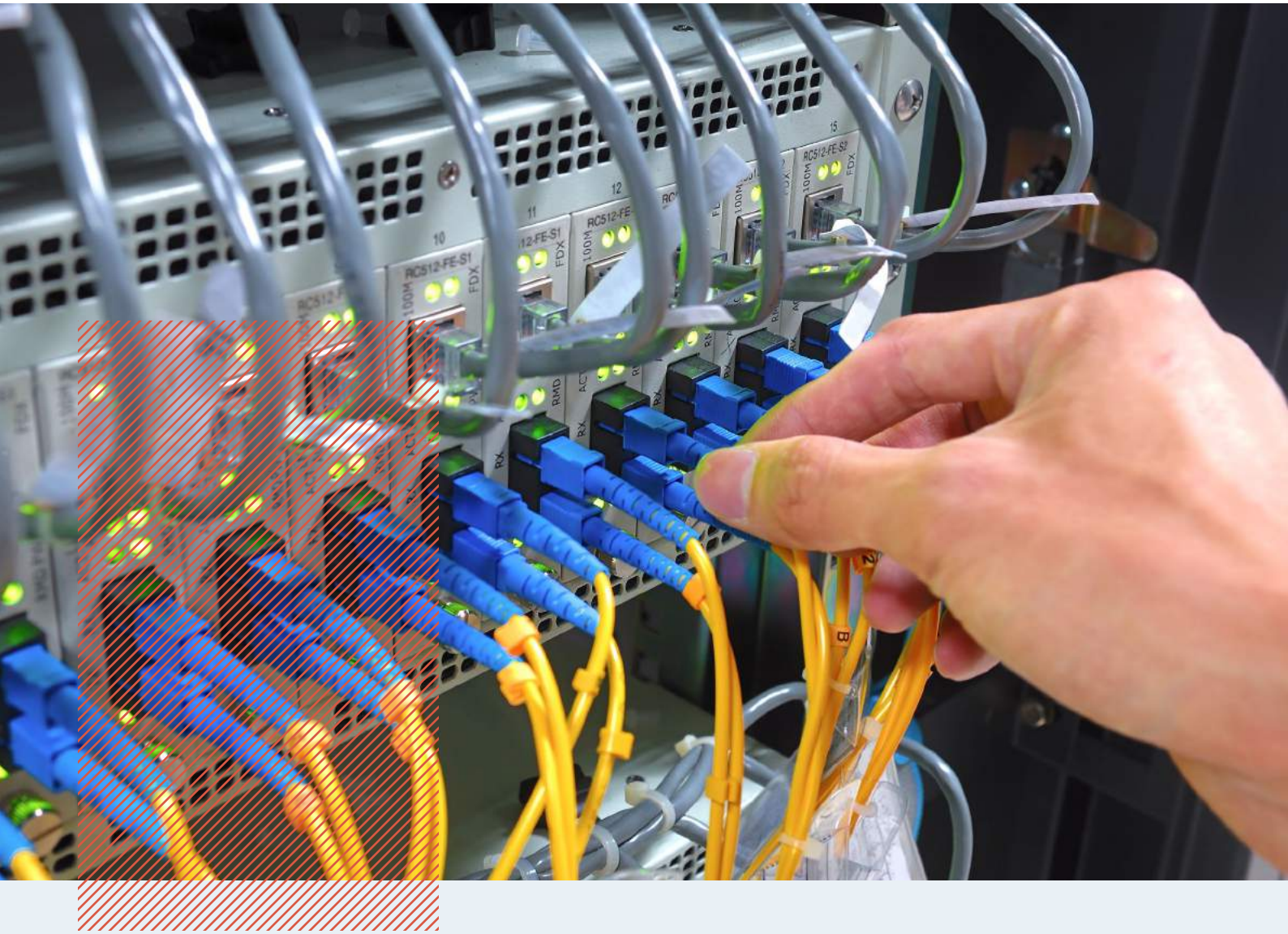
Your test schedule will depend on your UAT strategy. For a risk based strategy, for example, the tests with high level risks will be run first.

Test schedules will have to take into account a number of factors, like:

- Priority of tests;
- Availability of test environment;
- Availability of testers

The UAT testing lifecycle can be depicted using this block diagram.





The test schedule can help you save time by streamlining the tests based on whether the preconditions and postconditions of different tests match up with one another or on the basis of how the different modules deal with data input and error handling.

This table can be used to fill out a detailed testing schedule.

Sequence #				
Requirement #				
Module				
Test Case #				
Test Case Description				
Input				
Expected Output				
Test Script ID				
Tester				
Process				
Date Of Completion				

Determining Progress Of Testing And Reporting

You will need to assign activities from the detailed test schedule to individual testers and ensure that they have the necessary test scripts and test environments in place. Testers then set up and run their tests according to the test script.

Tests may be allocated on a 'first come, first served' basis where any available tester takes on the next scheduled test script, or test scripts may be annotated for execution by testers from a particular speciality.

All testing activity is entered into the test log which starts out as the copy of the test schedule.

The test log will have to be continuously updated, and will have the records of the following:

- Incidents, defects, and details of retesting.
- Tests rescheduled due to delay or lack of resources.
- Tests rescheduled around retesting.
- Regression tests.
- Feedback.

Issue #				
Test-script ID				
Tester				
Step #				
Date Raised				
Module Tested				
Test Case Description				
Defect Desccription				
Defect Severity (1-3)				
Repeated				
In-scope (Y/N)				
Assigned To				
Issue Resolution Status				

In the event that a test doesn't give the expected output you will need to raise an incident report. The format of the report can be something like this:

Acme Systems UAT Incident #

Incident Report

Incident Date/Time

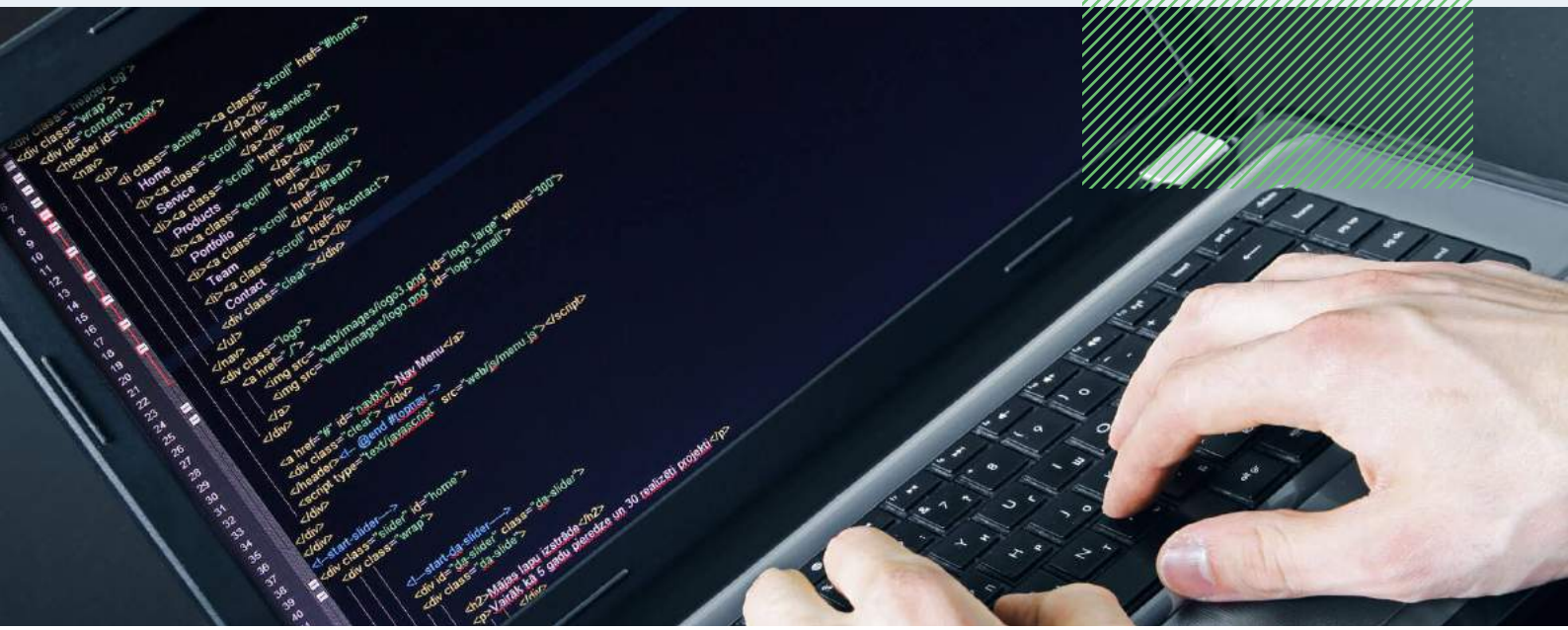
Tester: xxxxxxxx Contact Details; xxxxxxxxx

Test script ID: xxxxx

Incident Severity: Highly

Repeatable? : Yes

Incident Desc: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx



The test logs and the incident reports will be passed over to the UAT team leader for evaluation, and from there, to the development team.

The test schedule and the test logs will tell show you the rate of progress of the UAT.

Another way to measure test progress is by benchmarking against acceptance criteria.

Creating A Status Report

The UAT status report is a summary of all the progress information, estimates of when UAT will be completed, and recommendations with data to support them.

The status report will be needed for evaluating the results of UAT and can look like this:

Acme UAT Summary Report

Date:
Overview: (Outline of tests performed since last report)

Summary Assessment:

- Tests Planned
- Tests Run
- Tests Passed
- Tests failed
- Tests blocked
- Incidents reported
- Incidents resolved

Progress To Date

- Tests run/total tests
- % Tests passed
- %Tests failed
- % coverage achieved
- Tests still to be run
- Incidents resolved

Status Against Plan

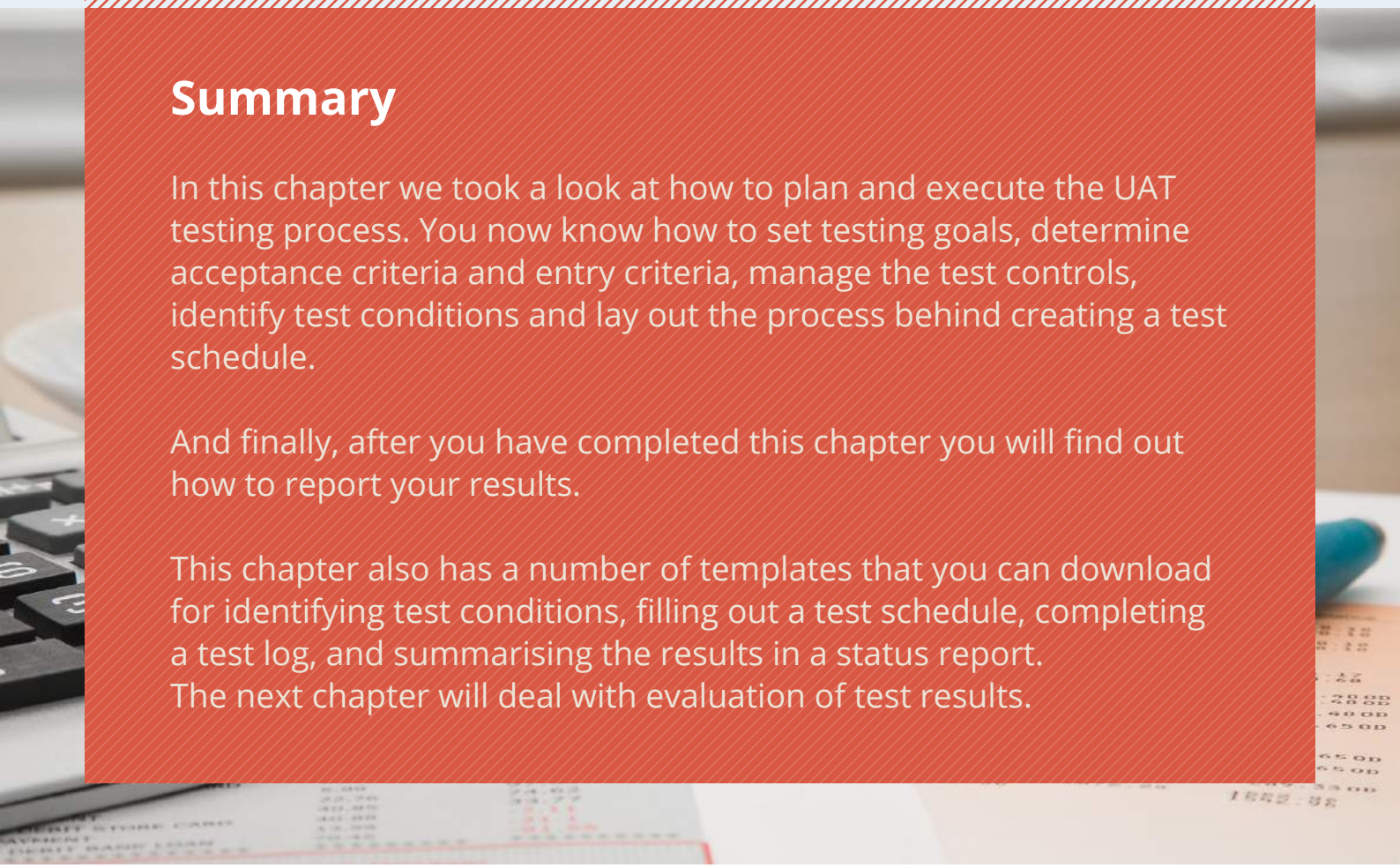
- % tests complete
- Schedule Slippage
- Schedule status
- Tests outstanding
- Blocked tests
- Regression tests outstanding

Status Against Acceptance Criteria

- % tests complete
- %requirements coverage achieved
- Defects outstanding
- Critical
- Serious
- Routine

Recommendations

Signature **Date**



Summary

In this chapter we took a look at how to plan and execute the UAT testing process. You now know how to set testing goals, determine acceptance criteria and entry criteria, manage the test controls, identify test conditions and lay out the process behind creating a test schedule.

And finally, after you have completed this chapter you will find out how to report your results.

This chapter also has a number of templates that you can download for identifying test conditions, filling out a test schedule, completing a test log, and summarising the results in a status report.
The next chapter will deal with evaluation of test results.

CHAPTER
06

Evaluating The Test Results

The decision of when to stop user testing and accept the system depends on how the system was built or acquired, who the stakeholders are, and their needs.

When To Stop Testing

If you have followed the processes laid out in the previous chapters you will have a clear idea of the acceptance criteria, a test plan, and an implementation roadmap.

Ideally, testing stops when exit criteria has been met.

In the real world you might face unforeseen situations like:

- The business requirements no longer support UAT.
- The system release is time critical.
- The testing is finding few, if any, problems.

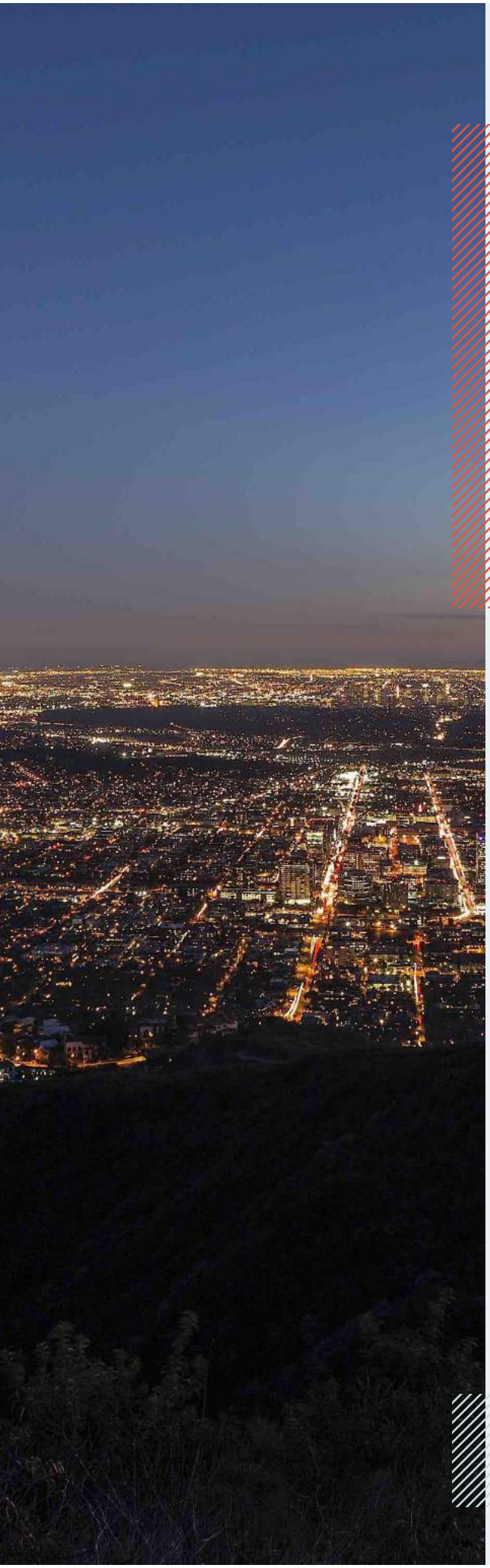
That's why routine reporting on the testing is necessary. As part of that reporting, you will need to maintain regular progress updates towards the acceptance criteria so that you can have a clear idea of where you are in relation to the release decision at every stage in testing.

If you have to end testing prematurely you will have to evaluate the risk of release and the business value of the system so that you can be prepared for any exigency.

To help you with this evaluation you have to consider three factors which comprise the emergency release criteria:

- Stability of the system
- Usability of the system
- Coverage of the testing.

An individual assessment will ensure that you are not caught unawares when the system is released.



There are three checkboxes to tick for comprehensively accepting UAT results:

1. Contract Acceptance Completed

If the system was outsourced or acquired from third parties there are certain criteria associated with system acceptance.

You will have to evaluate the criteria which relates to testing and report the results to the relevant stakeholders with a recommendation on whether to accept or not to accept the results.

The process moves on to step 2 if third parties aren't involved.

2. Acceptance Criteria Met

You can evaluate the system based on whether acceptance criteria has been met and make appropriate recommendations.

You can document this in a UAT completion report that describes the testing done for UAT and the results of that testing in the context of acceptance criteria.

3. Risk Of Release Assessed

In case acceptance criteria are not met, you should assess the risk of releasing the system in its current state.

Creating The UAT Completion Report

The UAT completion report is generated once the tests have stopped. The format of the report is something like this:

UAT Completion Report Outline

Introduction
Purpose of the document
Identification of the system under test
Scope of UAT

Overview
Outline of test processes, testing activities, environment, time taken, participants etc.

Acceptance criteria
Identify individual criteria and acceptance levels.

Constraints
Any constraints on testing related to environments, availability of resources etc.

Test Results Summary
Summary of all the tests planned, run and failed.

Test Incident Summary
Summary of test incidents raised and resolved by severity, including unresolved test incidents.

Acceptance Criteria Evaluation
Details of evaluation for each criterion.

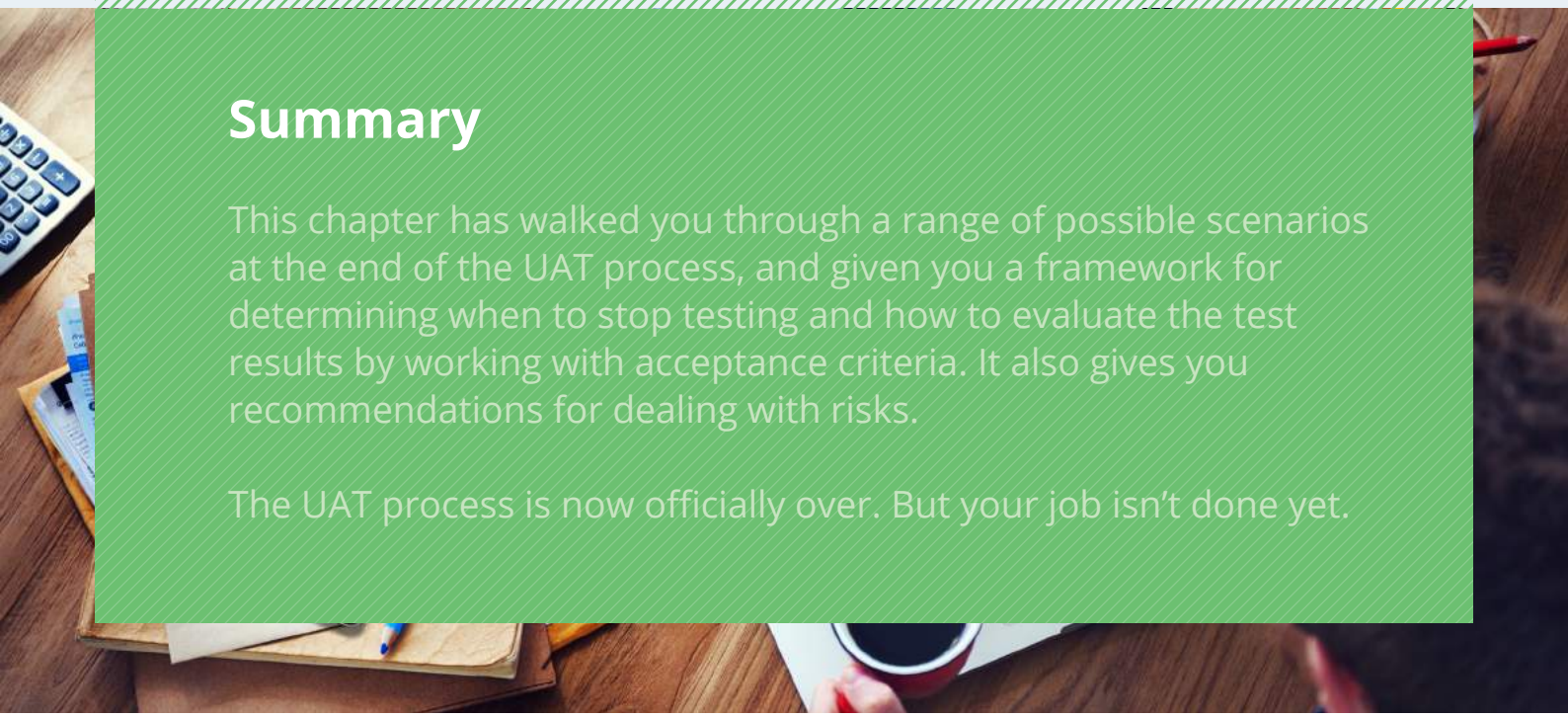
Overall Assessment
Overall assessment of all acceptance criteria against required levels.

Recommendations
Recommendations related to release or risk reduction.

Appendices
Detailed Test Results
Test Incident Reports

We can identify a range of possible recommendations at this point:

- Release the system as-is.
- Push release date until important risk reduction measures are in place.
- Release the system with additional support to deal with the risk of early problems.
- For critical issues, push release date and plan for risk reduction with additional support.
- Reject the system.



Summary

This chapter has walked you through a range of possible scenarios at the end of the UAT process, and given you a framework for determining when to stop testing and how to evaluate the test results by working with acceptance criteria. It also gives you recommendations for dealing with risks.

The UAT process is now officially over. But your job isn't done yet.

CHAPTER

07

After UAT, Now What?

UAT is usually an activity that is completed against a backdrop of pressure, and completion is often a relief to all concerned. Once you have completed the test evaluation process, you should analyse the outcomes.

This analysis will give you the opportunity to reflect on the learnings from the UAT process and help you think about the activities which will guarantee successful system rollout.

Here's the format of a post UAT Analysis Report:

Post UAT Analysis Report

Introduction: (Purpose of system, implementation dates)

Purpose of the report: (Analysis of UAT, recommendations for implementation)

Test Results Summary:

- Detailed Test Results
- Coverage Analysis

Incidents and Defects Summary:

- Defects Analysis
- Unresolved Incidents
- Workarounds Identified

Support Summary:

- Technical Support
- Business Support (help)

Recommendations:

- End User Training

User Guides

- Help System and FAQs
- Technical Support Issues
- Implementation Planning
- Risk Assessment
- Business Evaluation

Signature **Date**

How To Prepare A Roll Out Strategy

Depending on the size and geographical spread of the organisation, there might be a range of possible strategies, from putting the system on every desktop at once) to a series of pilot releases.

You should consider these points for system rollout:

- If there are high defect rates in UAT you might want to go with a smaller initial pilot
- In case of user interface problems, you can launch the pilot project with support until kinks are ironed out.
- You can test workarounds, help guides etc. in an initial pilot.
- If there are fewer UAT problems than anticipated you can roll out the system ahead of deadline.

Implementation And Post-Implementation Defect Correction

Depending on feedback from UAT, the required levels of technical support and business support (help desk), can be estimated during implementation of the system.

Post implementation, some defects will emerge from the increased level of usage. Some of these defects will need to be corrected urgently, while others will be placed on the prioritised list of changes to be made over time.

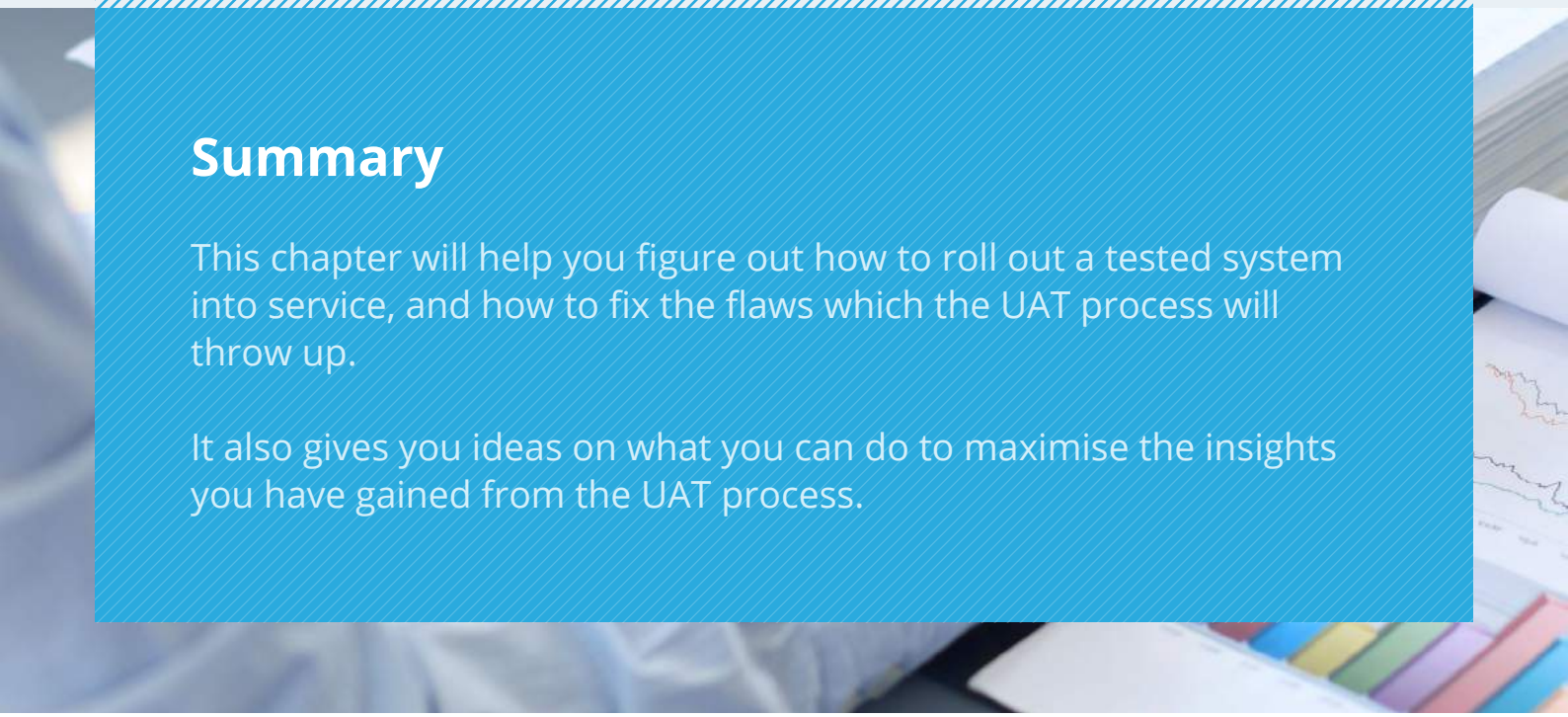
In the early post-implementation period defect correction might require a new release of the system at a smaller scale. You might have to run a mini-UAT as part of your risk reduction strategy.

Measuring Business Benefits

You can start measuring desired business benefits when the system is completely rolled out across the organisation by analysing the data to identify expected changes.

This process will take some time as you will need to account for the common factors present before and after the UAT exercise.

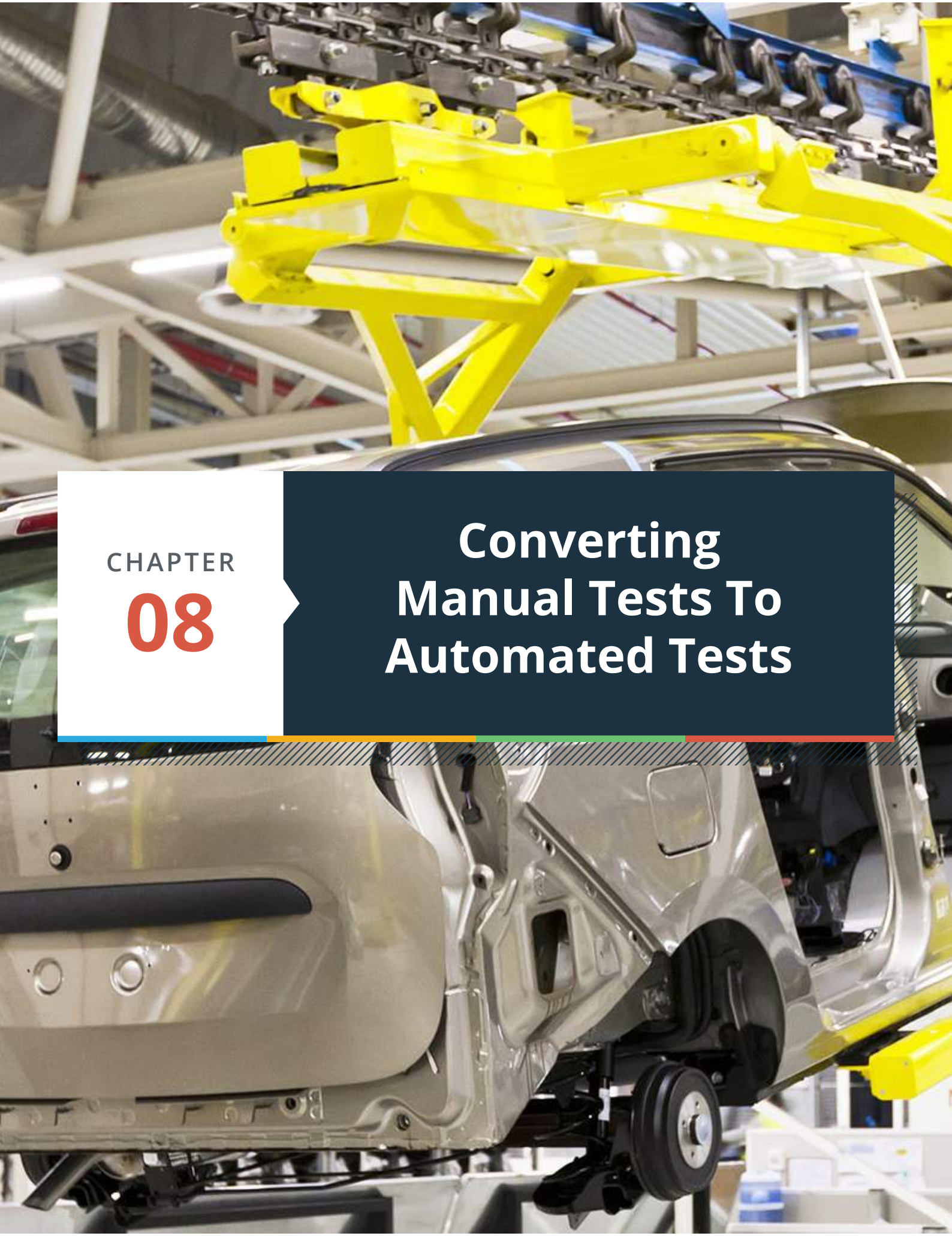
Because the UAT team has extensive experience with the new system they can help measure the business benefits by running the experimental data through the system before the changes are released and then running the same data after the system is rolled out.



Summary

This chapter will help you figure out how to roll out a tested system into service, and how to fix the flaws which the UAT process will throw up.

It also gives you ideas on what you can do to maximise the insights you have gained from the UAT process.



CHAPTER

08

Converting Manual Tests To Automated Tests

Cost and time factors are one of the major reasons why organisations don't test as extensively as they should.

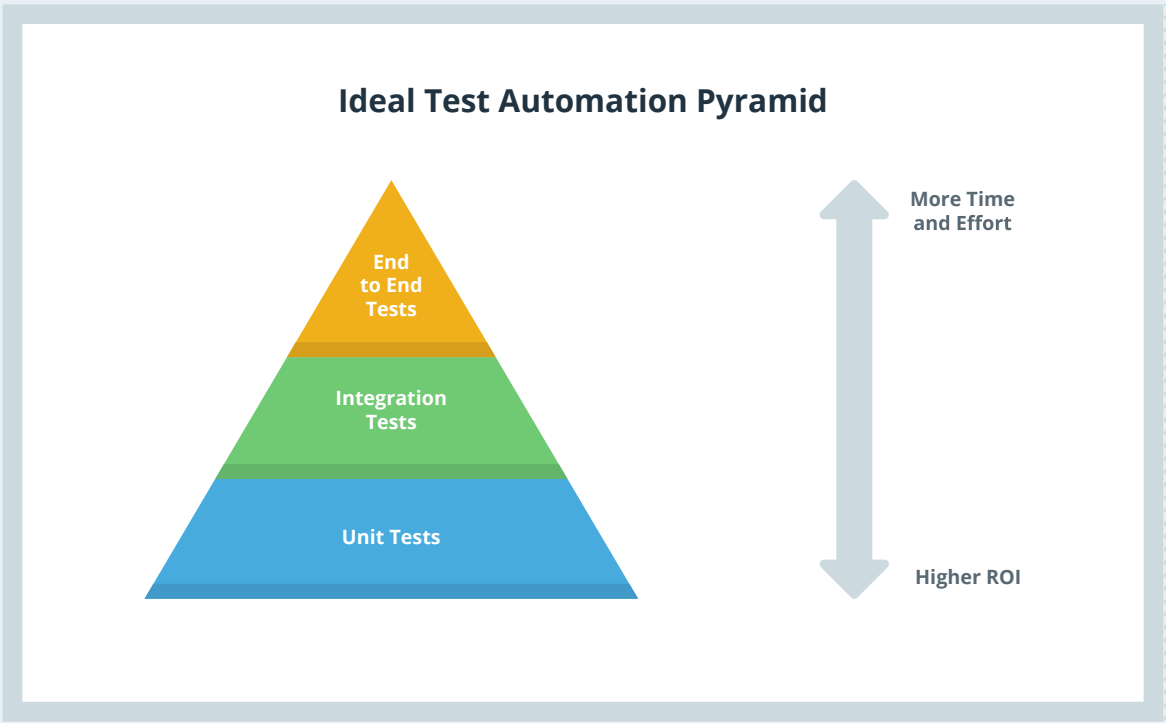
Most of that time is taken up by manual testing, run without tools or scripts, mostly through the user interface.

This strategy is resource intensive and works well in the following scenarios:

- **Exploratory Testing:** In the absence of written specifications, manual testing can be used to test the limits of the system and is based on the tester's knowledge, skills and experience.
- **Usability Testing:** Manual testing is the only way to find out the user experience on a platform or a system.
- **Ad-Hoc Testing:** In this scenario, manual testing can be used to find out the normal capabilities of a system. It's not a replacement for rigorous testing.

For other types of testing like regression, load or performance testing you CAN automate the process and get 5-10x more tests done at the same time, subject to certain caveats:

- The asset you are planning to run tests on is critical to your business
- You understand that setting up automated tests is equivalent to any software development project
- You have manually run the tests you wish to automate (sounds oxymoronic, but more on this point later)
- You have a handle on the various costs associated with test automation, from setting up the test environment to running and maintaining the tests
- You follow the Ideal Test Automation Pyramid by running multiple unit tests for every end to end test to maximise ROI.



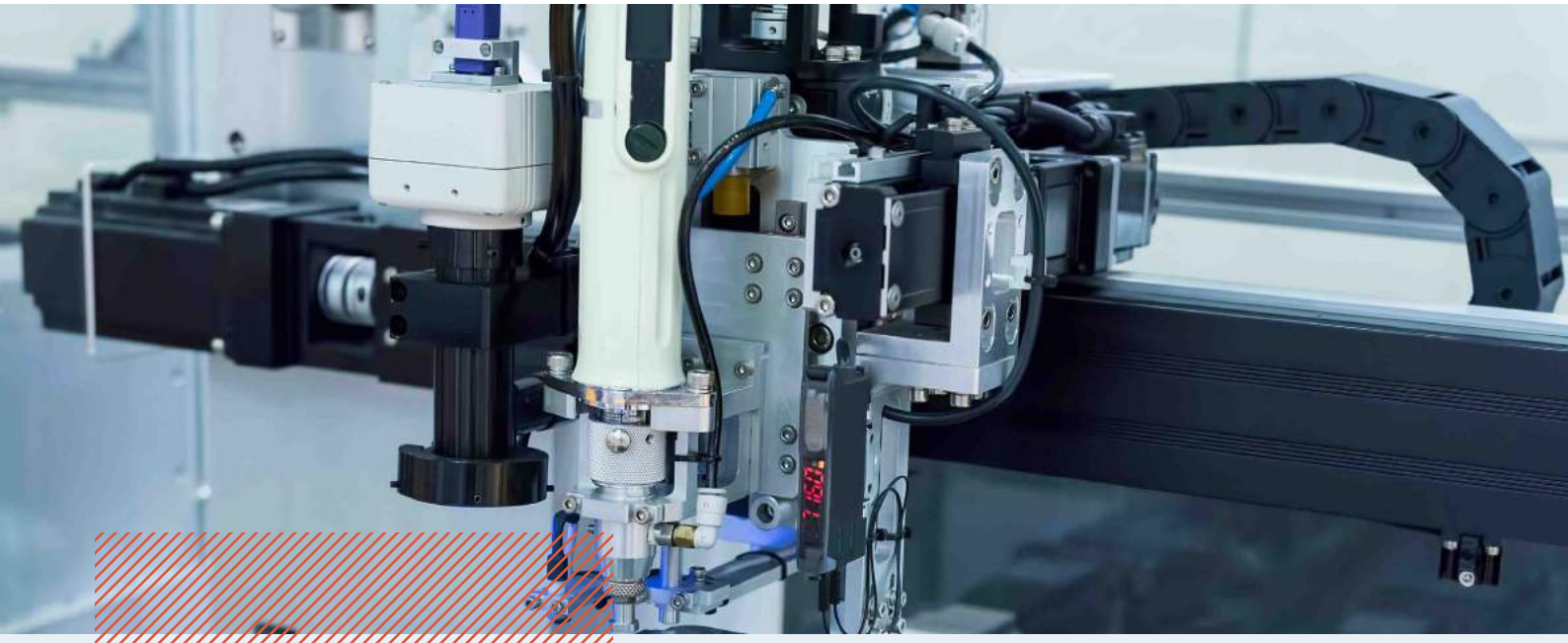
Once your expectations are set, here's how you can calculate the ROI of automated testing

If a tester on average costs $50 an hour and if a senior tester who creates automated tests costs $75 an hour, that would cost about $400 and $600 respectively per day per tester.

Now, consider a team of 10 testers, five senior-level and five entry-level, with a monthly loaded cost of $105,000 (for 168 hours per month). You would get a total of 1,350 hours costing $78.00/ hour (this is assuming each tester realistically works 135 hours per month due to breaks, training days, vacations, etc.). If you automate testing, the cost of labor would remain the same, but for the effort of 3 test automation engineers, you would achieve 16 hours a day of testing and will run 5x more tests per hour.

This results in the equivalent of 5,040 hours per month of manual testing created by the three test automation engineers. Then, consider the rest of the team doing manual testing (7 people x 135 hours/month). That amounts to 945 hours more, ending with a combined total of 5,985 hours of testing at $17.54/hour ($105,000 divided by 5,985 hours)."

Source: Abstracta http://www.abstracta.us/2015/08/31/the-true-roi-of-test-automation/



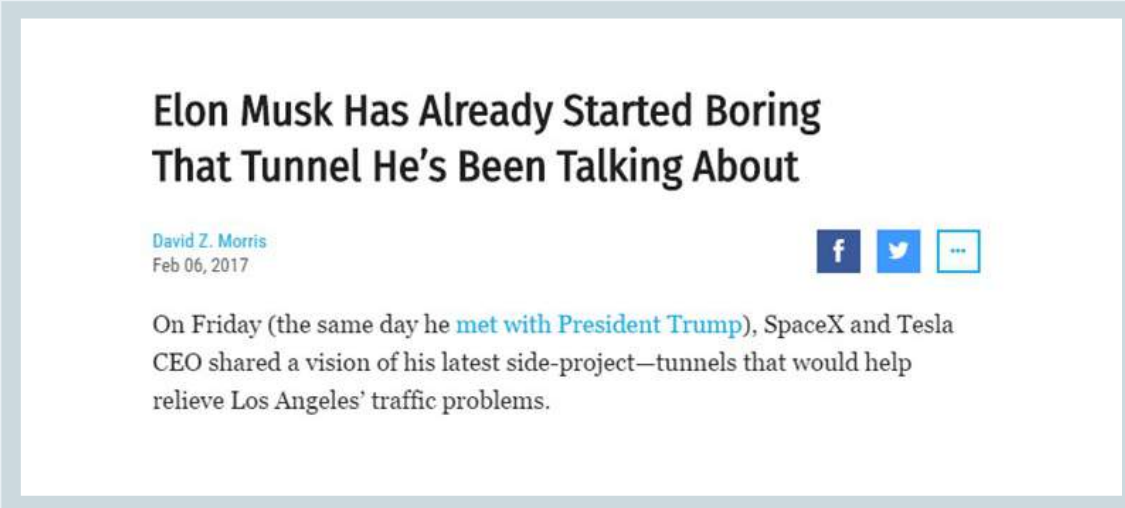
But most people don't achieve these returns and end up mothballing their automated testing project.

The Right Way To Automate Tests

That's because they're approaching test automation the wrong way. They assume that setting up an expensive test environment and writing a bunch of test scripts is all there is to the process.

That's like Elon Musk boring tunnels under every road in Los Angeles to solve its apocalyptic traffic problem.

Instead, he would start by surveying traffic data to find out the busiest routes, and figure out how to connect these areas without disrupting existing utility lines or risking the foundations of heritage buildings.



Before automation, start with manual tests so that you can understand the capabilities and limitations of the system. After you have run through the first iteration of manual tests you will get a feel for different test cases and the workflows associated with these tests.

If you can convert the workflow to a given-when-then or arrange-act-assert you can convert these into test scripts, load them up in your test tool, and press enter.

Here are a few points to keep in mind as you automate your tests:

- Understand what feature you are trying to test. As noted in the previous section, automated testing isn't going to work in the context of, say, UX.
- Document how you can automate certain tests, in plain English or in pseudocode.
- Share that document with your team and other stakeholders to get their input.
- If an automated test fails don't take it at face value. It might be a false positive because you might not have understood the requirements well.
- Instead of tests which just tell you whether a system is working or not, write tests which can isolate a specific issue.
- Because writing tests is writing software, keep an eye on the costs. You will have to allocate resources to maintaining the test environment.
- Manual testing should not be put off till the very end, but should be regularly run during the development process.

Summary

Incorporating automation to your testing process can result in more reliable systems.

But automation isn't a replacement for manual testing: it's simply a way of speeding up a subset of manual test processes. However, don't automate if your only goal is to save money.

Accelerate Test Cycles & Bullet-Proof Software Releases with Gamified User Acceptance Testing

Bugwolf lets you transform software testing into competitive UAT challenges that accelerate digital releases, lower customer support calls and reduce defect costs.

Watch on as professional testers race against the clock to dramatically improve your app or website. Six hours is all it takes to conduct deep functional, usability, user experience, or user acceptance testing for key user journeys.

During your Bugwolf challenge, you'll receive severity-ranked video reports that make it quick and easy to replicate and resolve bugs.

It's not uncommon for Bugwolf to uncover one hundred or more bugs in a single six hour challenge. Our process and experience helps you to condense test cycles from weeks into days, achieve a +500% return on investment, and significantly reduce costs.

"Every digital leader has a responsibility to ensure their organisation doesn't become the next software testing disaster story. We still encourage people to optimise their in-house teams, however, it's time to go a few steps further. That's where Bugwolf comes in." Ash Conway, CEO & Founder of Bugwolf.

Bugwolf Is The World's Only Gamified UAT Solution

We Help Accelerate Test Cycles, Bullet-Proof Software Releases And Prevent Catastrophic Software Event

The best way to find out more is to schedule a short, 15-minute demo with the Bugwolf team:

REQUEST A 15-MINUTE DEMO

Bugwolf

www.bugwolf.com

Australia	+61 3 9001 0270
New Zealand	+64 48 310 703
United Kingdom	+44 207 048 7168
United States	+1 415 735 3300
Singapore	+65 3159 1102

ABN 13 151 896 721